Daniel Marolt

Layout Automation in Analog IC Design with Formalized and Nonformalized Expert Knowledge

Dissertation

Layout Automation in Analog IC Design with Formalized and Nonformalized Expert Knowledge

(Layoutautomatisierung im analogen IC-Entwurf mit formalisiertem und nicht-formalisiertem Expertenwissen)

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von

Daniel Marolt

aus Reutlingen

Hauptberichter:	Prof. Dr. habil. Jörg Schulze
Mitberichter:	Prof. Dr.-Ing. Jürgen Scheible

Tag der mündlichen Prüfung: 04.06.2018

Institut für Halbleitertechnik der Universität Stuttgart

2018

Acknowledgments

A good novel tells us the truth about its hero;
but a bad novel tells us the truth about its author.
Gilbert Keith Chesterton (English writer)

If the dissertation at hand tells us the truth about its author, this is probably my fault. If it tells us the truth about a heroic achievement in electronic design automation, this is due to a lot of honorable people to whom I owe a debt of gratitude. Without their aid, the thought of writing this thesis at Reutlingen University and receiving a doctorate from the University of Stuttgart would have been nothing but a castle in the air.

First of all, I want to thank my doctoral advisors Prof. Dr. habil. Jörg Schulze from the University of Stuttgart and Prof. Dr.-Ing. Jürgen Scheible from Reutlingen University for their unswerving confidence in my ambitions, the steady guidance of my efforts, and the devoted support that helped me bring this work to life and put it down on paper. My compliments also go to my co-examiners Ms. Prof. Dr.-Ing. Nejila Parspour and Prof. Dr.-Ing. Jörg Roth-Stielow for their kind participation in my thesis defense at the University of Stuttgart.

Further tribute shall be paid to Dipl.-Ing. Göran Jerke from the Robert Bosch GmbH, whose valuable expertise and honest mentoring has been shaping my educational path since my earliest days as a rookie student. I would also like to give due credit to Mr. Thomas Burdick and Dipl.-Phys. Peter Herth from Cadence Design Systems for the open-minded exchange of big ideas and the fruitful cooperation that came into existence in the orbit of this work.

Just as well, my colleagues at the Robert Bosch Center for Power Electronics deserve their share of appreciation, be it for words of advice with respect to teething infants or for the constant replenishment of coffee beans. Particular kudos is directed towards my doctoral comrades Andreas Gerlach, Florian Leber, and Matthias Schweikardt for their technical input, collegial teamwork, and high spirits within our research group.

I could not conclude this acknowledgment without taking my hat off to my father, mother, and brother, who not only remained a constant source of motivational inspiration, clean laundry, and humorous quips (respectively), but represent a splendid example for what can be accomplished by putting forth sufficient diligence. The same is true for my in-laws, whose sedulous help –especially regarding our housebuilding and houseworking duties– spared me the time to finish this dissertation.

Last but not least, my deepest gratefulness is dedicated to my wife: for believing in me whenever I start to lose all faith in myself, and for presenting me with gifts that are more precious than any academic title – such as a little child.

Contents

Abstract

Brevity is the soul of wit.
English proverb (from William Shakespeare's Hamlet)

After more than three decades of electronic design automation, most layouts for analog integrated circuits are still handcrafted in a laborious manual fashion today. Obverse to the highly automated synthesis tools in the digital domain (coping with the *quantitative* difficulty of packing more and more components onto a single chip – a desire well known as *More Moore*), analog layout automation struggles with the many diverse and heavily correlated functional requirements that turn the analog design problem into a *More than Moore* challenge. Facing this *qualitative* complexity, seasoned layout engineers rely on their comprehensive *expert knowledge* to consider all *design constraints* that uncompromisingly need to be satisfied. This usually involves both *formally* specified and *nonformally* communicated pieces of expert knowledge, which entails an *explicit* and *implicit* consideration of design constraints, respectively.

Existing automation approaches can be basically divided into *optimization algorithms* (where constraint consideration occurs explicitly) and *procedural generators* (where constraints can only be taken into account implicitly). As investigated in this thesis, these two automation strategies follow two fundamentally different paradigms denoted as *top-down automation* and *bottom-up automation*. The major trait of top-down automation is that it requires a thorough formalization of the problem to enable a self-intelligent solution finding, whereas a bottom-up automatism –controlled by parameters– merely reproduces solutions that have been preconceived by a layout expert in advance. Since the strengths of one paradigm may compensate the weaknesses of the other, it is assumed that a combination of both paradigms –called *bottom-up meets top-down*– has much more potential to tackle the analog design problem in its entirety than either optimization-based or generator-based approaches alone.

Against this background, the thesis at hand presents *Self-organized Wiring and Arrangement of Responsive Modules* (SWARM), an interdisciplinary methodology addressing the design problem with a decentralized multi-agent system. Its basic principle, similar to the roundup of a sheep herd, is to let responsive mobile layout modules (implemented as *context-aware* procedural generators) interact with each other inside a user-defined layout zone. Each module is allowed to autonomously move, rotate and deform itself, while a supervising control organ successively tightens the layout zone to steer the interaction towards increasingly compact (and constraint-compliant) layout arrangements. Considering various principles of self-organization and incorporating ideas from existing decentralized systems, SWARM is able to evoke the phenomenon of *emergence*: although each module only has a limited viewpoint and selfishly pursues its personal objectives, remarkable overall solutions can emerge on the global scale.

Several examples exhibit this emergent behavior in SWARM, and it is particularly interesting that even optimal solutions can arise from the module interaction. Further examples demonstrate SWARM's suitability for *floorplanning* purposes and its application to practical *place-and-route* problems. The latter illustrates how the interacting modules take care of their respective design requirements implicitly (i.e., *bottom-up*) while simultaneously paying respect to high-level constraints (such as the layout outline imposed *top-down* by the supervising control organ). Experimental results show that SWARM can outperform optimization algorithms and procedural generators both in terms of *layout quality* and *design productivity*. From an academic point of view, SWARM's grand achievement is to tap fertile virgin soil for future works on novel *bottom-up meets top-down* automatisms. These may one day be the key to close the automation gap in analog layout design.

Kurzfassung

Nach mehr als drei Jahrzehnten Entwurfsautomatisierung werden die meisten Layouts für analoge integrierte Schaltkreise heute immer noch in aufwändiger Handarbeit entworfen. Gegenüber den hochautomatisierten Synthesewerkzeugen im Digitalbereich (die sich mit dem *quantitativen* Problem auseinandersetzen, mehr und mehr Komponenten auf einem einzelnen Chip unterzubringen – bestens bekannt als *More Moore*) kämpft die analoge Layoutautomatisierung mit den vielen verschiedenen und stark korrelierten funktionalen Anforderungen, die das analoge Entwurfsproblem zu einer *More than Moore* Herausforderung machen. Angesichts dieser *qualitativen* Komplexität bedarf es des umfassenden *Expertenwissens* erfahrener Layouter um sämtliche *Entwurfsconstraints*, die zwingend eingehalten werden müssen, zu berücksichtigen. Meist beinhaltet dies *formal* spezifiziertes als auch *nicht-formal* übermitteltes Expertenwissen, was eine *explizite* bzw. *implizite* Constraint-Berücksichtigung nach sich zieht.

Existierende Automatisierungsansätze können grundsätzlich unterteilt werden in *Optimierungsalgorithmen* (wo die Constraint-Berücksichtigung explizit erfolgt) und *prozedurale Generatoren* (die Constraints nur implizit berücksichtigen können). Wie in dieser Arbeit eruiert wird, folgen diese beiden Automatisierungsstrategien zwei grundlegend unterschiedlichen Paradigmen, bezeichnet als *top-down Automatisierung* und *bottom-up Automatisierung*. Wesentliches Merkmal der top-down Automatisierung ist die Notwendigkeit einer umfassenden Problemformalisierung um eine eigenintelligente Lösungsfindung zu ermöglichen, während ein bottom-up Automatismus –parametergesteuert– lediglich Lösungen reproduziert, die vorab von einem Layoutexperten vorgedacht wurden. Da die Stärken des einen Paradigmas die Schwächen des anderen ausgleichen können, ist anzunehmen, dass eine Kombination beider Paradigmen –genannt *bottom-up meets top-down*– weitaus mehr Potenzial hat, das analoge Entwurfsproblem in seiner Gesamtheit zu lösen als optimierungsbasierte oder generatorbasierte Ansätze für sich allein.

Vor diesem Hintergrund stellt die vorliegende Arbeit *Self-organized Wiring and Arrangement of Responsive Modules* (SWARM) vor, eine interdisziplinäre Methodik, die das Entwurfsproblem mit einem dezentralisierten Multi-Agenten-System angeht. Das Grundprinzip besteht darin, ähnlich dem Zusammentreiben einer Schafherde, reaktionsfähige mobile Layoutmodule (realisiert als *kontextbewusste* prozedurale Generatoren) in einer benutzerdefinierten Layoutzone interagieren zu lassen. Jedes Modul darf sich selbständig bewegen, drehen und verformen, wobei ein übergeordnetes Kontrollorgan die Zone schrittweise verkleinert um die Interaktion auf zunehmend kompakte (und constraintkonforme) Layoutanordnungen hinzulenken. Durch die Berücksichtigung diverser Selbstorganisationsgrundsätze und die Einarbeitung von Ideen bestehender dezentralisierter Systeme ist SWARM in der Lage, das Phänomen der *Emergenz* hervorzurufen: obwohl jedes Modul nur eine begrenzte Sichtweise hat und egoistisch seine eigenen Ziele verfolgt, können sich auf globaler Ebene bemerkenswerte Gesamtlösungen herausbilden.

Mehrere Beispiele veranschaulichen dieses emergente Verhalten in SWARM, wobei besonders interessant ist, dass sogar optimale Lösungen aus der Modulinteraktion entstehen können. Weitere Beispiele demonstrieren SWARMs Eignung zwecks *Floorplanning* sowie die Anwendung auf praktische *Place-and-Route* Probleme. Letzteres verdeutlicht, wie die interagierenden Module ihre jeweiligen Entwurfsanforderungen implizit (also: *bottom-up*) beachten, während sie gleichzeitig High-Level-Constraints berücksichtigen (z.B. die Layoutkontur, die *top-down* vom übergeordneten Kontrollorgan auferlegt wird). Experimentelle Ergebnisse zeigen, dass Optimierungsalgorithmen und prozedurale Generatoren von SWARM sowohl bezüglich *Layoutqualität* als auch *Entwurfsproduktivität* übertroffen werden können. Aus akademischer Sicht besteht SWARMs große Errungenschaft in der Erschließung fruchtbaren Neulands für zukünftige Arbeiten an neuartigen *bottom-up meets top-down* Automatismen. Diese könnten eines Tages der Schlüssel sein, um die Automatisierungslücke im analogen Layoutentwurf zu schließen.

Chapter 1

Introduction

This "telephone" has too many shortcomings to be seriously considered as a means of communication.
Western Union, internal memo (1876)

Electrical engineering is a broad discipline, having come a long way from the early experiments by Benjamin Franklin, Alessandro Volta, André-Marie Ampère and other luminous scientists, over the beginning electrification of the earth in the 1880s, up to the present-day impact of electronic data processing and "new media" on almost every part of our life. A subsidiary but similarly multifaceted area is the field of *electronic design automation* (EDA), to which this dissertation belongs. Therefore, to concretize the subject of the presented work, this chapter deals with the question of what exactly this thesis is about.

1.1 Motivation for this Thesis

Due to an unparalleled technological progress and a lively receptiveness in society, economy and industry, microelectronic products inexorably continue to control, connect, and change our world in the 21$^{\text{st}}$ century. Apart from the triumphal procession of smartphones, contemporary achievements such as activity trackers, delivery drones and electric vehicles, as well as the ongoing innovations regarding bio-implants, driverless cars and the Internet of Things not only address numerous medical, logistical and environmental problems of today, but profoundly revolutionize modern life in many more aspects.

Driven by a market demand for low-cost, multi-purpose, and densely *integrated circuits* (ICs), the semiconductor industry can be seen to follow a trend towards system-on-chip solutions with a growing amount of mixed-signal content, i.e., both digital and analog IC sections. And while digital circuits have long been the main field of interest due to the ever-rising need for more computational power, now analog circuitry also gains notable attention, primarily incited by the desire for functional diversification and system integration. Amongst others, the soaring importance of sensor functionality and the sophistication of advanced human-machine interfaces make analog circuit parts more and more indispensable.

To keep up with increasing chip complexity and shortening product life cycles, the task of creating an IC design asks for electronic design automation, both in the digital and in the analog domain. But although digital IC design already followed highly automated algorithmic synthesis flows early on, analog circuits are commonly still handcrafted by expert designers with only a low grade of automation. Especially the step of *layout design*, where a circuit schematic is to be turned into a physical circuit implementation represents a severely time-consuming bottleneck in the overall design flow (depicted in the simplified overview of Figure 1.1). Why is that so?

The task of putting a system specification into effect as an electronic circuit is a quite creative act. This is particularly true for the step of analog layout design, on which this thesis is focused. Analog layout design significantly depends on the involved designer's expertise, intuition, and creativity, and is therefore considered as an art by many of the experts in this field. This is also reflected by the title of Alan Hastings' *The Art of Analog Layout* [1], which is reckoned a standard work in the analog layout

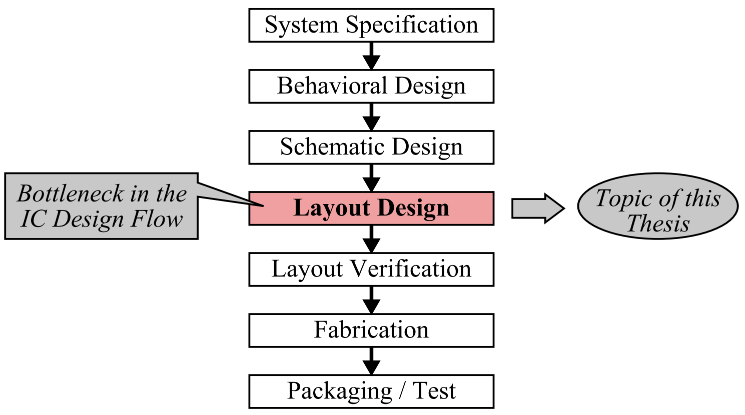


Figure 1.1: Simplified illustration of the integrated circuit design flow.

community. Unfortunately, human qualities such as these cannot be easily automated to a degree that meets industrial requirements. A view on EDA history suggests, that the extent of this observation was not obvious from the very beginning.

The overwhelming success of digital design synthesis since the mid-1980s also immediately sparked a golden age of analog EDA, culminating in a plethora of nameable tools around the year of 1990 which aim at fully-automated analog layout creation. Prominent examples are ILAC [2], LADIES [3], ALSYN [4] and many others, as shown in Figure 1.2. But with the exception of very special application cases, these tools have not turned out to be generally capable of achieving the demanded level of layout quality. This becomes particularly obvious when comprehensive *expert knowledge* is required to take care of the intricate layout details that must be uncompromisingly satisfied in the analog domain to ensure proper electrical performance.

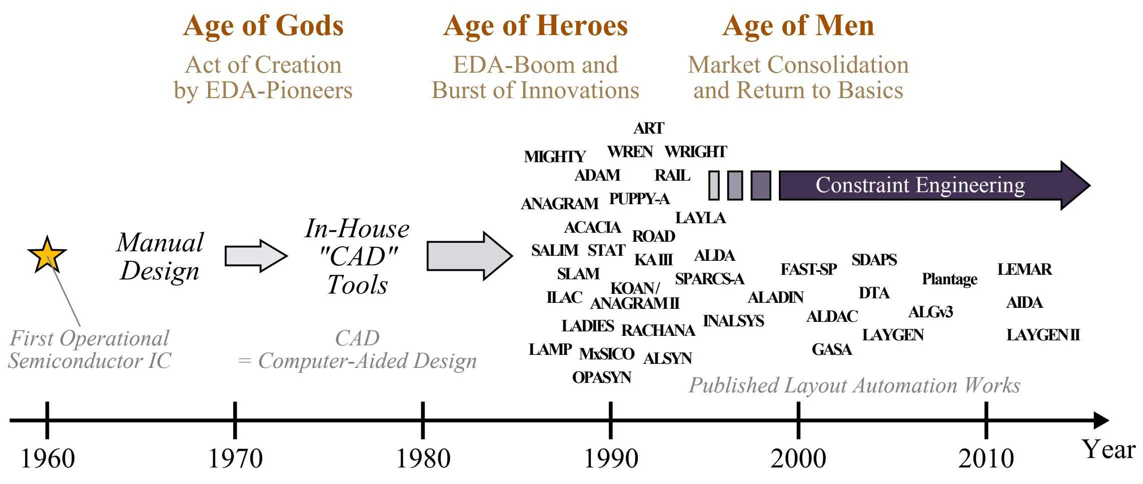


Figure 1.2: The “tides of EDA” with the bloom of analog layout automation systems around 1990.

After this era of euphoria (labeled as the “age of heroes” in the *Tides of EDA* [5], where EDA history is divided into three ages) the realization set in, that analog design automation would not be accomplished as easily as its digital counterpart. Following a sobered renunciation of full-fledged synthesis tools, analog EDA until today primarily concentrates on the incorporation of expert knowledge in the form of *design constraints* – a research field also referred to as *constraint engineering*. Still, despite lively activities and remarkable scientific advancements in EDA, analog layout automation cannot be observed to find evident acceptance in the industry so far. As Rob A. Rutenbar puts it: “Engineers don’t want automated layout tools, because there is an aesthetic involved that serves as a surrogate for correctness” [6]. It is precisely this issue that the thesis at hand means to overcome, as will now be explicated in Section 1.2.

1.2 Contribution of this Thesis

This thesis contributes an innovatively novel layout automation methodology for analog IC design. The particular purpose of the presented methodology is to devise and realize an approach that combines the algorithmic formalisms of digital-inspired synthesis tools with the less tangible aesthetic appeal of analog layout mentioned by Rutenbar (see quotation above). As will be seen, these two ingredients correspond to the notion of *formalized* and *nonformalized* expert knowledge in the title of this dissertation. So far, it is fair to say that there exists no layout automation approach which facilitates a well-balanced consideration of both.

Before commencing with the lecture of the thesis, the reader shall be advised of the following three points:

- *The presented work intends to do the splits between science and industry:*
  As indicated in Section 1.1, the field of analog layout automation has persistently revealed an incredible gap between academic effort and practical assertiveness. This thesis thoroughly examines the roots of that discrepancy to carve out the misconceptions of existing automation approaches. With the conviction that analog layout design cannot be tackled without a decided amount of pragmatism, the developed methodology makes an earnest attempt to be both scientifically valuable and industrially viable.

- *The presented work pursues a fundamentally new idea of layout automation:*
  Many algorithmic automation approaches typically build upon an existing approach and improve it by a certain delta, for example to increase its computational performance or to add some capabilities which allow to handle a special problem. In contrast, the idea behind the methodology discussed in this thesis can be considered to be quite novel in a more profound way. This is reflected in the extent of the description needed to cover the methodology in its entirety.

- *The presented work interdisciplinarily joins ideas and concepts from different fields:*
  The developed methodology is the first one to address the problem of analog layout automation with a system of individual, self-organizing layout modules. As such, this thesis not only sits at the junction between electrical engineering and computer science, but crosses the boundaries of various areas related to multi-agent systems. Inspired by natural phenomena in biology and physics, these areas include software agents and artificial life, complexity theory and chaos theory, synergetics and game theory, cybernetics and systems theory, as well as mathematics in general and geometry in particular. An extra chapter is spent to pave the way for an understanding of the methodology and its interdisciplinary approach.

It is also for these reasons that the thesis at hand has become relatively copious in volume. The following Section 1.3 gives a survey of all chapters in order to outline how this thesis is structured.

1.3 Outline

The thesis is divided into three parts, each of which comprises three chapters.

Part I

Roughly speaking, the first part focuses on the problem that the thesis deals with. After covering some relevant basics of analog layout design, Chapter 2 underlines the necessity for an automation methodology that is able to consider both formalized and nonformalized expert knowledge. Taking a look at the state of the art, Chapter 3 contemplates existing automation approaches first from a more concrete and then from a rather abstract point of view, thereby discerning two distinct automation paradigms. While the challenge of teaming these two paradigms represents the academically tough nut to crack, Chapter 4 is dedicated to the question by which criteria the practical viability of an analog layout methodology can be assessed.

Part II

The second part is the main body that explicates the automation methodology developed in this work. The reader may immediately jump here after merely skimming through the first part, since Chapter 5 recapitulates the technical, academical, and practical objectives discussed therein. Next, Chapter 6 deduces the multi-agent approach of the developed methodology from the basic idea of decentralization, giving insight into some astonishing capabilities of decentral organisms in nature and describing a couple of relevant engineering concepts based on decentralized systems. Chapter 7 represents the core chapter of the thesis and provides a comprehensive breakdown of the new methodology, starting off with an illustrative overview before elaborating on the nuts and bolts in detail.

Part III

The third part of this thesis puts the developed methodology into action. Chapter 8 depicts its practical implementation in an industrial IC design environment, demonstrating various different examples and evaluating the obtained results. Chapter 9 places the presented work in a greater context and discusses its incorporation into the overarching vision of a seamless "higher-level" design flow of which this methodology represents one piece. Finally, Chapter 10 concludes the thesis with a summary and an outlook. A vocabulary at the very end lets the reader look up certain acronyms and technical terms, separating those that are well-established in the field of EDA from those that are newly introduced in this thesis.

Part I

The Problem

Chapter 2

The Aim of this Thesis

. . . and yet a true creator is necessity,
which is the mother of our invention.
Plato: The Republic, Book II, 369c

As already stated in the introduction, the essential aim of this thesis is to contribute a new methodology for analog layout automation which is able to take *formalized* as well as *nonformalized* expert knowledge into account. Taking a closer look at the analog layout design problem, this chapter provides the economic motivation and the technical background required to comprehend the necessity for such a methodology. In the course of the subsequent discussion from Section 2.1 to Section 2.5, it will be explained that taking both formalized and nonformalized expert knowledge into account is synonymous to an *explicit* and *implicit* consideration of *design constraints*.

2.1 The Problem of Analog Layout Design

This section covers certain aspects of analog layout design that are important for an understanding of the presented work, and introduces technical terms that will be referred to in the remainder of this thesis. As mentioned in Section 1.1, the problem of *layout design* is to take a given electronic circuit and turn it into a physical representation, which is itself also called a layout design.[1] The purpose of that physical representation is to describe the detailed chip geometries on the photolithographical masks which need to be created for the various layers of the semiconductor manufacturing process. These geometries include the layout of the circuit components and their electrical interconnections with interconnect holes between isolating layers, as well as the so-called bondpads for the chip's connections to its periphery, and all other geometries such as inscriptions, logos and test structures.

Although the input circuit could be provided in the abstract form of a mere netlist, it is commonly given as a visually graspable schematic diagram. This can be put down to two major reasons:

- In contrast to the digital domain, where circuits are synthesized from high-level specifications, analog circuits are manually created by design experts. This necessitates a convenient display and editing format.
- Turning an electronic circuit into a functioning layout involves the consideration of many design restrictions and design objectives which arise from an understanding of the circuit's function. Thus, the readability of a schematic diagram is not only irremissible for the circuit creator but also for the layout designer.

The topic of design restrictions and design objectives will now be further addressed in Section 2.1.1. Then, Section 2.1.2 deals with a very important principle called *matching* while Section 2.1.3 is dedicated

[1] So, the term *layout design* is ambiguous since it can denote either a resulting layout or the process of creating one. In the remainder of this thesis, *layout design* is only used in the latter sense while the design result is simply referred to as a *layout*.

to hierarchy levels in analog layout design, before Section 2.1.4 discusses the main design tasks called device generation, floorplanning, placement, and routing.

2.1.1 Design Restrictions and Design Objectives

From a mathematical point of view, layout design is an *optimization problem* and can be regarded as a search for an optimal solution inside an immensely huge *solution space*. Thereat, design restrictions define a valid region in the solution space, while the design objectives specify an optimum inside that valid region, as depicted in [7]. Design restrictions are commonly divided into three categories:

Technological restrictions are meant to ensure the manufacturability of the integrated circuit. They are derived from the chosen semiconductor technology and formulated as geometrical *design rules*. Design rules can be very complex, but most of them belong to one of the following groups: minimum width, minimum distance, minimum overlap, or minimum enclosure.

Functional restrictions (also referred to as electrical restrictions) are supposed to guarantee the circuit's proper electrical functioning. They can be separated into circuit-specific requirements (e.g., to prevent unwanted coupling effects) and process-specific requirements (such as the limitation of current density in electrical wires to avoid electromigration).

Design-methodical restrictions are deliberately introduced to reduce the complexity of the layout design problem, thereby making the design task amenable to computer-aided automation approaches. An example is given by layer-dependent wire directions for the purpose of automated routing (e.g., metal1: horizontal, metal2: vertical).

While design restrictions are *strict confinements* which (apart from the design-methodical restrictions) must definitely be satisfied, design objectives represent gradual *optimization goals* that are pursued as good as possible. They can be roughly classified into economic optimization goals and functional optimization goals. Economic optimization goals include the reduction of product costs (e.g., by minimizing the total chip area and the number of required metalization layers) as well as reducing the development costs (e.g., by minimizing the design effort via design automation). Examples for functional optimization goals are the minimization of the total wirelength as well as optimizing the chip's heat dissipation to prevent critical hot spots.

As the name implies, functional restrictions and functional optimization goals pertain to the functionality of an integrated circuit. Herein, three basic issues can be embraced by the term *functionality*:

- Does the circuit work accurately enough to perform the desired function?
- How well is the circuit set up against long-term failure owing to effects of degradation?
- What measures are taken to prevent an instantaneous malfunctioning due to fabrication problems?

The latter two issues will be postponed to Chapter 4 since they can be considered secondary to the first issue: accuracy. In the analog domain, accurate circuit behavior is of utmost importance. This is rooted in the continuous-valued nature of analog signals, where even the slightest derogation from a circuit's nominal operating point can have a critical impact on its functionality. For that reason, many functional restrictions and functional optimization goals –such as minimizing the distance between certain layout components, for example– are pursued in favor of an essential and very characteristic principle of analog design called *matching*, which is to be discussed in Section 2.1.2.

2.1.2 Matching

In short terms, matching is employed to obtain analog signal accuracy via "electrical symmetry". This can be understood as follows. The manufacturing tolerances exhibited by the many steps of an IC fabrication process are so large that their cumulative effect causes an unbearably high deviation in a component's parameters from their desired values. Hence, one can say that the absolute exactness of a circuit's components is extremely poor. However, the parametrical deviation among circuit components of the same type are comparable. This means that the relative exactness of components from the same manufacturing cycle (and even moreso regarding components on the same chip) are rather good [8].

In the analog domain, circuits are specifically designed and layouted in a way which takes advantage of that relative exactness by "matching" certain components. With this technique, the electrical behavior of these components can be effectively equalized in relation to each other so that inevitable variations (not only manufacturing tolerances, but also other derogating influences such as thermal gradients and parasitic effects) don't affect their overall electrical functioning. This principle asks for (1) identifying circuit components which should work electrically symmetric, and for (2) creating the layout such that it attains that symmetric behavior. Without this technique, obtaining a functional analog integrated circuit would be virtually impossible. Hence, achieving a good matching is one of the most important duties in the daily work of analog layout designers.

The matching of circuit components in general requires that these components are of the same type and that they have equal dimensions. Those necessities already have to be taken care of by the circuit designer. Further matching measures in the layout often demand to place these components with consistent orientations and to align them in a compact, interdigitated, common centroid arrangement [9] (for an example, see Figure 3.14). As will be discussed in Chapter 3, matching also represents the primary concern –and the major difficulty– for state-of-the-art layout automation approaches.

Typically, matching is demanded for circuit components which belong closely together because they serve a dedicated and impartible electrical function. This is reflected in the practice of encapsulating such components in a compound *module*, which leads over to Section 2.1.3.

2.1.3 Levels of Design Hierarchy

Analog IC layouts are usually built in a hierarchical fashion, which is achieved by putting design components inside other design components. Prime design entities such as transistors are commonly given as *primitive devices*, i.e., fundamental *cells* that do not have a subhierarchy. On higher levels, a group of design entities which belong together –and thus form a functional unit– can also be encapsulated as a single (but in this case hierarchical) cell. Like a primitive device, the functional unit thus becomes a modular library component that can be *instantiated* in a layout (i.e., inside another cell).

To avoid semantical confusion when speaking of cells, it is appropriate to classify them with respect to the characteristics that they exhibit depending on their location in the design hierarchy. For that purpose, this thesis proposes and adheres to the following terminology:

Primitive devices are the bottom-level components in layout design and consist of plain polygons. For the most part, they realize transistors, resistors, and capacitors. Other structures such as wells and guardrings are often also implemented as primitive devices.

Simple modules represent well-established *analog basic circuits* with highly regular patterns. This means that a simple module contains a set of identical primitive devices that are put together in a strictly tiled, matrix-like arrangement.

Advanced modules also represent analog basic circuits, albeit being composed of different components (primitive devices and simple modules). The components' relative positions follow a well-established, often symmetric arrangement, notwithstanding their actual dimensions.

Blocks are large hierarchical cells that perform high-level electrical functions. They consist of unequal devices and modules, assembled in a rather irregular arrangement that has to be individually determined by a layout expert in the respective case.

Chip denotes the top-level entity of a layout. It is made of hierarchical layout blocks (in case of a mixed-signal chip: both analog and digital blocks) but can also contain single devices such as power transistors that realize high-current output stages. Although a family of chips can share topological similarities, a chip is always a unique design.

The proposed cell taxonomy is also displayed in Table 2.1, which gives a couple of examples for each kind of cell. As indicated in the table, it can be said that the degree of (re-)utilization is basically reciprocal to the level in the design hierarchy.

Table 2.1: Classification of hierarchical cells in analog/mixed-signal design.

	Hierarchy Level	Examples	Degree of (Re-)Utilization
Chip	Top Level	Airbag Sensor Evaluation Circuits, ABS/ESP Brake Control Systems, Motor Drivers, Audio Amplifiers	very low
Blocks	Block Level	Analog-Digital Converters, Digital-Analog Converters, Bandgaps, Voltage Regulators	low
Advanced Modules	Module Level	Operational Transconductance Amplifiers Differential Amplifiers, Comparators	medium
Simple Modules	Module Level	Differential Pairs, Current Mirrors, Resistor Arrays, Capacitor Arrays	high
Primitive Devices	Device Level	Transistors, Resistors, Capacitors, Guardrings, Isolation Tanks	very high

2.1.4 Main Design Tasks: Device Generation, Floorplanning, Placement, Routing

As already indicated, creating an analog layout involves several tasks. The main design tasks are denoted as *device generation*, *floorplanning*, *placement* and *routing*.

Device generation is the task of creating the layouts for the individual components of the given input circuit. Herefor, every component needs to be individually layouted according to its respective sizing (e.g., the channel width and channel length of a MOS transistor). In the past, this task has been an integral duty on the shoulders of an IC design team. Today it is common practice that the primitive devices of a semiconductor technology are readily delivered by the vendor as part of a so-called *process design kit* (PDK) and usually implemented as *procedural generators* (which will be the topic of Section 3.1.2). With the aid of such generators, many layout components come in a multitude of different possible layout variants. Contrasting the use of standard cells in the digital domain, this variability represents an essential trait of analog design which is indispensable for being able to satisfy the many different restrictions and objectives of the design problem. Even primitive devices have an immense layout variability, and one major source of this variability is *device folding*. For example, a native MOS transistor can be folded by changing its so-called *number of fingers*. As shown in Figure 2.1, the transistor variants thus have different aspect ratios while preserving the total channel width and channel length. Device generation is already important during floorplanning for estimating the total size of a layout block.

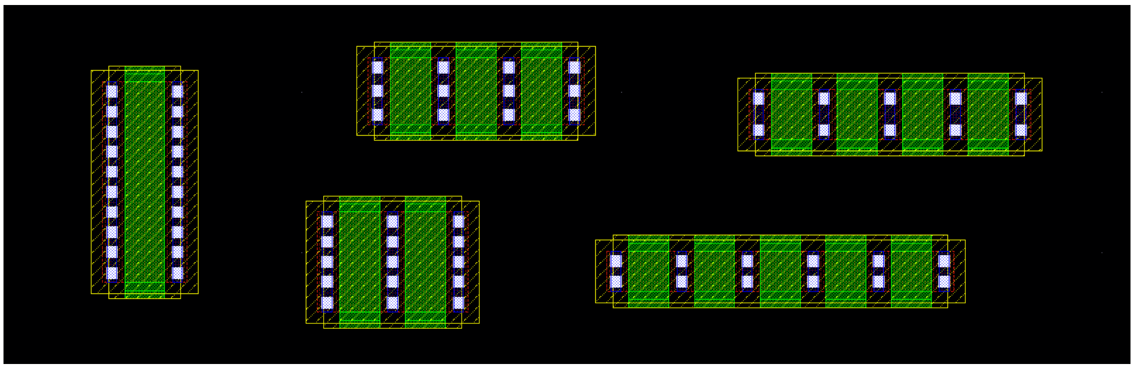


Figure 2.1: Different layout variants of a MOS transistor with the same total channel width and length.

Floorplanning, as listed in Table 2.2 (a), is the task of specifying locations, aspect ratios, and pin positions for the layout blocks of a chip. Therein, each block is treated as a black box whose area is roughly estimated by the floorplan designer from generating the block's devices. For economical and electrical reasons, the primary objectives in floorplanning are to minimize the total layout area and the total wirelength, as well as to optimize the power supply and the current flow. A hard restriction concerning the layout area can be that the blocks must fit into a fixed outline, depending on the semiconductor *package* chosen for the physical sealing of the chip during the final stage of the fabrication. In general, the top-level chip boundary is demanded to be a rectangle whose aspect ratio should not depart too far from a square. In the context of wirelength minimization, some blocks are required to be positioned close to the chip boundary because they will later be connected with the periphery. On the other hand, it may also be necessary to keep a certain minimal distance between dedicated blocks such that sensitive signals are not disturbed by unwanted thermal and electrical influences. A large block can contain subordinate blocks that also need to be floorplanned.

Table 2.2: The main tasks in analog layout design: (a) floorplanning, (b) placement, (c) routing.

	(a) Floorplanning	**(b) Placement**	**(c) Routing**
Considered Components	Circuit Blocks (treated as black boxes)	Primitive Devices and Modules	Wire Segments + Vias (to cross metal layers)
Quantities to be set by the Design Task	Block Locations Aspect Ratios Pin Positions	Locations Orientations Layout Variants	Wire Paths, Segment Layers and Widths + Via Positions and Sizes
Typical Restrictions	Rect. Chip Outline Block Distances Chip Regions	Block Outline Space for Routing Parasitics	No Wires Above Devices Available Metal Layers Parasitics and Currents
Primary Objectives	Minimize Area and Wire-length, Optimize Power Supply and Current Flow	Device Matching Overall Symmetry	Minimize Number of Vias and Num. of Metal Layers Homogenize Wire Density

After floorplanning, the internal full-custom layout of each block has to be created. This is done by first of all generating the primitive devices in the layout block, as given by the circuit schematic. Provided that procedural generators are available for all these devices, the mere layout creation for the primitive devices proceeds in an automated way – a functionality commonly referred to as *Generate from Source*. However, this does not release the designer from the duty of deciding upon the variant that each device is to assume. That is done during the challenging tasks of placement and routing which follow this initial device generation. Thereby, devices that belong together must be brought together and then need to be connected with each other. In turn, these modules are then also placed and routed until the overall block layout is complete.

Placement –see Table 2.2 (b)– not only means to move the layout components into appropriate locations, but may also require to rotate them and to vary their layout without affecting their electrical function (for example by changing the *number of fingers* discussed above), such that all components fit into the block outline defined during the floorplanning phase. As in floorplanning, it is desired to obtain a device placement wherein the total area and wirelength are minimized, but usually these objectives first and foremost stem from the need to obtain a good device matching. The same is true for higher-level modules whose placement aims at achieving an overall symmetry of the layout block where the modules are placed in. Opposing the need to place the components close to each other, a common demand is that a sufficient amount of space must be reserved between the components to accommodate their routing.

The demand to allow for routing space between layout components is rooted in the fact that sensitive circuitry often forbids electrical wires to be drawn above these components. As mentioned in Table 2.2 (c), this has to be taken into account during the routing task. Another restriction is to confine which metal layers are available for the routing. On the lower design levels, a common agreement is that only the first two (i.e., the two bottommost) of the available metal layers may be used in order to retain the remaining metal layers for the later top-level routing. Leading a wire across different metal layers requires to connect the respective wire segments with a *via* (vertical interconnect access). The size of a via and the width of a wire segment must be set with respect to the expected current load. In general, analog layout designers refrain from vias with only one clearance hole, but insist on using so-called *double-cut vias* that are large enough to enclose at least two clearance holes. Primary objectives in routing are to minimize the number of vias (i.e., to avoid crossing between metal layers if possible), to minimize the number of metal layers (and thus the number of necessary photolithographical masks), and to homogenize the overall wire density. In some cases, the routing of certain nets is also demanded to be symmetric to each other.

In digital design, the different design tasks are mostly separated from each other and may be even further divided (e.g., routing is usually performed in two steps called global routing and detailed routing). By contrast, the tasks of device generation, floorplanning, placement, and routing are heavily interrelated in the analog domain. This represents a significant obstacle for automation approaches and contributes to the problem that analog layout design is a bottleneck in the overall design flow, as Section 2.2 is about to point out.

2.2 Analog Layout Design: A Bottleneck in the Design Flow

Since layout design is an *optimization problem*, it can –in principle– be automatically solved using *optimization algorithms* [10]. In the well-established synthesis flows of digital IC design, such optimization algorithms (which will be covered in Section 3.1.1) are successfully employed to place and route millions and billions of logic gates per chip. Encouraged by this success, EDA keeps a resolute focus on trying to adopt suchlike approaches in the analog domain. But despite an enormous amount of research and development work over the past three decades, optimization-based automation has repeatedly met with disapproval among the analog design community and is still struggling to find its way into industrial environments. Thus, layout design is the step of the analog design flow with the least support by commercially available tools [11].

Due to this rejection of the many existing automation approaches, analog layouts in practice are predominantly still engineered by human experts in a laborious and largely manual fashion, putting up with the downside that the design productivity is significantly lower than in the digital domain. Actually, this can be acknowledged in two regards: (1) the effort for creating an analog layout is much *higher* than for a digital layout, although (2) the number of design components is usually *smaller* by several orders of magnitude. That circumstance, also known as the *mixed-signal design problem* [12], is illustrated in Figure 2.2.

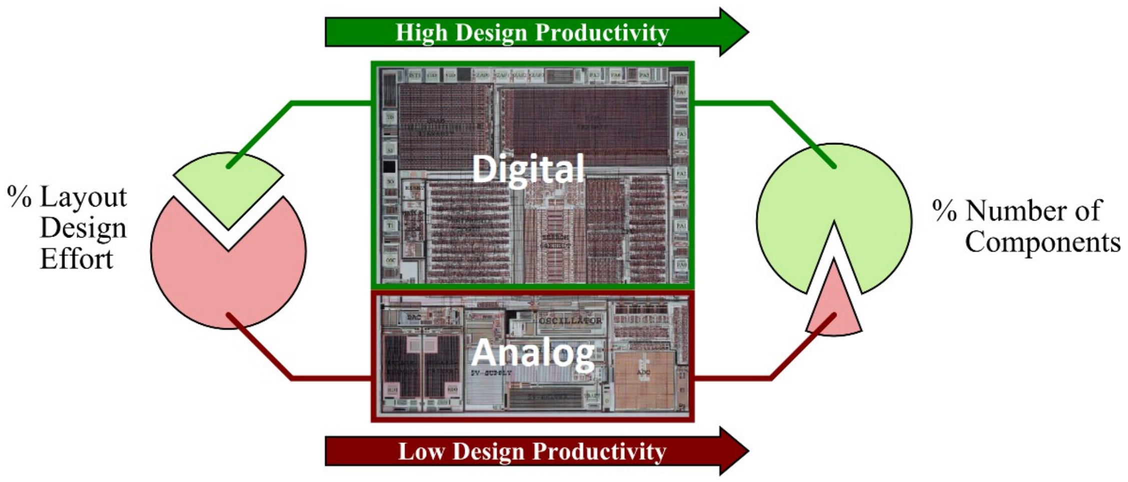


Figure 2.2: The mixed-signal design problem [12]: in the analog domain, the layout design productivity is significantly lower than in the digital domain.

So, while optimization algorithms are effectively employed for digital design synthesis, they do not find as much acceptance in the analog domain even though the conceded lack of automation involves a severe economic penalty. The detriment of this bottleneck for the overall design flow is even more worrisome when considering that time-to-market continually decreases due to shortening product life cycles and price deflation [13]. Still, in contrast to the digital domain, the persistent skepticism towards analog design automation remains, and as will be discussed in Section 2.3, this fact can be put down to entirely different design prerequisites originating from the disparate purposes of digital and analog circuitry.

2.3 The Design Style and the Design Flow – Digital vs. Analog

Creating a layout is a very complex problem.[2] But in terms of this complexity, one should clearly distinguish between the discrete-valued nature of the digital domain and the continuous-valued nature of the analog domain, considering the respective purposes of the two domains in IC design as illustrated in Figure 2.3. These different purposes highly affect the corresponding *design style* and inherently the corresponding *design flow* of digital and analog layout design, as will be explained in Section 2.3.1 and Section 2.3.2 respectively.

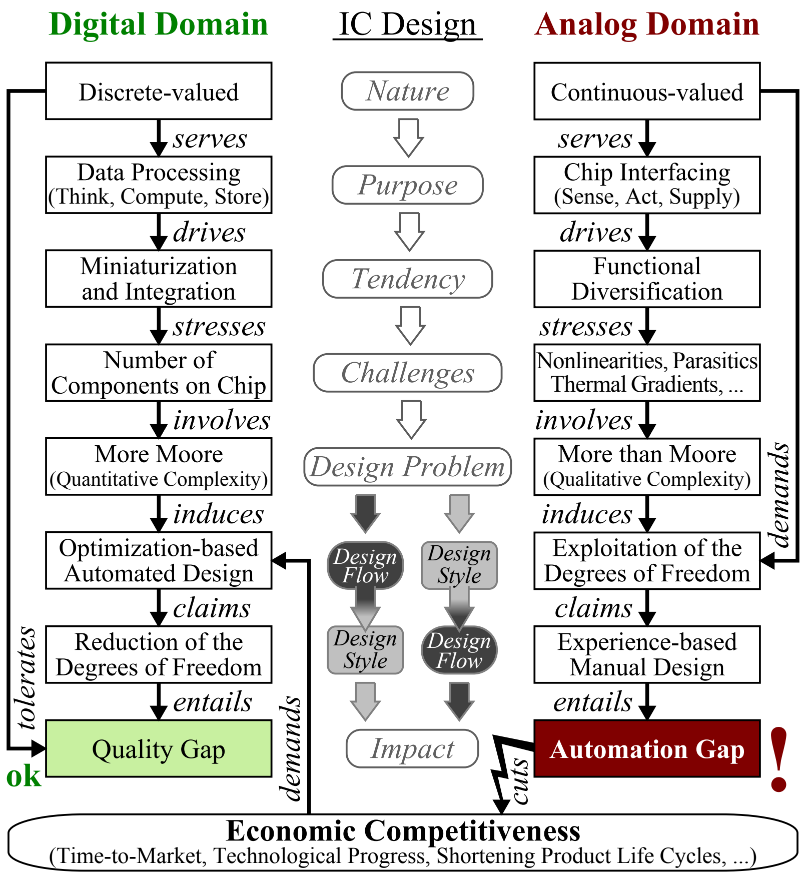

Digital Domain
IC Design
Analog Domain
Discrete-valued
Nature
Continuous-valued
serves
serves
Data Processing (Think, Compute, Store)
Purpose
Chip Interfacing (Sense, Act, Supply)
drives
drives
Miniaturization and Integration
Tendency
Functional Diversification
stresses
stresses
Number of Components on Chip
Challenges
Nonlinearities, Parasitics Thermal Gradients, ...
involves
involves
More Moore (Quantitative Complexity)
Design Problem
More than Moore (Qualitative Complexity)
induces
induces
Optimization-based Automated Design
Design Flow
Design Style
Exploitation of the Degrees of Freedom
demands
claims
claims
Reduction of the Degrees of Freedom
Design Style
Design Flow
Experience-based Manual Design
entails
entails
tolerates
Quality Gap
Impact
Automation Gap
ok
demands
cuts
!
Economic Competitiveness
(Time-to-Market, Technological Progress, Shortening Product Life Cycles, ...)

Figure 2.3: In the digital domain, the design flow determines the design style. In the analog domain, the design style determines the design flow.

2.3.1 The Digital Design Style: Standardized

A conventional mixed-signal chip with both analog and digital content essentially takes a set of analog input signals from its environment, converts them into discrete signals, processes them through digital logic, and converts the processed data into analog output signals again to perform a certain function. The data processing abilities of digital IC sections primarily depend on the amount of implemented logic, thus driving the ongoing miniaturization and integration of microelectronic devices. Referring to Moore's Law [15], this challenge of putting more and more components onto an integrated circuit has

[2]Computationally, the problem of layout design is NP-hard [14]. This means that no known algorithm can ensure to find a globally optimal solution in polynomial time.

become well-known as *More Moore* [16], which emphasizes that the respective design complexity in creating the physical layout is mainly a matter of *quantity*. Without support from design automation, creating the layout for a modern digital IC that contains millions or billions of logic gates is not just a question of economic efficiency, but is virtually impossible.

So, the *quantitative complexity* in digital layout design is addressed via automation approaches based on optimization algorithms, but it should be noted that their practical application upon large-scale circuits requires a considerable simplification of the overall design problem. As will be detailed in Section 3.1.1, this is achieved by (1) abstracting the design problem –via (1a) a translation of the design problem into a mathematical model and (1b) the introduction of design-methodical restrictions for the sake of standardization, such as the use of fixed-height standard cells and the placement of standard cells in rows– as well as (2) resorting to heuristics in order to trade optimality and completeness for runtime reduction (also see **EDA View** in Figure 3.1 on page 33).

The mentioned simplification of the design problem benefits the optimization algorithm because it reduces the *degrees of freedom* and diminishes the *solution space*. On the other hand, such a simplification entails a loss of layout quality, because even an algorithmically optimal solution (which is not even guaranteed to be found in case heuristics are used) is electrically suboptimal in reality. Yet, this *quality gap* can be tolerated due to the discrete-valued nature of digital signals, and so the overall automation strategy sustains itself (Figure 2.3, left). Therefore, in digital layout design the design flow determines the design style, which is commonly referred to as *standardized* or *semi-custom* design [17].

2.3.2 The Analog Design Style: Full-custom

While digital sections can be thought of as the "brain" of an integrated circuit, analog parts represent the indispensable interface of an IC to its continuous-valued environment and also serve as the subsystem for powering the chip. In contrast to the persistent downscaling in the digital domain, analog content[3] keeps up an eager interest to expand the functional diversity of ICs throughout and beyond all facets of sensing and acting and to drive the implementation of system-on-chip designs. Reflecting the immanent challenge to handle nonlinearities, parasitic coupling, thermal gradients, high voltages, external physical influences and many other intricate effects, this desire is called *More than Moore* [16] and makes the complexity of analog layout design –in contrast to its digital counterpart– rather an issue of *quality*.

In terms of this *qualitative complexity*, maintaining the integrity of analog signal transmission calls for an optimal layout that comprehensively utilizes the entire spectrum and variety of all available degrees of freedom [18]. For the most part, this –in turn– opposes optimization-based design automation because an abstraction of the design problem and a lack of adequate heuristics cause the resulting layout solutions to be insufficient for practical application. So, the loss of layout quality that is condoned in digital design is precisely what cannot be tolerated in the analog domain due to the delicate sensitivities of its continuous-valued signals.

In a nutshell, the need to *exploit* all degrees of freedom defies the use of optimization algorithms because these require a *reduction* of the degrees of freedom. For that reason, analog layout design is still done in a highly manual way and relies heavily on the knowledge, experience, skills, and inventiveness of human experts. So, the lack of automation –the *automation gap*– is tolerated as the lesser of two evils, since layout quality does not permit any tradeoffs (Figure 2.3, right). Thus, entirely opposite to the digital domain, the design style –which is commonly referred to as *full-custom* design [19]– determines the design flow, despite the inherent economic penalties with respect to design productivity. In other words: the demand for automation is overruled by the layout quality requirements – and, as follows in Section 2.4, these are intimately tied to the so-called *design constraints*.

[3]More generally, one should speak of non-digital content instead of analog content here, because it may also involve micromechanical and microfluidic devices, for example.

2.4 Constraints: Formalized and Nonformalized Expert Knowledge

The quality of a layout depends on its satisfaction of the functional design restrictions and functional design objectives. In the remainder of this thesis, such restrictions and objectives are subsumed under the term *design constraints* (or just *constraints*).[4] These constraints are the fulcrum: with the necessity of meeting the extremely high accuracy demands of continuous-valued analog signals –encumbered by the More than Moore challenges discussed before– determining the constraints and satisfying them is the major difficulty in analog layout design. This also represents the Achilles heel of design automation, since the constraints are where optimization algorithms commonly fail to achieve the demanded level of layout quality.

In contrast to other publications where the term *constraint* is used to denote various kinds of mandatory restrictions, this work's definition of constraints decidedly refers to circuit *functionality* and specifically includes not only strict confinements but also "nice-to-have" requests and optimization goals.[5] The practical reason is that such an understanding of constraints can already be found in the common parlance of analog designers. More importantly, this also reflects the day-to-day business of real-life productive design flows, where constraints are sometimes concisely described in a constraint management system but may also appear in a rather loose (i.e., not computer-assisted) fashion, covering a broad range of rigidity from "strongly required" to "only desired" and "as good as possible".

Although strict confinements and optimization goals are entirely different constructs from a mathematical and logical point of view, in analog IC design the dividing line is rather blurred (also see Figure 2.6). Apart from this, the above notion of constraints in this thesis is further legitimated by a look at other disciplines. For example, in artificial intelligence the term *soft constraints* is used to denote restrictions whose satisfaction is not mandatory [20] or to denote optimization goals with a certain desirability [21], whereas strict confinements (i.e., restrictions which *must* be satisfied) are denoted as *hard constraints* [22].

To put it another way, a constraint in analog IC design can be considered as an information (e.g., a restriction, objective, requirement, desire, request, instruction, or intention) not contained in a given schematic circuit, but relevant to obtain a corresponding layout that achieves the intended circuit functionality. Against this backdrop, the problem of layout design may be regarded as shown in Figure 2.4, which underlines the integral role of constraints from the viewpoint that any layout design problem can be said to expect three inputs: (1) a set of design rules, i.e., the technological restrictions, (2) a structural description of the circuit, i.e., a netlist or preferably a schematic diagram, and (3) the design constraints, i.e., the functional restrictions and objectives. Correspondingly, there are three mandatory obligations for the resulting layout, relating to the three given inputs: the layout must (1) adhere to the design rules, (2) match the given circuit, and (3) satisfy all design constraints.

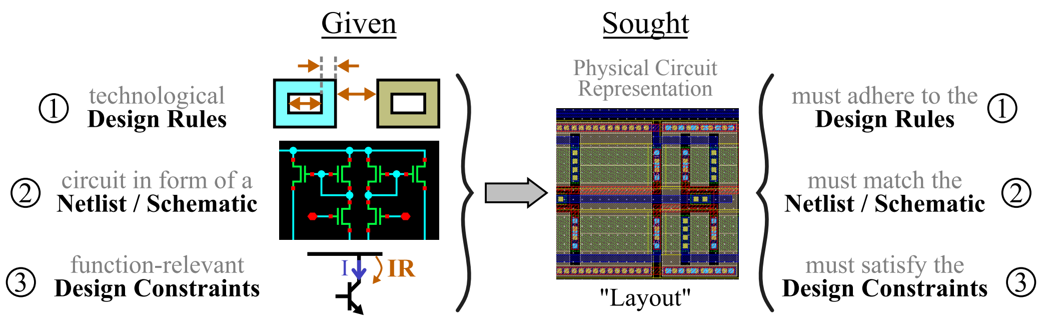

Given
Sought
① technological Design Rules
② circuit in form of a Netlist / Schematic
③ function-relevant Design Constraints
I
IR
Physical Circuit Representation
"Layout"
must adhere to the Design Rules ①
must match the Netlist / Schematic ②
must satisfy the Design Constraints ③

Figure 2.4: The layout design problem of turning an electronic circuit into a physical representation.

As a side note, the difficulty of satisfying the constraints in analog layout design is associated with the problem that constraint compliance cannot even be fully verified today due to a lack of descriptiveness. The situation is quite different with obligations (1) and (2) in Figure 2.4: whether a layout adheres to

[4]Thus, the notion of satisfying a constraint can either imply the question *if* the constraint is satisfied but also *how well* the constraint is satisfied.

[5]Still, whenever a distinction is necessary, this will be clearly articulated in this thesis.

the design rules can be verified with a *design rule check* (DRC), while a *layout versus schematic* (LVS) check determines if the layout matches the schematic circuit. Both kinds of checks are part and parcel of the design flow.[6] In modern IC design frameworks, the equality of circuit and layout is already asserted during design (albeit on a symbolic level), which defines the so-called *schematic-driven layout* (SDL) design flows that are common practice today. And although concepts for constraint management and constraint verification are still in their infancy compared to DRC and LVS checks, it is generally agreed that establishing a *constraint-driven design* flow is the next logical step in analog IC design [7][23]. However, this aim is not easy to achieve due to the diffuse nature of constraints.

Technically, constraints are rooted in the type, purpose, and application of the intended circuit, since these attributes define the circuit's particular functional requirements in terms of accuracy and robustness. However, a designer's ability to grasp all relevant constraints depends on how familiar the designer is with the functionality of the circuit and its possible pitfalls, and also how versed the designer is in layout creation (especially concerning the respective semiconductor technology). In that regard, constraints can be considered as manifestations of *expert knowledge*.

Along with the circuit schematic, all respective constraints must somehow be communicated from the circuit designer to the layout designer (if the circuit designer is also responsible for creating the layout, then of course the constraints need not be communicated between different people – however, the constraints must at least be mentally articulated by the designer). In doing so, it can be observed that for every constraint, this can be done either in a more formal or in a rather informal way, and that in today's flows of manual layout design, both ways are encountered in equal measure. While there is not a unique, distinguishing mark to determine whether a constraint appears in a *formalized* or *nonformalized* representation, a distinction can be made according to several characteristics. A formalized constraint is a constraint representation which

(1) expresses a clear and definite condition or relation,
(2) is unambiguous in that it leaves no room for interpretations,
(3) can be checked mathematically, logically, or algorithmically,
(4) is typically –but not necessarily– stored in a specific, computer-aided format, e.g., in the constraint database of a constraint management system included in the IC design framework.

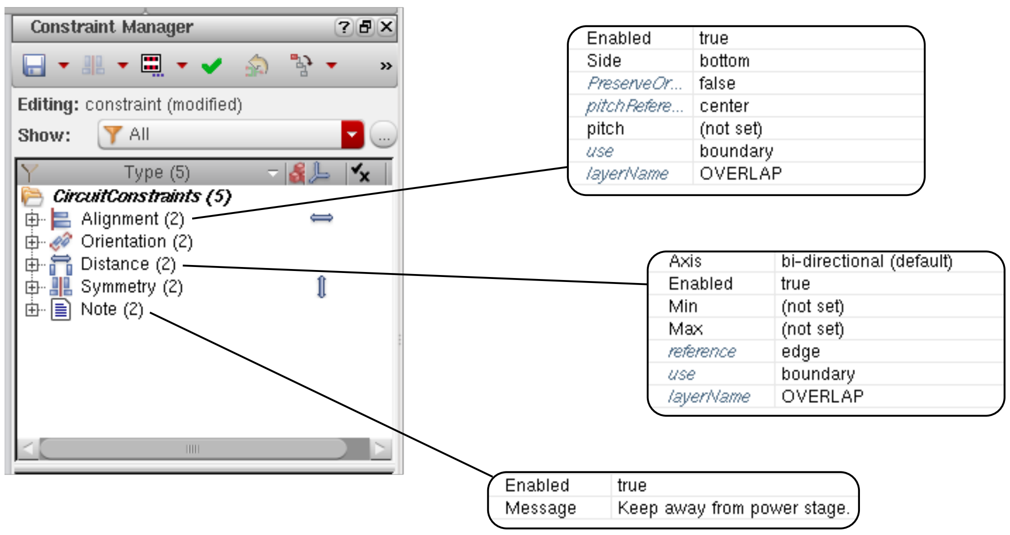
In contrast, a nonformalized constraint is a constraint representation which

(1) may fuzzily articulate any kind of functionality-relevant information,
(2) requires a semantical understanding and depends on the comprehension by the layout designer,
(3) can hardly be checked computationally but rather only according to the opinion of a human expert,
(4) is typically –but not necessarily– communicated to the layout designer via verbal conversation, textual notes, or prosaic labels in the schematic diagram.

As an example, Figure 2.5 shows a couple of constraints entered into the constraint management system of [24]. Most of these constraints (here those of type Alignment, Orientation, Distance, and Symmetry) are thoroughly formalized and allow to be stated even more precisely by setting constraint-dependent options (e.g., a certain edge for alignment or a maximum bound for the distance). However, the fact that a constraint is stored in such a tool-assisted way does not necessarily signify a formalized representation: for example, the constraint can just as well be only a prosaic comment (as is the case with constraint type Note in Figure 2.5) and thus not be formalized at all.

Figure 2.6 shows several examples of nonformalized constraints, taken from actual designs of an IC layout engineering group for automotive electronics, where these constraints are stored as text labels in the schematic diagrams. Although all constraints are prosaic (and are therefore not directly processible computationally), those marked with an asterisk (*) can be considered to be formalized constraints because they are quite unmistakable. However, most constraints are nonformalized, namely those constraints marked with a dagger (†). For example, some of these constraints imply several other requirements without clearly stating them (e.g., the requirements to achieve a good matching), while others articulate such an imprecise restriction that it is definitely up to the understanding of the layout expert

[6]In fact, a finished layout *must* pass both checks, otherwise it is rejected by the semiconductor fabrication plant.



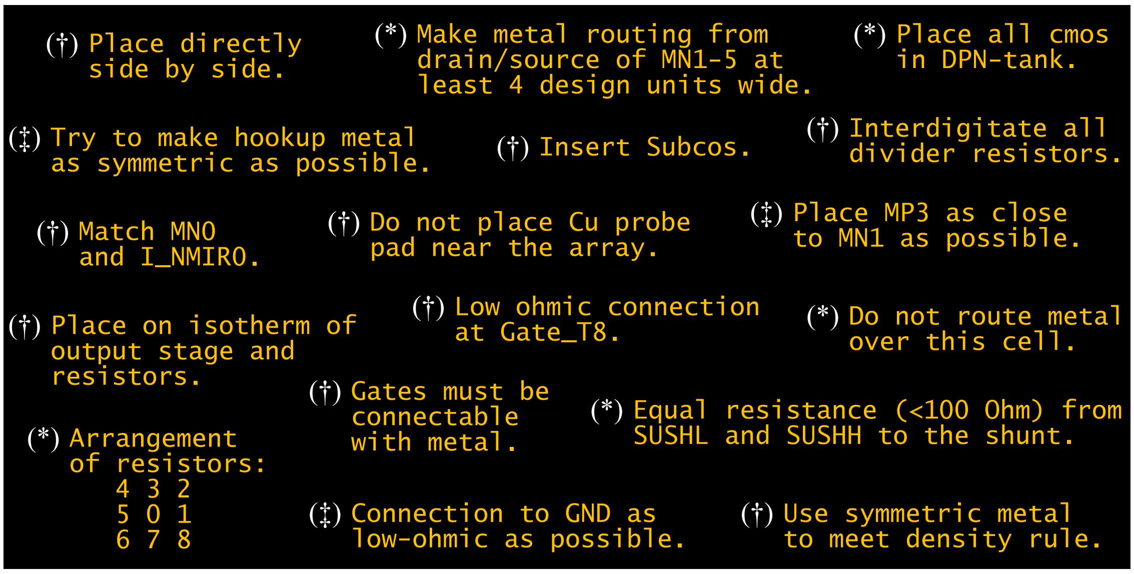
Enabled	true
Side	bottom
PreserveOr...	false
pitchRefere...	center
pitch	(not set)
use	boundary
layerName	OVERLAP

Axis	bi-directional (default)
Enabled	true
Min	(not set)
Max	(not set)
reference	edge
use	boundary
layerName	OVERLAP

Enabled	true
Message	Keep away from power stage.

Figure 2.5: Examples of largely formalized constraints, as entered into a constraint management system.

to deduce the respective implications. Finally, constraints marked with a double dagger (‡) describe – strictly speaking– optimization goals, and are unspecific in the sense that their respective importance in relation to the other design objectives is not given (and is thus also subject to the interpretation of the layout expert).


(†) Place directly side by side.

(*) Make metal routing from drain/source of MN1-5 at least 4 design units wide.

(*) Place all cmos in DPN-tank.

(‡) Try to make hookup metal as symmetric as possible.

(†) Insert Subcos.

(†) Interdigitate all divider resistors.

(†) Match MN0 and I_NMIR0.

(†) Do not place Cu probe pad near the array.

(‡) Place MP3 as close to MN1 as possible.

(†) Place on isotherm of output stage and resistors.

(†) Low ohmic connection at Gate_T8.

(*) Do not route metal over this cell.

(†) Gates must be connectable with metal.

(*) Equal resistance (<100 Ohm) from SUSHL and SUSHH to the shunt.

(*) Arrangement of resistors:
```
4 3 2
5 0 1
6 7 8
```

(‡) Connection to GND as low-ohmic as possible.

(†) Use symmetric metal to meet density rule.

Figure 2.6: Examples of largely nonformalized constraints, stored as prosaic labels in circuit schematics.

Looking at formalized and nonformalized constraint representations with regard to the respective existence or absence of sufficient formal descriptiveness and semantical concreteness, one can basically discern two forms of constraint consideration: an *explicit* consideration of constraints and an *implicit* consideration of constraints. Formalized constraints can be considered explicitly, whereas nonformalized constraints can only be considered implicitly.

Since both kinds of constraint *representation* can be encountered in today's flows of manual layout design, both forms of constraint *consideration* can be found. In general, it can be said that high-level design requirements usually must be specifically enunciated in a formalized way to be explicitly taken into account by the layout designer, while low-level layout aspects are often implicitly taken care of by a seasoned layout expert without the need to formalize them. Quoting [25], analog designs are "exceptionally rich in critical *inexplicit* constraints".

Now, the distinction between explicit and implicit constraint consideration can also be examined in view of design automation, discerning the two prevalent automation strategies already mentioned before: *optimization algorithms* and *procedural generators*. The background for this distinction are the ruminations in Section 3.1, which point out that optimization algorithms can consider constraints only explicitly (given that they are provided in a formalized fashion), while procedural generators can consider constraints only implicitly (without the need to formalize them), and that no existing approach is truly able to support both.

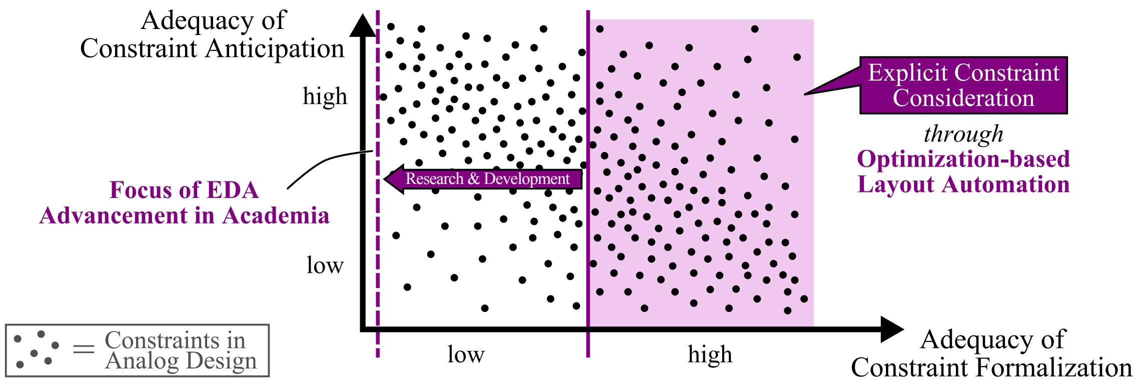
Recalling the fact that both formalized and nonformalized constraints are encountered in practice, the following conclusion must be drawn: as long as it is not possible to make *all* constraints accessible to either an entirely explicit or an entirely implicit constraint consideration, none of the two automation strategies alone is able to tackle the problem of analog layout design in its entirety. This in turn substantiates the neccessity for a new layout automation methodology, as will be discussed in Section 2.5.

2.5 Necessity for a New Layout Automation Methodology

With respect to the consideration of constraints, there are basically three approaches imaginable to address the problem of analog layout automation: formalize all constraints and consider them explicitly with an optimization-based approach (see Section 2.5.1), elide the formalization of constraints and consider them all implicitly with a generator-based approach (see Section 2.5.2), or provide a new methodology that is able to consider constraints both explicitly and implicitly (see Section 2.5.3).

2.5.1 Optimization-based Layout Automation with Explicit Constraint Consideration

Trying to consider all design constraints in a purely explicit way using optimization algorithms is the major focus of EDA advancement in academia. This is visualized in Figure 2.7, where constraints of analog IC design have been figuratively illustrated as dots that are distributed across a two-dimensional plane, depending on how adequately the constraints can be formalized and on how adequately they can be anticipated in advance.



Adequacy of Constraint Anticipation
high
low
Focus of EDA Advancement in Academia
Research & Development
Explicit Constraint Consideration
through
Optimization-based Layout Automation
= Constraints in Analog Design
low
high
Adequacy of Constraint Formalization

Figure 2.7: Focus on optimization-based layout automation with explicit constraint consideration.

As indicated in the image, optimization algorithms are quite appropriate for targeting constraints whose adequacy of formalization is high. For instance, a well-formalizable constraint is the so-called Fixed Outline constraint (covered in Section 3.1.1.2) which demands that all components must fit inside a given layout contour. This constraint represents an unambiguous requirement which exactly formulates a practical design restriction one-to-one as a concise and verifiable geometrical condition.

However, a *purely* explicit constraint consideration requires that *all* constraints are explicitly expressed in a formal, comprehensive, unambiguous and consistent representation that can be processed by the algorithm. While this is manageable in the digital domain, where most constraints can be described quite concisely (such as a maximum signal delay, for example), the situation is different in the analog domain, since it is tremendously intricate and simply not possible nowadays to *efficiently* and *sufficiently*[7]

[7]The two terms signify an economic and a technical implication, which will also be reflected in Chapter 4.

describe the full diversity, various impacts and correlated dependencies of all functional design restrictions and design objectives in a formal fashion due to their More than Moore character. This difficulty becomes particularly apparent when many tightly-linked, contrary, low-level layout requirements need to be satisfied concurrently. Simply put, such constraints are extremely hard to formalize. A good example is given by *matching* on device level, which

- involves several different constraints such as compactness, orientation, interdigitation, and common centroid (see Section 2.1.2),
- must still allow for certain degrees of freedom (e.g., a single-row or a dual-row arrangement as shown in Figure 3.5 (a) and (b) respectively, on page 36),
- has to bring in line both the placement and the routing although the respective necessities may oppose each other (see Figure 3.12 on page 44 for an example).

The gap between the transistor rows' formal alignment edges in Figure 3.5 (b) indicates that translating such design requirements into formalized constraint representations often fails to cover the needs of the real world, while examples (b) and (c) in Figure 3.12 illustrate how even state-of-the-art automation approaches can be seen to be incapable of performing placement and routing in harmony. Section 3.1.1 will discuss the inherent difficulties in greater detail, but for the argumentation in this chapter it suffices to realize that these difficulties are undeniably confirmed by reality, where optimization-based tools have not truly found their way into industrial design environments so far. Thus, from the present point of view, one has to acknowledge that it is not possible to formalize all relevant design constraints in a practicable and profitable way that facilitates a purely explicit constraint consideration.[8]

In spite –or because– of these issues, analog EDA research continues to work on a vast variety of topics in the context of optimization algorithms, in particular dealing with the formalization and consideration of constraints. But whether optimization algorithms and a purely explicit consideration of constraints will someday be sophisticated enough to prevail beyond academic exercise and succeed in analog layout automation as it is the case in the digital domain, remains open to debate. As [26] comes to the point, constraint information is seldomly written down – and when it is, "its form differs from team to team, product to product, and company to company".

Even if all constraints are expressed in a formalized way, the layout solution that an optimization algorithm might find can at most be as good as the mathematical model by which the algorithm works. Furthermore, the use of inadequate heuristics may prevent the algorithm from finding the (theoretically) optimal solution, while –on the other hand– an omission of heuristics makes it less likely that the algorithm finds a feasible solution within practical runtime limits. Hence, the application of optimization-based approaches to More than Moore problems is hindered by the reciprocity between the accurateness of the modeling and the nearness of the model's optimum to the solutions that can be expected to be found by the algorithm.

2.5.2 Generator-based Layout Automation with Implicit Constraint Consideration

Although EDA research is strongly focused on optimization algorithms, comparably simple procedural generators remain the most frequented pieces of automation in practice. Despite their petty abilities to create layouts for –basically– primitive devices, they are practically indispensable for a human designer's day-to-day layout work. While academia has mostly overlooked them for the reason of being rather trivial from a scientific perspective, industrial design teams can be observed to pursue an advancement of simple device generators towards more powerful hierarchical generators which are able to create entire layouts for small circuits consisting of multiple devices and their electrical interconnections.

As will be described in Section 3.1.2, the output of a procedural generator is denoted as a layout *result* in this thesis, while –in contrast to an optimization algorithm– the cognitive layout *solution* is not found by the automatism but is actually preconceived by the human expert who implements the generator. On the one hand, this is what gives procedural generators the advantageous ability to consider

[8]Again, there is an economic and a technical implication here because *practicable* should be understood as "technically feasible", while *profitable* should be understood as "economically viable".

design constraints implicitly, but the immanent downside is the effort required to develop the generator and the difficulty to think of all relevant design requirements in advance for incorporating them into the implementation of the generator.

As indicated in Figure 2.8, procedural generators are particularly fruitful for addressing constraints which can be easily anticipated. This makes procedural generators a perfect choice for automating *simple modules*: the primary demand for such modules is to achieve a good device matching, and these matching requirements are usually crystal-clear and can therefore be handled by the generators without the need for further *constraining*. Regarding other constraints, which are rather encountered when proceeding towards higher-level modules, the adequacy of anticipation is much lower. Again, an example is the Fixed Outline constraint since it is not feasible to anticipate all potential outlines in advance, nor is it expedient to preconceive layout solutions for all these outlines. Despite the technical possibility to do this, the effort would be unreasonably high because such constraints are so hard to anticipate.

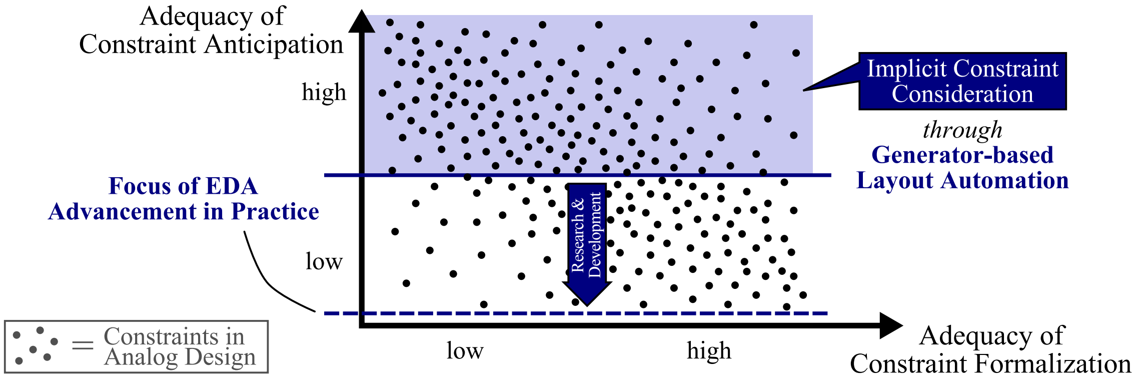

Adequacy of Constraint Anticipation
high
low
Focus of EDA Advancement in Practice
Research & Development
Implicit Constraint Consideration
through
Generator-based Layout Automation
low
high
Adequacy of Constraint Formalization
= Constraints in Analog Design

Figure 2.8: Focus on generator-based layout automation with implicit constraint consideration.

In summary, it is not possible to anticipate all relevant design constraints in a practicable and profitable way that facilitates a purely implicit constraint consideration.[8] Due to this observation, the use of procedural generators in the industry is not showing to be lucrative enough above the level of basic circuit modules such as Current Mirrors and Differential Pairs. Mature generator development tools (as will be presented in Section 3.1.2.4) can help reduce the implementation effort to a certain degree, but a more profound conceptual breakthrough which allows to meet the economic and technical implementation demands of higher-level modules has yet to be delivered. Until then, the development of procedural generators keeps grappling with a tradeoff between the generators' implementation effort and their potential benefit for design productivity.

2.5.3 New Methodology with Explicit and Implicit Constraint Consideration

The figures above visualize an important argument: most design constraints *either* have a high adequacy of anticipation *or* a high adequacy of formalization. This can for example be seen in the already mentioned matching requirements and the Fixed Outline constraint respectively (as discussed in Section 2.5.1 and Section 2.5.2). A couple of –comparably few– constraints can be *both* adequately anticipated *and* formalized. An example would be the restriction to prevent routing wires from running above transistor channels. This demand is commonly adhered to by procedural generators implicitly, although an optimization algorithm would just as well be able to take this constraint into consideration when explicitly being told to do so. At the other end of the spectrum, there may always be a minority of constraints that can *neither* be adequately anticipated *nor* formalized. Presumably, such exotics will never be completely addressable via design automation but always rely on human expertise.

For the remaining "ordinary" constraints, the preceding contemplations advocate a novel automation strategy that combines the assets of optimization algorithms and procedural generators to support an explicit as well as an implicit consideration of constraints. Such an automation strategy is meant to facilitate a truly balanced consideration of both formalized and nonformalized representations of expert knowledge, since these are equally indispensable to cope with the More than Moore design complexity in the analog domain. This conclusion phrases the focus of this work and marks the technical aim of this thesis. As shown in Figure 2.9, a suchlike methodology has significantly more potential for analog layout automation than optimization-based or generator-based approaches alone.

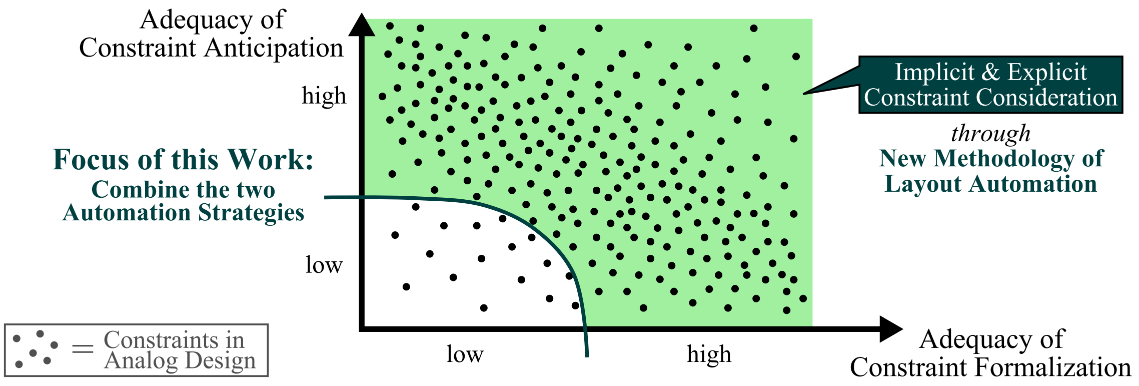

Adequacy of Constraint Anticipation
high
low
Focus of this Work:
Combine the two Automation Strategies
Implicit & Explicit Constraint Consideration
through
New Methodology of Layout Automation
low
high
Adequacy of Constraint Formalization
= Constraints in Analog Design

Figure 2.9: Novel automation strategy with explicit and implicit constraint consideration.

But how can these two automation strategies be married? As Figure 2.7 and Figure 2.8 hint at, procedural generators and optimization algorithms are oriented in such different ways that they are not compatible with each other off the shelf. When combined, the strengths of the one automation strategy redound to a disadvantage for the other. This crux shall be confirmed by two thought experiments based on two obvious ideas:

- The first idea is to include procedural powers in an optimization algorithm by applying that algorithm not to a set of primitive devices (as is typically done) but to a set of generator-based modules. Thereby, many placement and routing tasks on device level can be performed by the modules instead of the algorithm. Furthermore, the modules may provide the valuable variability that would otherwise be lost due to inadequately strict, explicit constraints on the devices. However, this variability is detrimental for the algorithm because the supported degrees of freedom blow up the solution space that is to be searched through. Metaphorically speaking, there are more fish to catch but the algorithm has to fish in a much larger amount of muddy water.
- The second idea is to inject algorithmic capabilities into a procedural generator by facilitating an exertion of influence on the generator's contents through some form of intervention (e.g., forcing some of the subdevices inside the generator to be placed at a certain side). If suchlike abnormalities do not have to be preconceived by the generator developer, this would reduce the implementation effort, but at the same time it would be fateful for the generator because of its inability to handle such an unforeseen –i.e., not anticipated– situation (in this example: to automatically adjust the positions of the other subdevices). Similar to the body of a transplant recipient, the generator can be thought of as a closed organism rejecting a donor organ.

From these considerations, it becomes clear that a combined algorithmic-procedural approach needs to engage the two automation strategies in a harmonic way such that decisions of the one strategy are not fatal for the other. However, this does not mean to separate the two strategies up to isolation nor to diminish the authority of one strategy near to insignificance: in either case, the two counterparts would not really profit from each other. Instead, a truly hybrid methodology is supposed to join the two automation strategies in a balanced relationship where each one is able to *react* to the other's doings.

As will be detailed in Chapter 3, no such methodology has been developed so far, which adds authority to the question in how far this endeavor is realizable – if at all. The inherent difficulty (and likewise, the big opportunity) will be nailed down in Section 3.2.3: providing a methodology that facilitates an explicit and implicit consideration of constraints means to combine two fundamentally different *automation paradigms*.

Chapter 3

State of the Art - Two Views on Existing Approaches

The most insidious world-views are held by people who have never viewed the world.
Alexander von Humboldt (German explorer)

This chapter is dedicated to existing EDA approaches for automatic layout creation, discerning between *optimization algorithms* and *procedural generators*. Section 3.1 first takes a concrete look on existing approaches to pinpoint the shortcomings of these two automation strategies before Section 3.2 examines that state of the art from a more abstract point of view. In continuation of the argumentation in Chapter 2, the subsequent discussion concludes that facilitating a combined explicit *and* implicit consideration of constraints requires the coalescence of two fundamentally different automation paradigms here denoted as *top-down automation* and *bottom-up automation*. This comprehension delineates the scientific challenge of the work presented in this thesis.

3.1 A Concrete View on Existing Approaches: Optimization Algorithms and Procedural Generators

While EDA research has put forth a rich variety of analog layout automation approaches over the past decades, these can be divided into two fundamental categories already mentioned in Chapter 2: *optimization algorithms* and *procedural generators*. The following Section 3.1.1 and Section 3.1.2 will describe these two automation strategies in greater detail, elaborating on their strengths and weaknesses. Then, although a couple of further and seemingly hybrid approaches have also been developed, Section 3.1.3 points out that no existing approach facilitates a sincere combination of both automation strategies. Finally, Section 3.1.4 draws a couple of conclusions for the new approach taken by this thesis.

3.1.1 Optimization Algorithms

With reference to [27], the development of optimization algorithms in EDA largely concentrates on a canonical form depicted in the center of Figure 3.1, where a single candidate layout is repeatedly optimized in a loop of solution space *exploration* and candidate *evaluation*. An exploration engine navigates through the solution space to refine the candidate layout. If the algorithm is *constructive*, the refinement is achieved by successively completing the candidate layout, whereas in the case of an *iterative* algorithm, the candidate layout is repeatedly modified. An evaluation engine validates if the new candidate layout satisfies all design restrictions, and rates its quality according to a formal metric (e.g., a cost function that is to be minimized). Depending on the rating, the layout is accepted or rejected and the cycle of optimization continues until the final *solution* to the given layout design problem is obtained with respect to a certain stop criterion.

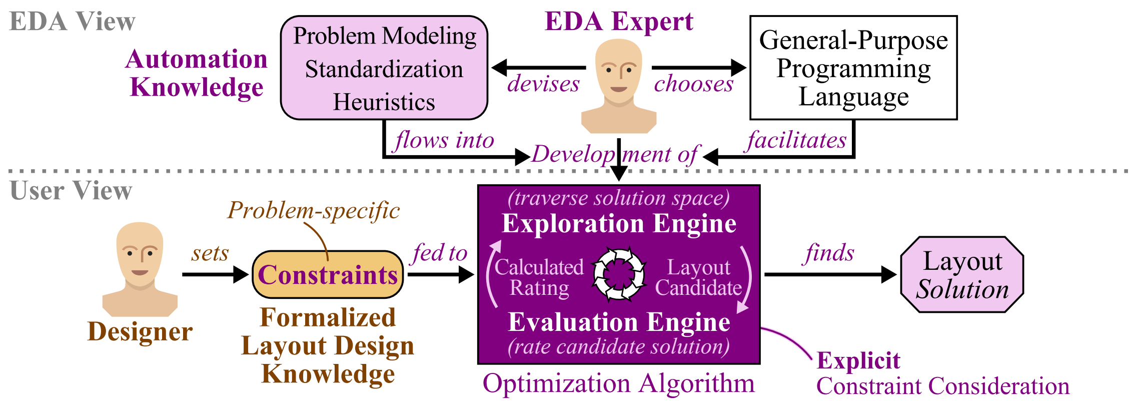


Figure 3.1: Working principle of an optimization algorithm (adapted from [27]).

The EDA View in Figure 3.1 (which has already been referred to in Section 2.3.1) as well as the constraint-related entries in the User View will now be further discussed in Section 3.1.1.1.

3.1.1.1 Constraint Handling

Thanks to their abstract nature, a particular strength of optimization-based layout algorithms is that they can be versatilely applied to various kinds of circuits. An affiliated asset is their ability to take into account formal expressions of high-level design constraints which easily elude the scope of human attention in manual design. But as already mentioned in Section 2.3.1, an optimization algorithm –apart from applying heuristics and design-methodical standardization– works by translating the optimization problem into a mathematical model, thereby specifically optimizing *just* the modeled aspects [28].

This means, that a design constraint is not taken into consideration unless the algorithm is explicitly told to do so. In other words, the expert knowledge that goes into the development of an optimization algorithm –usually implemented with a *general-purpose programming language*– (see EDA View in Figure 3.1) can be considered as automation knowledge, but it does not involve any palpable layout design knowledge. Instead, *all* relevant layout design knowledge for a specific design problem must be completely formalized as explicit constraints that can be handled by the algorithm (see User View in Figure 3.1). This imperative represents a tremendous challenge because it involves several requisites. As illustrated in Figure 3.2, for every single constraint these requisites are:

(1) *Semantics*:
The user must think of how the constraint can be precisely articulated in formal terms.
(2) *Syntax*:
The constraint must be formally describable, which requires a dedicated “language”.
(3) *Formulation*:
The user must formally describe the constraint, which can take significant effort.
(4) *Consistency*:
The constraint must not conflict with other constraints (although optimization goals are often mutually contradictory).
(5) *Processibility*:
The algorithm must be able to take that type of constraint into consideration.
(6) *Computation*:
Regarding the problem at hand, the algorithm must find a solution that satisfies the constraint.

With regard to the More than Moore complexity of analog IC design, the difficulty to address each and every constraint in this manner restrains an incorporation of human expertise into the automation. The initial requisite alone (i.e., getting the detailed implications of a constraint semantically right in the first place) represents a major obstacle for the employment of optimization algorithms in practice. These semantics raise the question, what types of constraints can be addressed at all (Section 3.1.1.2).

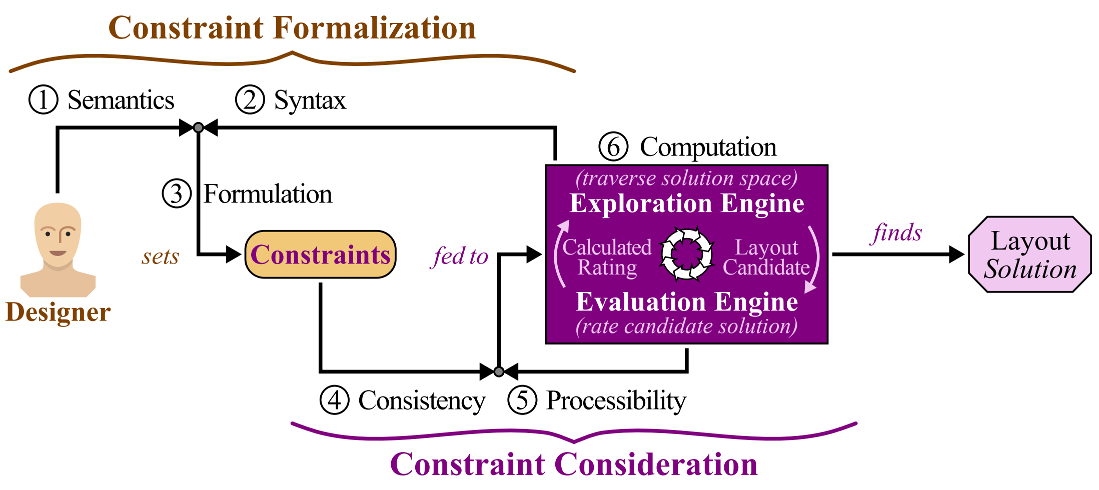


Figure 3.2: The requisites of formalizing and explicitly considering a design constraint.

3.1.1.2 Types of Constraints

The different types of constraints that have been algorithmically addressed so far in literature, can be divided into placement constraints and routing constraints. In general, placement constraints may also play a role in floorplanning, but many of these constraints –especially those that are meant to ensure a good matching– are of rather little interest for floorplanning compared to device placement. Frequently encountered placement constraints, as applied to a set of layout components, are illustrated in Figure 3.3 and can be briefly described as follows:

Alignment:
A specific edge of each component lies on the same horizontal or vertical axis.

Abutment:
The components are aligned in one direction and placed adjacent to each other in the orthogonal direction.

Boundary:
Each component is placed along the left, right, top, or bottom border of the layout.

Clustering:
The components are placed adjacent to a given target component so they form a cluster.

Common Centroid:
Subgroups of components are placed such that their geometrical centers fall onto the same location.

Distance (Proximity/Separation):
The distance of two components is greater than a specified minimum and/or less than a specified maximum.

Equal Parameters:
All components to be matched are identical, i.e., each one has assumed the same layout variant.

Fixed Outline:
Every component is placed within a specified rectilinear or rectangular layout contour.

Orientation:
The components are rotated or flipped such that all of them have equal or mirror-inverted orientation.

Preplacement:
Components are placed in a predetermined fixed location to which they must adhere.

Symmetry:
Identical components are placed in mirror-inverted orientation about a horizontal or vertical axis.

Symmetry Island:
The components are placed in symmetry and each of them abuts at least one of the other components.

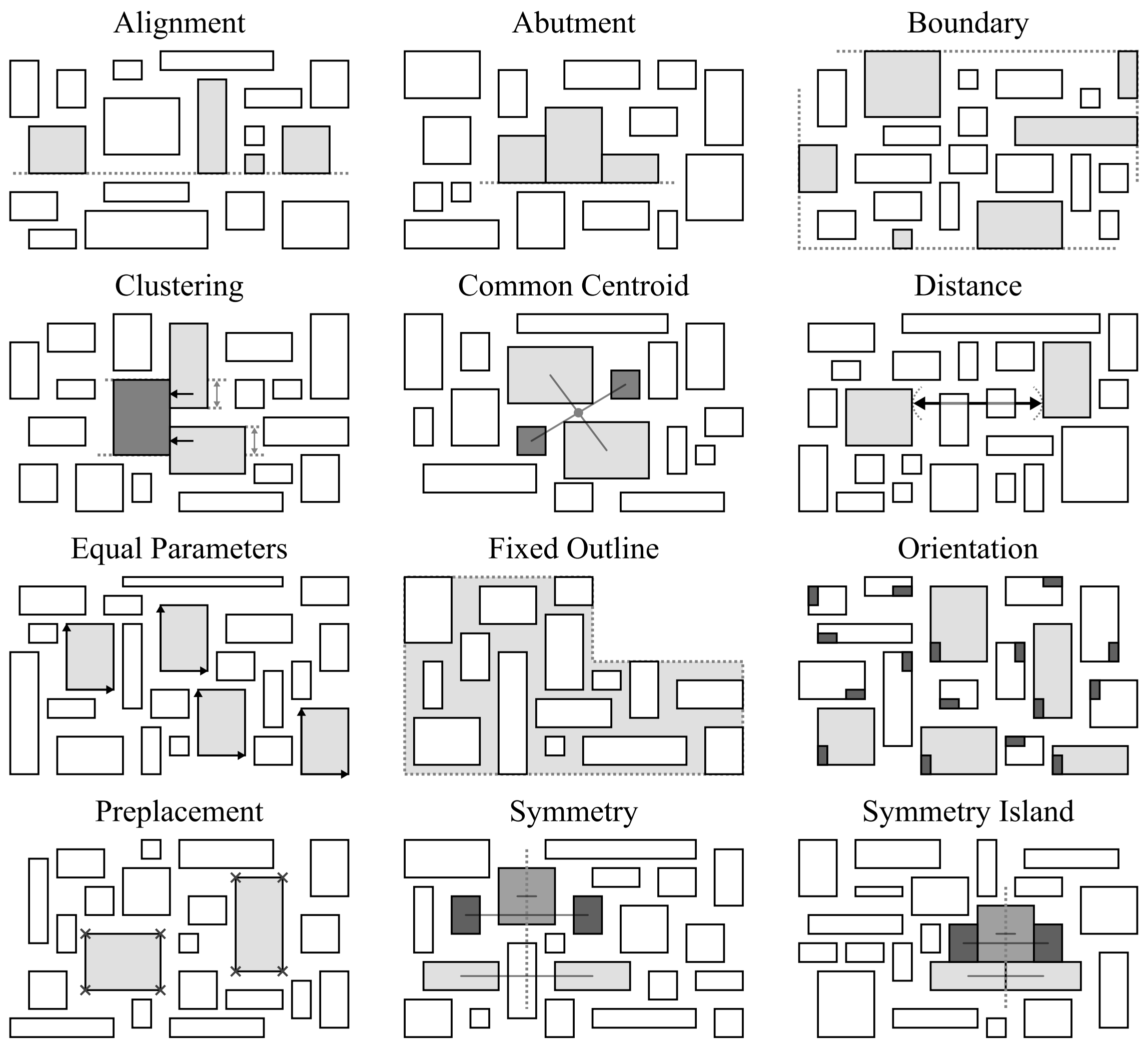


Figure 3.3: Illustration of common placement constraints in optimization-based layout automation.

One point of critique that can be brought up here is how cumbersome certain constraints are to express an actually plain desire. As an example, if a group of devices need to be arranged in a two-dimensional matrix-like fashion (e.g., a resistor array), the only serviceable type of constraint to formally express this request is the Alignment constraint. However, specifying one single Alignment constraint does not suffice: for an n by m array of devices with n columns and m rows, $n + m$ Alignment constraints need to be specified. And although the effort to do so already is unnecessarily high, it does not yet state any demands about the horizontal pitch Δx between the array columns nor about the vertical pitch Δy between the array rows. This would require another $n - 1$ Distance constraints to set the desired distance of the array columns, plus $m - 1$ Distance constraints for the array rows.

As an example, Figure 3.4 shows an 8 by 6 resistor array which requires eight constraints for the horizontal alignments and six constraints for the vertical alignments. With 7 plus 5 additional constraints for the distances, there is a total of 26 constraints to be formulated. But then, even if that amount of effort is spent, these constraints do not yet express how the devices should be interdigitated across that array (which ironically is often the basic reason for such an array arrangement). There is another issue, because on every device at least two constraints are imposed (horizontal Alignment and vertical Alignment, possibly plus Distance constraints). This means that the algorithm must first of all be able to consider more than one such constraint per device. Much more convenient than such a constraint-based approach would be to employ an automatism specifically dedicated to the creation of device arrays and requiring only three parameters: the interdigitation pattern, the horizontal pitch, and the vertical pitch.

Unfortunately, one cannot stint on the constraining effort since *all* constraints need to be formally expressed: it already occurs with *one* missing constraint that the algorithm is not bound to produce a feasi-

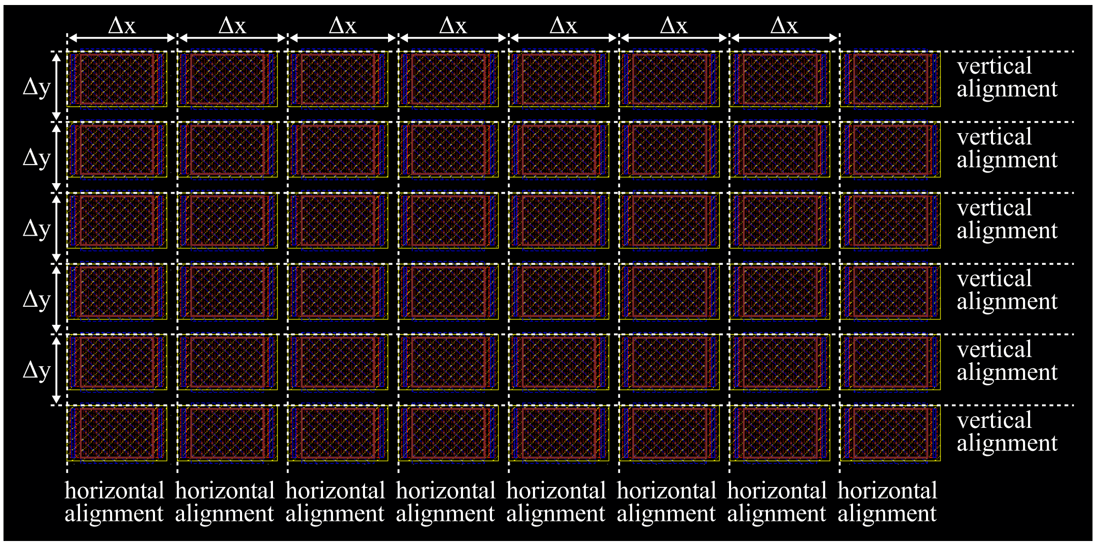


Figure 3.4: Resistor array requiring an awkwardly large number of placement constraints.

ble solution even if all other constraints are satisfied. But there is more to that issue because –on the other hand– the constraints can also be too strict in two aspects. One aspect is denoted as *overconstraining*, i.e., a situation where the articulated constraints cannot be satisfied because they formally contradict each other. Discovering such conflicts in advance requires an inference system that is able to resolve all articulated constraints via a logical calculus (also see Section 3.1.1.4). However, the dependencies can be so convoluted that an a priori constraint resolution is hardly possible.

The other aspect is that many formal constraint representations are too static to express geometric variability. For example, an Alignment constraint could be used to enforce a Current Mirror layout where all transistors are assembled side by side in one single row, as in the example of Figure 3.5 (a). However, the transistors can also be folded and spread over two rows. Although such an arrangement even improves the matching, it is not admitted by the Alignment constraint because the alignment edge of the transistors in the first row is not the same as the alignment edge of the transistors in the second row (b). Since this kind of variability often is a vital requisite to be capable of satisfying all constraints across an entire chip design, its absence from formal constraint representations is a big problem for optimization-based layout automation.

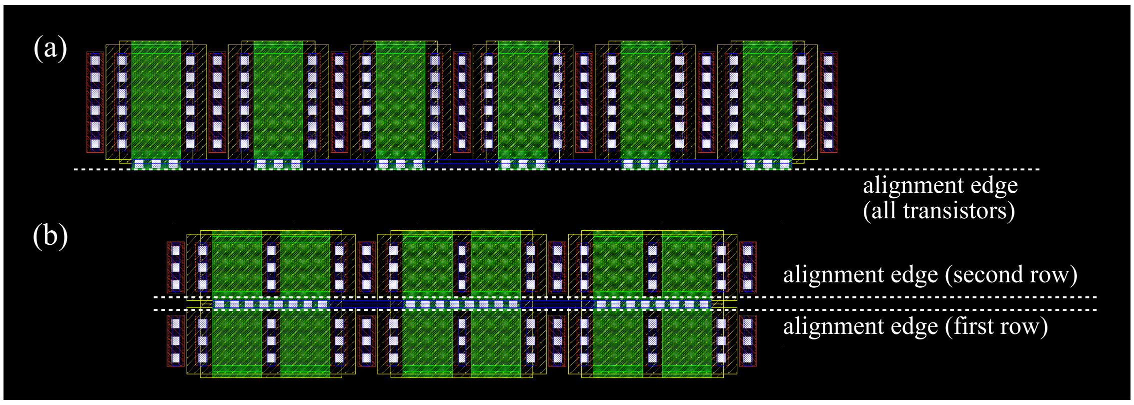

(a)
alignment edge (all transistors)
(b)
alignment edge (second row)
alignment edge (first row)

Figure 3.5: Transistor arrangement with (a) satisfied and (b) violated Alignment constraint.

Routing constraints (see Figure 3.6), including those already mentioned in Section 2.1.4 as well as further ones addressed in literature, are:

Blockage:
Prohibition of wires above certain areas such as the channel of a transistor.

Layer Limitation:
Limited set of metal layers available for the segments of a wire.

Wire Width:
Variable widths of a wire's segments, depending on the respective current load.

Sensitivity:
Decoupling of sensitive nets from noisy nets via wire separation or wire shielding.

Double-Cut Vias:
The vias of a wire contain two clearance holes (at least).

Length Matching:
The segments of two wires add up to the same total wirelength respectively.

Common Centroid:
Nets are routed symmetrically with respect to a common center point.

Symmetry:
Nets are routed symmetrically with respect to a common axis.

Topological Matching:
The segments of two wires match in number, length, layer, and bends.

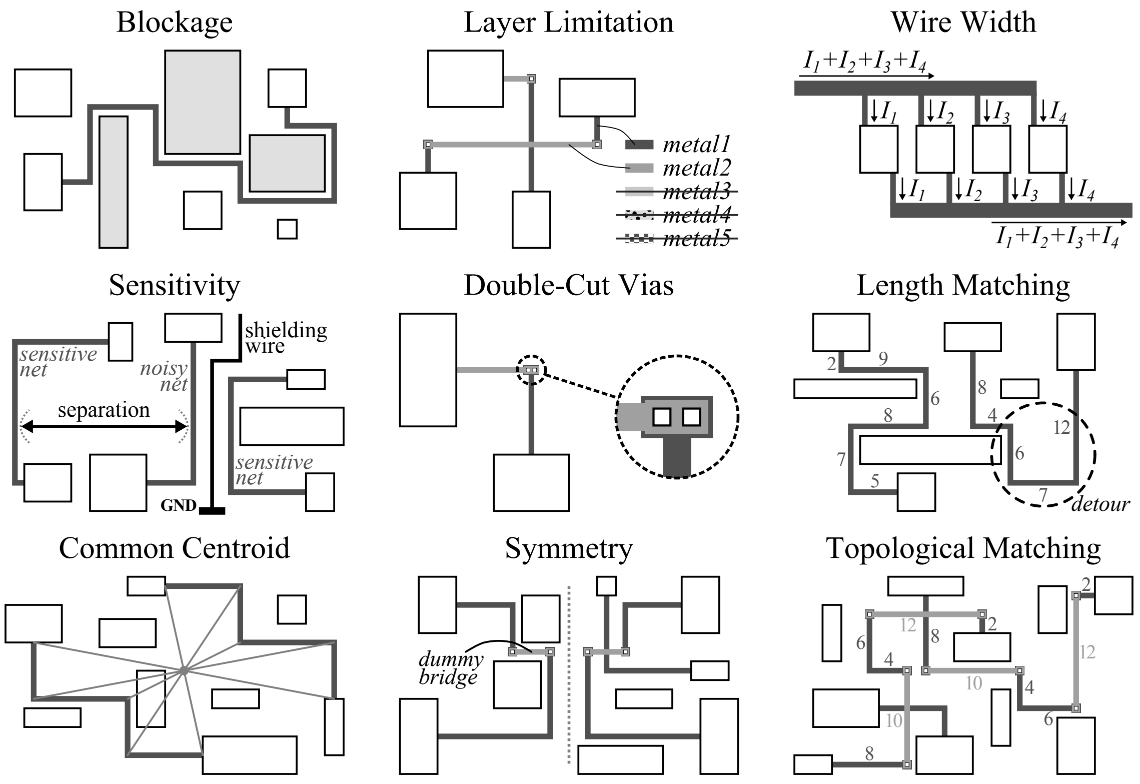


Figure 3.6: Illustration of common routing constraints in optimization-based layout automation.

Routing constraints are good to exemplify one major nuisance: in individual cases, a violation of an otherwise obligatory constraint may be necessary, but even if that violation is electrically uncritical it is rejected as a formal dissatisfaction of the constraint. For example, the Layer Limitation constraint can be used to enforce a device-level routing which employs only metal1 and metal2. For some nets these layers might not suffice, but unfortunately the constraint does not permit the usage of a metal3 wire here even if it would be unproblematic to have a few of such metal3 wires in the device-level routing. This formal obstinacy can prevent the routing algorithm from finding a solution at all. Theoretically, it might be possible to allow for the specification of certain exceptions. However, this would not only require further constraining effort but also insight into (or feedback from) the algorithm to see where such exceptions could be helpful.[1]

[1]Then again, if the algorithm is not deterministic, these exceptions would become futile when re-running the algorithm.

Another facet of that problem is that even the slightest deviation from a constraint causes formal dissatisfaction. If, for instance, there is a marginal difference in the length of two wires (e.g., because the routing grid causes potential wire detours to cover an odd number of grid units in distance where an even number would have been necessary instead), then the Length Matching constraint is formally violated (unless a certain tolerance has been specified, which would again involve more constraining effort in turn). Figure 3.7 shows an example with a wire whose length of 7 is to be matched by another wire. With a direct, axis-oriented connection –as seen in image (a)– that other wire has a length of 4 (the wire's so-called *Manhattan distance*). By adding a detour as in (b), the wire's length can be increased to 6 (which is still not enough) while the next possible detour overexpands the wire to a length of 8 (c). Obtaining the desired wire length 7 necessitates to put some wire nodes off the routing grid (d). In practice, where the grid size is typically much smaller than the length of a routing wire, length mismatches as in (b) and (c) are electrically negligible but would still represent a formal constraint violation.

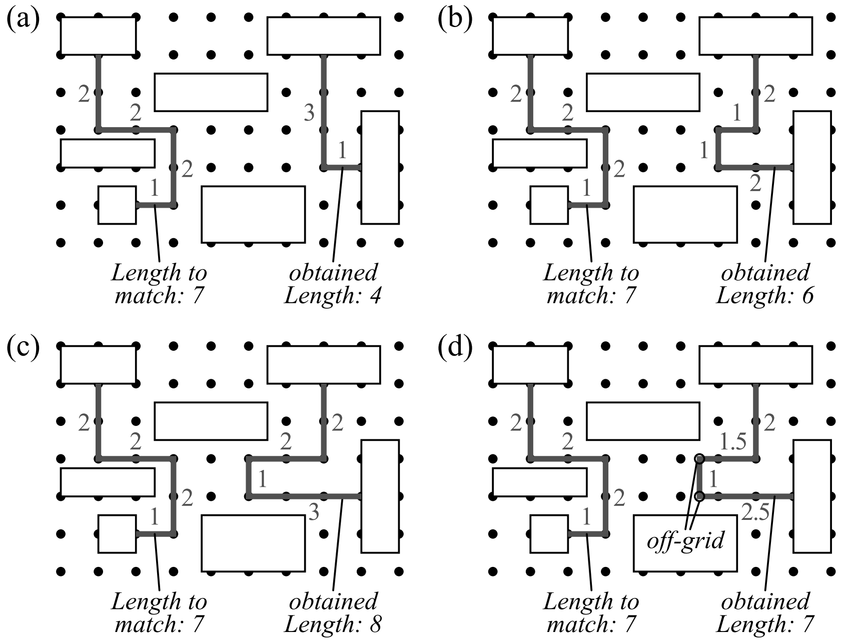


Figure 3.7: Inability of satisfying a Length Matching constraint and staying on the routing grid.

The illustrated problem points to a further aspect: a constraint can be entirely contradictory to a design-methodical restriction. To give an example, assume that a transistor is to be used as a diode, which is done by connecting the transistor's gate with its drain. Unfortunately, there is a failure mechanism known as plasma-induced gate oxide damage (where a gate dielectric can break down by collecting electrical charges due to reactive-ion etching during manufacturing). One common measure to prevent this so-called *antenna effect* is to connect the transistor gate with a diffusion using only metal1. However, if there is a design-methodical restriction claiming horizontal wires to be created on metal1 and vertical wires to be created on metal2 (or vice versa), then a metal1-only connection is not possible because the connection includes a horizontal and a vertical wire segment. The first restriction case is exemplified in Figure 3.8 (a) while the second case is illustrated in image (b): in both cases, a Layer Limitation constraint demanding a metal1-only connection would not be satisfied. Such a connection, as shown in (c), involves a violation of the design-methodical restriction.

Apart from the described troubles with the formalized types of placement and routing constraints, another question is inhowfar existing approaches are able to handle these constraints in the first place. This question is subject to the following Section 3.1.1.3.

3.1.1.3 Comparison of Existing Approaches

Due to the complexity of the layout design problem, most optimization-based approaches concentrate on just one of the different design tasks. Thus, only few approaches deal with the challenge of performing

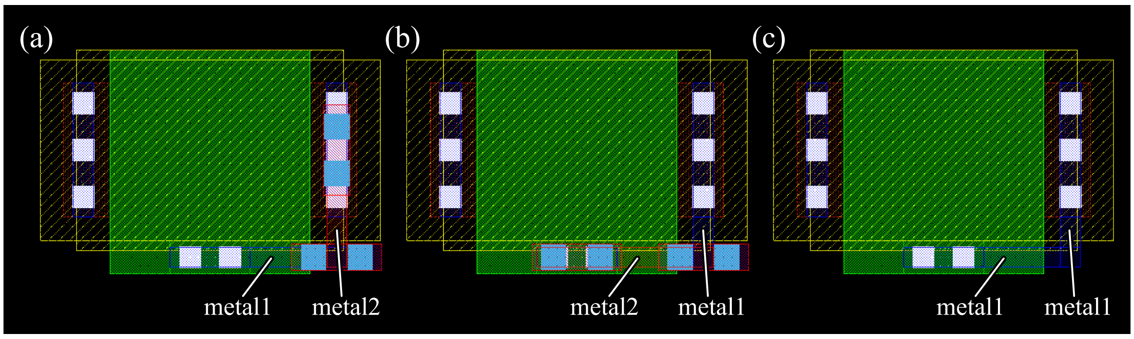


Figure 3.8: Violating either a Layer Limitation constraint (a)(b) or the routing layer preference (c).

placement and routing simultaneously, although these two tasks are heavily correlated. For every task from (1) floorplanning, (2) placement, and (3) routing to the (4) combined placement and routing, a representative selection of existing approaches is provided in each of the subsequent paragraphs. This is done with a particular focus on the types of constraints that can be handled by the different approaches.

(1) Floorplanning

While the early work of [29] (one of the first general floorplan automatisms from 1983) did not take wirelength into account, most contemporary floorplanning approaches are able to pursue both main objectives of the floorplanning task, i.e., area minimization and wirelength minimization. The primary design restriction in floorplanning is the layout contour into which the sought floorplan has to fit; but as Table 3.1 shows, some approaches do not pay respect to this Fixed Outline constraint. If the floorplanning targets only a certain part of the chip, it can even be the case that the outline is not necessarily rectangular but arbitrarily rectilinear (e.g., L-shaped or T-shaped). However, none of the existing floorplanning approaches supports such nonrectangular outlines, as is reflected by the table.

Table 3.1: Comparison of selected floorplanning approaches for analog layout design.

Reference	Year of Public.	Considered Fix Outline	Supported Variability	Floorplan Structure	Deterministic	Solution Approach
[30]	1989	rectangular	var. width	only slicing	yes	standard cell approach
[31]	1991	none	discrete	only slicing	yes	fix slicing tree traversal
[32]	1993	not reported	none	nonslicing	yes	local transformations
[33]	2003	rectangular	full	nonslicing	no	Simulated Annealing
[34]	2004	rectangular	free shapes	nonslicing	no	modified Min-Cut
[35]	2004	rectangular	nondiscrete	nonslicing	no	Simulated Annealing
[36]	2006	rectangular	full	nonslicing	no	Simulated Annealing
[37]	2006	rectangular	none	trapez. rows	partly	named Traffic
[38]	2007	none	none	nonslicing	no	evolution. computation
[39]	2011	none	none	nonslicing	no	Simulated Annealing
[40]	2014	rectangular	none	not reported	no	swarm algorithm
[41]	2014	not reported	none reported	only slicing	no	Simulated Annealing
[42]	2015	rectangular	full	only slicing	yes	analytical approach

One decisive aspect that distinguishes floorplanning from placement is that the aspect ratios of the floorplan blocks may have full variability with dimensions in a continuous range (whereas the devices in placement usually have just discrete variability, i.e., a finite set of different layout variants that they can assume). Nevertheless, only few approaches support this kind of full variability, as can again be seen in Table 3.1. Apart from this deficiency, a further shortcoming is that some approaches are limited to so-called *slicing floorplan*[2] structures. This design-methodical restriction reduces the solution space in favor of the floorplanning algorithm but is usually insufficient for practical analog designs because these may in general ask for a –computationally more challenging– *nonslicing floorplan*.

Last but not least, the majority of floorplanning approaches are not deterministic, as Table 3.1 underlines. Unfortunately, nondeterministic behavior is often itemized as one major reason for the low acceptance of optimization-based layout automation approaches in practice. In particular, it represents a serious hurdle for the constraining process: since it is seldomly possible to get all constraints adequately formalized straight away, the constraining typically consists of repeatedly re-running the algorithm and gradually specifying further constraints depending on the algorithm's solution. However, if the algorithm behaves in a nondeterministic way, such a stepwise constraining approach becomes highly unintuitive, unsystematic, and uncomfortable.

Still, many floorplanning approaches approximate the optimal solution via the stochastic *Simulated Annealing* algorithm [44], a metaheuristic technique that mimics the slow annealing of solids in metallurgy. Simulated Annealing is an example of a "hill-climbing" algorithm which can abandon local optima in favor of better optima by temporarily accepting worse solutions. An attribute of Simulated Annealing is that the probability of accepting a worse solution slowly decreases with runtime, analogous to the decline of an atom's thermodynamic energy in the annealing material. Despite the reservations against nondeterministic algorithms, Simulated Annealing remains the most popular optimization engine in published works, not only for the floorplanning task but even more in placement.

(2) Placement

With classic approaches such as *Force-Directed Placement* [45] (1975) and the *Min-Cut* algorithm [46] (1977), the task of placement is by far the most actively investigated EDA topic in the analog domain. For that reason, the representative selection of placement approaches shown in Table 3.2 focuses on works that have been published within the last ten years. For each of the listed approaches, the table indicates which types of placement constraints are taken into consideration respectively. And as can be seen, there is *no* single approach that covers *all* types of placement constraints. A closer look reveals that two of the constraints depicted in Figure 3.3 are missing from Table 3.2, namely Equal Parameters and Orientation. This is because virtually every placement approach (and definitely every one of those listed in the table) supports the Symmetry or Symmetry Island constraint, which implies that the equality of the respective components' parameters and the consistency of their orientation is also taken into account. Therefore, it is superfluous to list those two constraints in the table individually.

A major deficiency among existing placement approaches, apart from a few works such as [64] or [65], is their lack of support for custom device interdigitation. Metaphorically speaking, device interdigitation is the staff of life in practice, where high accuracy requirements for a group of devices are served by splitting each device into a set of smaller devices such that they can be intermingled with each other in a two-dimensional fashion. To that end, the devices are usually dispersed throughout a matrix-like array according to a custom interdigitation pattern which yields a common centroid arrangement. However, this goes beyond a mere Common Centroid constraint because the granularity and the concinnity of the interdigitation determines how good the matching of the device group really is in the face of parasitic disturbances.

Apart from this device interdigitation, there are several other placement constraints which cannot be addressed with existing approaches so far. As an example, temperature variations on the chip can lead to thermal gradients which entail a critical discrepancy in the electrical behavior of certain design components. To avoid this kind of mismatch, a custom remedy is to place these components on an isothermal curve. Unfortunately, no placement algorithm is capable of considering this practice through a formal constraint (although such a constraint is probably even more suitable for algorithmic consideration than the prevalent –but troublesome– attempts in device matching, where the established types of constraints

[2]A floorplan is denoted as a slicing floorplan if it can be obtained by recursively bisecting a rectangle via horizontal and vertical lines, otherwise it is called a nonslicing floorplan [43].

Table 3.2: Comparison of selected placement approaches for analog layout design. The abbreviated constraints are: Alignment (AL), Abutment (AB), Boundary (BD), Clustering (CL), Common Centroid (CC), Distance (DS), Fixed Outline (FO), Preplacement (PP), Symmetry (SY), and Symmetry Island (SI).

Ref-	**Year of**				**Considered Constraints**							**Deter-**	**Solution**
erence	**Public.**	AL	AB	BD	CL	CC	DS	FO	PP	SY	SI	**ministic**	**Approach**
[47]	2006	x				x				x		yes	Less Flexibility First
[48]	2006	x	x	x			x	x	x	x		no	Simulated Annealing
[49]	2007									x		no	Simulated Annealing
[50]	2007										x	no	Simulated Annealing
[51]	2008						x			x		no	Simulated Annealing
[52]	2008		x			x	x			x		yes	named Plantage
[53]	2009									x		no	Simulated Annealing
[54]	2009				x						x	no	Simulated Annealing
[55]	2009				x	x				x		no	Simulated Annealing
[56]	2010			x							x	no	Simulated Annealing
[57]	2010				x	x	x			x		no	Simulated Annealing
[58]	2011			x		x					x	no	Simulated Annealing
[59]	2011	x	x	x		x	x	x	x	x		no	Simulated Annealing
[60]	2012									x		no	Simulated Annealing
[61]	2013	x	x	x			x	x	x	x		no	Simulated Annealing
[62]	2015						x			x		no	Simulated Annealing
[63]	2016	x				x	x			x		yes	Microsoft Z3 SMT

are meant to achieve electrical symmetry via geometrical symmetry). A similar constraint not supported so far is the requirement of placing components on an isobar to prevent piezoelectric effects.

Another problem with optimization-based placement approaches is that the components to be placed are commonly treated as black boxes: this does not pay respect to the mutual interdependency between a component's layout and the placement. For example, let there be two transistors: one with a bulk pin to its right and one with a bulk pin to its left, as in Figure 3.9 (a). Placing these transistors side by side can be done as in (b) such that their bounding boxes adjoin, but a more elegant solution is shown in (c) where the two bulk pins lie on top of each other. For that purpose however, also known as *diffusion sharing*, the algorithm must be able to acknowledge that such an overlapping of the transistors' bounding boxes is indeed valid, i.e., the algorithm must have an understanding of the transistors' inner layout. Things can get even more difficult: if the transistors are rotated as in (d), the bulk pins lie on opposite sides. In that case, it could be convenient to change the transistor parameters so as to modify the bulk locations (e) such that the transistors can again share their bulk diffusions (f). Yet, no placement algorithm can be expected to have the finesse for taking care of such subtle –but important– issues.

As is the case with the floorplanning approaches discussed before, most placement approaches suffer from the detriment that they are not deterministic. Again, Table 3.2 certifies that the stochastic Simulated Annealing algorithm is by far the most popular choice of optimization engine (also see [66]), despite the resentment that its randomized behavior has been courting in the design community. Even if this and all the other shortcomings described so far are disregarded, there remains the undeniable point of criticism that detaching the task of placement from the task of routing also detaches the expectable solutions from the requirements of reality. Although some approaches (including [56], [58], [60], and [61]) take certain routability considerations into account during placement, they are still subject to a separate routing step.

(3) Routing

While routing algorithms have already been proposed in the 1960s (e.g., Lee's maze router [67] and Hightower's line router [68]), Table 3.3 lists a representative selection of modern approaches and

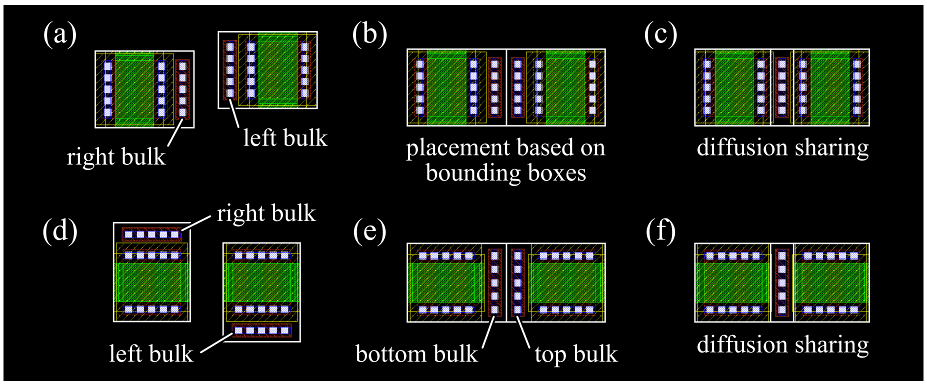


Figure 3.9: Example showing the interdependency between a component's layout and the placement.

indicates which of the routing constraints illustrated in Figure 3.6 are taken into consideration by each of these approaches. The Blockage constraint is omitted from the table since the ability to avoid layout obstacles is an essential claim for any routing algorithm. As can be seen, the constraint coverage here is even lower than with the placement approaches of Table 3.2 because many routing approaches are rather focused on just one or only a few particular constraints. Furthermore, it is again the case that other constraints which play important roles in real designs (such as complying with a maximal voltage drop, for example) are still absent from algorithmic consideration so far.

Table 3.3: Comparison of selected routing approaches for analog layout design. The abbreviated constraints are: Layer Limitation (LL), Wire Width (WW), Sensitivity (SN), Double-Cut Vias (DV), Length Matching (LM), Common Centroid (CC), Symmetry (SY), and Topological Matching (TM).

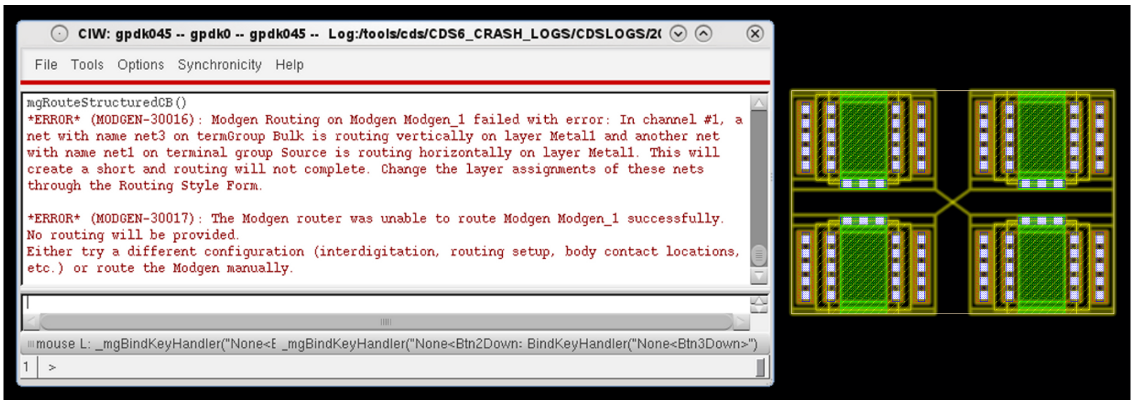
Ref-erence	**Year of Public.**	LL	WW	SN	DV	LM	CC	SY	TM	**Solution Approach**
[69]	2001		x							current-driven Steiner tree generation
[70]	2006		x							multi-expansion-step maze algorithm
[71]	2009								x	dynamic programming + A*-heuristics
[72]	2010			x		x		x		A*-search + maze algorithm
[73]	2011			x					x	gridless routing based on A*-search
[74]	2012			x	x			x		evolutionary computation
[75]	2012	x				x				tile-based detouring + rip-up & re-route
[76]	2014	x				x	x	x	x	integer linear programming
[77]	2014								x	integer linear programming
[78]	2015		x			x		x	x	integer linear programming

Apart from the evident inability to consider all types of routing constraints, an even more primal grievance is that some formalized constraint types are purely theoretical. As an example, [76] introduced the Common Centroid constraint for routing, claiming that it can reduce unwanted electrical effects and balance the parasitic resistances and capacitances. However, this argument is dubious since such parasitics do not depend on the mere symmetry of the wires but on (the interplay with) their surroundings. Hence, routing two wires in a common centroid fashion can even be unfavorable as it may induce additional unwanted coupling with the wires' neighborhoods. More accusatory, one might come to see this as a sad example of questionable EDA activity: establishing merely academical constraint types due to their suitability for algorithmic consideration but far from the necessities of reality.

Table 3.3 does not mention whether a routing approach is deterministic or not since this information is hard to obtain from the respective publications. Even if an approach utilizes a deterministic algorithm as its basic search engine, some form of randomization may have been introduced such that the overall behavior is nondeterministic (and thus again a barrier for the required constraining). Although Simulated

Annealing can be used for routing, most existing approaches resort to other solutions. As the table shows, there seems to be a trend towards *integer linear programming* at the moment. This may be due to the fact that integer linear programming facilitates a constructive routing method with parallel net consideration, as opposed to a sequential one where the nets are routed one by one. Other parallel or quasi-parallel routing methods involve hierarchical approaches [79] or rip-up and re-route techniques [80].

Despite such concepts, routing algorithms still do not live up to the difficulties in the analog domain when multiple nets need to be handled simultaneously. One notorious difficulty stems from the necessity to arrange devices in a highly interdigitated fashion: the problem for the routing is that such an arrangement entails a lot of wire crossings. In the digital domain, the number of wire crossing can be greatly reduced in advance by being considered during placement, whereas the need for interdigitation in analog designs rather exacerbates that problem. As an example, Figure 3.10 shows –only– four interdigitated transistors that are to be connected in a crosswise manner. Using a state-of-the-art auto-router, an error message is displayed saying that the auto-router is unable to perform the desired routing. The auto-router's recommendation to let the user try a different interdigitation is apt evidence of the described dilemma, i.e., having either a good arrangement or good routability – but not both.


CIW: gpdk045 -- gpdk0 -- gpdk045 -- Log:/tools/cds/CDS6_CRASH_LOGS/CDSLOGS/2(
File Tools Options Synchronicity Help

```
mgRouteStructuredCB()
*ERROR* (MODGEN-30016): Modgen Routing on Modgen Modgen_1 failed with error: In channel #1, a
net with name net3 on termGroup Bulk is routing vertically on layer Metal1 and another net
with name net1 on terminal group Source is routing horizontally on layer Metal1. This will
create a short and routing will not complete. Change the layer assignments of these nets
through the Routing Style Form.

*ERROR* (MODGEN-30017): The Modgen router was unable to route Modgen Modgen_1 successfully.
No routing will be provided.
Either try a different configuration (interdigitation, routing setup, body contact locations,
etc.) or route the Modgen manually.
```

```
mouse L: _mgBindKeyHandler("None<E _mgBindKeyHandler("None<Btn2Down: BindKeyHandler("None<Btn3Down>")
1 >
```

Figure 3.10: An auto-router's inability to connect four interdigitated devices in a crosswise manner.

Equivalent to the strong interdependency between the components' inner layouts and the placement (see Figure 3.9 above), a suchlike mutual relationship can also be observed between the component layouts and their routing. For instance, Figure 3.11 (a) shows a transistor with three separate gate fingers. Such a transistor usually features a dedicated parameter by which the underlying generator can be told to connect these gate fingers automatically (b). Although this feature is quite welcome in most cases, it can also be obstructive for certain routing requirements. Image (c) exemplifies this with a close-up section from an industrial layout, showing four three-finger transistors connected in a crosswise manner (as would have been desired in Figure 3.10). In consideration of the antenna affect (already discussed in the context of Figure 3.8), the transistor gates are connected using only metal1. As can be seen, this achievement involves an inventive routing solution with unconventional nonrectangular gate extensions where each gate contact covers only one specific half of the gate. To achieve such a routing, an algorithm would have to realize the impropriety of the transistors' own gate connections (b) and turn them off – hard to imagine how an algorithm could be smart enough for such an accomplishment.

Altogether, the routing task is a splendid example for the practical arduousness of constraint formalization. During design, the human brain's aptitude for visual thinking often lets a layout expert come up with inventive routing solutions, which are extremely cumbersome to circumscribe in formal terms, i.e., as constraints. Even if all these constraints are formally articulated (requisites (1)–(3) in Figure 3.2) and the routing algorithm is indeed able to find a solution that satisfies all these constraints (requisites (4)–(6) in Figure 3.2), this proceeding can be kind of perverted when the designer formalizes the expert knowledge such that it makes the routing algorithm find the layout solution that the designer already had in mind from the beginning. As long as the drawing effort for realizing that layout solution by hand is less than the effort for appropriately formalizing the design constraints, an algorithmic routing approach cannot compete with expert manual design.

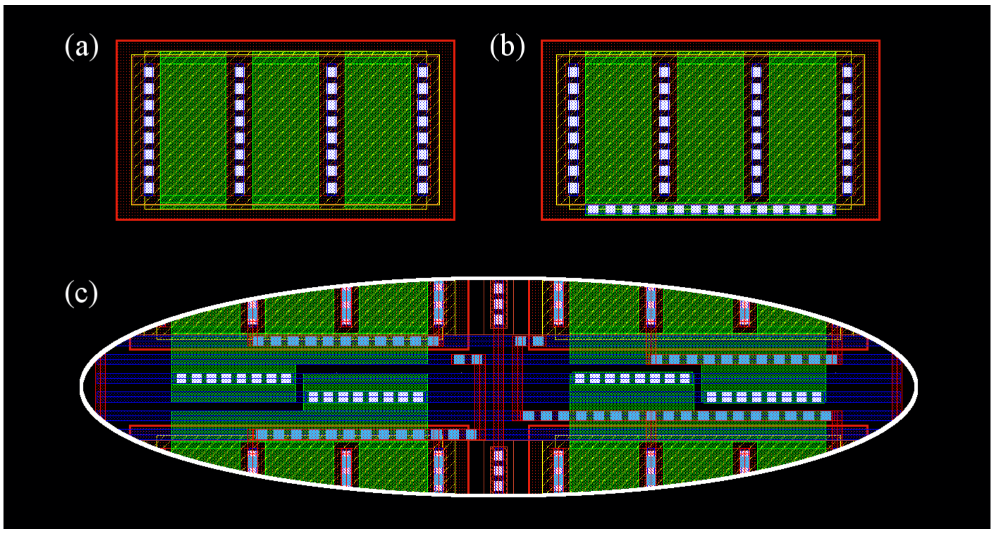


Figure 3.11: Example showing the interdependency between a component's layout and the routing.

(4) Combined Placement and Routing

While algorithmic routers may have their merits on higher design levels, they face immense difficulties coping with the layout necessities at device level (as evidenced by the previous examples). And it is in particular at device level, where the trouble with separating the routing task from the placement task becomes most obvious. To illustrate this, Figure 3.12 (a) shows eight transistors which have been placed in a way that pays respect to several matching principles such as alignment, orientation, and interdigitation. Image (b) displays a routing solution obtained with a commercially available auto-router. On the one hand, the routing itself is qualitatively poor due to its asymmetric wire guiding, excessive path lengths, use of single-cut vias, and disproportionately large area occupation. On the other hand, this in turn has a negative impact on the matching that had been established before: as can be seen, the transistors have been torn apart and re-oriented by the auto-router. The alternative solution of (c), obtained with another industry-standard tool, depicts the penalty for keeping the transistors in place: the majority of the wires run above the transistors (thus violating the Blockage constraint). In contrast, the manual solution in (d) demonstrates a highly compact routing which succeeds in satisfying both objectives: preserving the desired placement without leading any wires above the devices.

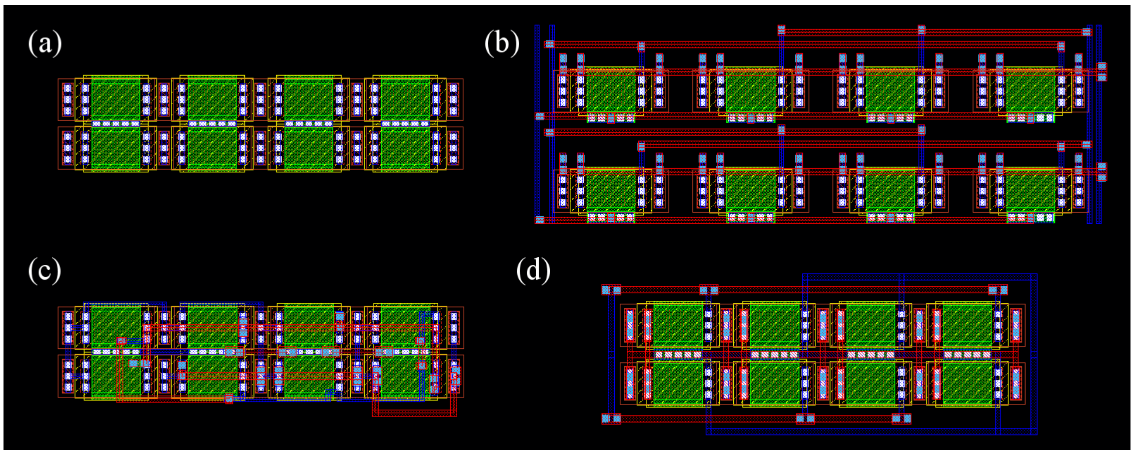


Figure 3.12: Example illustrating the importance of performing placement and routing in unison.

Although the example of Figure 3.12 underlines the importance of addressing the tasks of placement and routing in unison, most approaches follow a divide-and-conquer practice of separating the placement task from the routing task. This practice can not only be found in the purely placement-oriented and purely routing-oriented works discussed above, but also in the host of fully-automatic layout synthesis tools from around the year of 1990 (see Figure 1.2). Likewise, many such tools from the current millennium continue this habit of *place-and-route*.

For example, the ALDAC tool [81] from 2002 first obtains a placement with Simulated Annealing, and then performs an algorithmic routing (using two metal layers – one for vertical wire segments and one for horizontal wire segments) which is not further described. The layout system DTA [82] (2004) also uses Simulated Annealing for the placement task, while the subsequent routing is done via an algorithm merely referred to as a shortest path finding algorithm without providing any details. The SDAPS approach [83] –also published in 2004– divides the placement into a core-circuit placement (achieved with an unspecified deterministic algorithm) and a bias-circuit placement (again based on Simulated Annealing) and also separates the routing of critical nets (done with the A*-search) from the routing of general nets (performed via some improved maze routing algorithm).

The LAYGEN system [84] from 2005 employs evolutionary computation for the placement task, which is then followed by a two-stage routing process where a predefined prototype routing is first adjusted to the target circuit and then algorithmically optimized via genetic mutations. LAYGEN II [85] (2012) extends this approach to a fully automatic from-scratch layout generation of the routing in case no prototype is given; still, the routing process is kept separated from the placement. Together with the circuit synthesis tool GENOM-POF [86], LAYGEN II has been integrated into the design automation environment AIDA [87] in 2012.

More interesting are those layout automation approaches which combine placement and routing to some extent. As an example, the work of [88] (2004) proposes a two-stage placement two-stage routing technique: a global placement is performed at first via a combination of Simulated Annealing with evolutionary computation, to be then followed by a detailed placement wherein a minimum-Steiner-tree-based global routing is done, before finally a detailed routing is obtained through a modified maze router. Unfortunately, it is not reported which types of constraints this approach is able to take into account, but the follow-up publication [89] (2006) of that work mentions support for Symmetry and Proximity constraints during placement.

In [90] (2010), the routing is not really incorporated into the placement but at least considered therein. The placement engine –again based on Simulated Annealing– is capable of handling Symmetry, Common Centroid, and Clustering constraints. By taking congestion into account, the placer attempts to leave sufficient space between the devices with regard to the subsequent routing. Then, the nets are routed one by one on two metal layers, using a multi-pin maze routing algorithm which is able to perform symmetric routing. To route two nets symmetrically, one of them is routed first and will then be mirrored to produce the routing of its counterpart.

A similar approach is provided in [91] (2010), which is dedicated to the issue of device symmetry and wire symmetry. The proposed placement algorithm considers the Symmetry and Proximity constraints and also takes wire symmetry into account during the placement, not only for wires between the symmetric devices but also for connections with other devices. In contrast to the majority of other works, the placement is performed with a deterministic algorithm called DeFer [92]. After the placement, the symmetric routing is obtained with a pattern router or a maze router, although it is conceded that manual work may be necessary if certain nets are unroutable.

The work of [93] (2013) contributes an automatism for simultaneous placement and routing. The presented technique consists of two phases: in the first phase, a dynamic-programming-based global routing via A*-search is performed in conjunction with an integrated packing procedure; in the second phase, a performance-aware algorithm creates the detailed routing. These two phases are repeated in a Simulated Annealing optimization loop until the final solution is obtained, taking into account Wire Width and Symmetry constraints as well as considering the overall direction of the current flow while minimizing the total wirelength, bend number, and via count.

In [94] (2016), a deterministic mixed-signal layout synthesis approach extending the DeFer algorithm (see above) is described. The approach can generate various mixed-signal layout solutions through the integration of routing path planning, detailed placement, and detailed routing during an enumerative packing based on shape functions and dynamic programming. The packing procedure explores possible placements and considers Proximity, Symmetry, and Symmetry Island constraints while minimizing the total wirelength and separating analog signal paths from digital signal paths to eliminate switching noise. The detailed routing is done via wire ordering (to avoid intersecting wires) followed by the application of a channel router.

Despite the recent advances towards combined placement and routing, these approaches are still far from mastering all the other problems related to formal constraint consideration as discussed before. On the contrary, it even seems that performing placement and routing –more or less– hand in hand can only be achieved at the expense of other sacrifices, e.g., conceding that only few types of constraints are taken into consideration (compared to the placement-only and routing-only approaches listed in Table 3.2 and Table 3.3). Then again, with the prospective ability to consider many more placement and routing constraints simultaneously, the other side of the coin is that the troubles wander onto the shoulders of the user, raising the difficulty to formalize all these constraints in a self-consistent representation that is both as strict as necessary (to satisfy all design requirements) and as loose as possible (to exploit all degrees of freedom).

3.1.1.4 Further Topics

To be taken into account by an optimization-based layout automation approach, design constraints not only have to be algorithmically processible but also need to be available in the appropriate hierarchical design scope. Thus, apart from the representation and consideration of constraints, the EDA branch of *constraint engineering* deals with a couple of further topics briefly outlined as follows:

- *Constraint generation* denotes the automatic derivation of constraints based on a topological circuit analysis. For example, this can be achieved by identifying particular subcircuit structures in a given schematic design, for which the subcircuit-specific constraints can then be generated as done in [95] or [96].
- *Constraint unification* is a term used in [97] to denote the formalization of constraints in a unified, general description format (opposing a self-defined format as found in many other works). The authors propose a universal and easily extensible constraint semantic based on OpenAccess [98], an API (application programming interface) for EDA tool integration aimed at providing interoperability between IC design environments.
- *Constraint transformation* is the method of mapping an abstract constraint into a more concrete constraint or vice versa [99]. Thereby, the constraint nature might change, e.g., an electrical constraint such as a maximal voltage drop is turned into a physical constraint such as wire length [100].
- *Constraint propagation* passes constraints through different levels of the design hierarchy such that they become available in the respective cell [101]. Especially constraints that form relations between components in different hierarchical contexts are of pivotal importance here [102]. A systematic classification of the constraint propagation problem is given in [103].
- *Constraint resolution* can be used to make logical conclusions on a set of related constraints, which allows to simplify them, to resolve mutual dependencies, and to discover conflicts. As an example, the constraint engineering system [104] performs constraint resolution via concepts of constraint logic programming [105].
- *Constraint management* is attained with a software architecture that enables the storage, access, organization, and synchronization of constraints [106]. To keep the constraints up to date and guarantee their consistency and integrity, a close interaction between the management system and the design data is required [107].

Although these topics seem to be (and indeed may be) fruitful and beneficial aids for algorithmic constraint consideration, they can also be regarded as collateral requisites that are only made necessary by an algorithm's dependence on a formalization of the design problem. From that point of view, constraint engineering largely represents a massive overhead that could be eluded if many constraints are taken into account in a different –i.e., nonformalized– way.

3.1.1.5 Optimization Algorithms – Conclusion

In a nutshell, the powerful versatility of optimization-based layout automation is counterpointed by the need and the difficulty of comprehensively considering all design constraints in a formalized way. The various problems discussed in this Section 3.1.1 can be summarized and categorized according to the six requisites in Figure 3.2 as follows:

(1) *Semantics*:
- Ineptness of turning geometrical conceptions into adequate formal expressions.
- Close interdependency between component layouts and placement requirements.
- Close interdependency between component layouts and routing requirements.

(2) *Syntax*:
- Lack of syntactical support for articulating the permitted layout variability.
- Missing facilities for specifying necessary, uncritical constraint exceptions.
- Many practically relevant constraints not yet formally describable at all.

(3) *Formulation*:
- Unreasonably high constraining effort due to inadequate formal constraint types.
- Constraining becomes unintuitive because of nondeterministic algorithmic behavior.
- Auxiliary effort to make constraints available in the current hierarchical context.

(4) *Consistency*:
- Formal overconstraining hard to detect, leading to unresolvable contradictions.
- Functional constraints and design-methodical restrictions coming into conflict.

(5) *Processibility*:
- No known approach is able to consider all existing types of formalized constraints.
- Some constraint types are academically interesting but bear no relation to reality.
- Enhancing a specific ability of an algorithm compromises some of its other qualities.

(6) *Computation*:
- Challenge of considering multiple converse design requirements simultaneously.
- Focus on placement or routing instead of performing both tasks in unison.
- Rejection of acceptable solutions due to marginal constraint violations.

The difficulties with formal constraint handling are further accompanied by the necessity to diminish the solution space via a simplification of the design problem. This entails a trade-off between the optimality of the best theoretical solution and the algorithm's ability to find that theoretical optimum. All these considerations illustrate why pursuing purely optimization-based automation *alone* is far from able to cope with the More than Moore complexity of analog layout design.

However, one should not come to think that such approaches are of no avail – they have just been devised as "universalists": an optimization algorithm can be versatilely applied to various circuit classes, but concedes losses in terms of layout quality. Thus, the bottom line for this work is to take advantage of that versatility while countervailing the qualitative shortcomings of optimization-based approaches with other automatisms. This intention reaches out to Section 3.1.2 about procedural generators.

3.1.2 Procedural Generators

In layout design, a procedural generator is a parameterized design entity defined by two things: a specification of its supported input parameters and a script-like, successive command sequence that implements its functionality. As indicated in the User View of Figure 3.13, most of these commands are simple design operations (e.g., draw, copy, move) reflecting the duties of manual "polygon pushing". In action, a procedural generator takes a set of user-given input parameters (more precisely: parameter values) and follows the sequence of commands to produce a customized layout *result* for a specific design component. That result is namely an output of polygons whereas (in contrast to optimization algorithms) the cognitive layout *solution* is not found at runtime, but was preconceived by the design expert who implemented the generator (see **EDA View**).

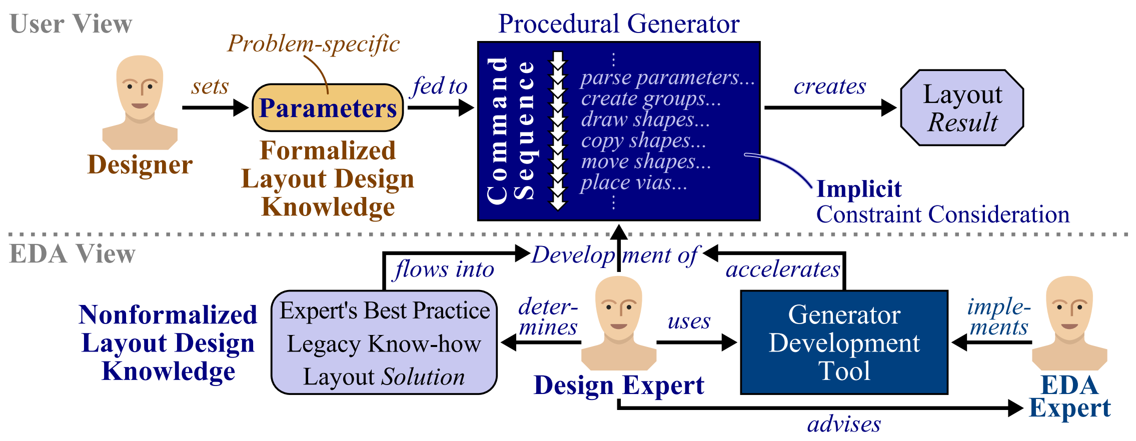


Figure 3.13: Working principle of a procedural generator.

Before addressing the **EDA View** of Figure 3.13 in the remainder of this chapter, Section 3.1.2.1 first discusses some important fundamentals of procedural generators.

3.1.2.1 Discerning Masters and Instances

As is the case with ordinary library components (see Section 2.1.3), procedural generators are usually also made available as *cells* –which can be used by being instantiated– and are thus commonly referred to as *parameterized cells* (PCells). In that regard, a subtle distinction should be made between the generator itself, which is called the *master*, and its concrete instantiations in the design, which are denoted as *instances*. Formally, a procedural generator can be defined like a mathematical function g:

$$g: \quad X \to Y, \tag{3.1}$$
$$x \mapsto y. \tag{3.2}$$

In that notation, where g can be considered as the *behavior* of the generator, equation 3.1 represents the master and equation 3.2 represents an instance. The *domain* X (domain in the mathematical sense) is given by the set of input parameters supported by the generator. For a set $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of n input parameters, the generator's domain X is the Cartesian product of the individual parameters' domains $\mathbb{D}_I$: $X = \mathbb{D}_{I_1} \times \mathbb{D}_{I_2} \times \cdots \times \mathbb{D}_{I_n}$.

The *codomain* Y depends on the type of the generator, which is not necessarily a layout generator but can for example also be a circuit generator that is able to create a schematic circuit. In the latter case, Y can be denoted as $\mathbb{S}$ (the *universe* of schematic circuits), in the former case as $\mathbb{L}$ (the universe of layout designs). The mathematical *image* $g(X)$ of the procedural generator is always a subset of that codomain: $g(X) \subset Y$. When being instantiated, the generator takes x, which is a set of parameter values, and produces $y = g(x)$, which is a circuit or layout contained in that image, i.e., $y \in g(X)$. The question of how the generator is able to handle design constraints thereby, will now be addressed in Section 3.1.2.2.

3.1.2.2 Constraint Handling

Apart from design automation, one other major approach to increase design productivity is *re-use*. However, the re-use of existing, handcrafted layouts is impeded by several reasons: (a) circuits are too application-specific, (b) small circuit modifications heavily impact the layout, (c) the semiconductor technology may change, and (d) the shape of the layout doesn't fit into the available space. The common problem behind all these reasons is basically the same: a fix layout has no degrees of freedom [7]. Hence, re-using such fix layouts inevitably requires manual adjustments, which often leads to an unsystematic and unmanageable proliferation of layout variants.

Procedural generators resolve this problem through parametrical generalization and thus naturally combine the two main efficiency-raising measures available in analog layout design by facilitating *automation* through *re-use*. In that manner, the re-use occurs on a higher level of abstraction since it is not achieved by simply duplicating a singular layout, but by imitating a human expert's *design procedure* of engineering such a layout. This augments the copy-paste fashion of layout re-use to a more sophisticated form of re-using expert knowledge – which inherently includes the consideration of crucial design requirements. That merit already holds good for primitive devices, but becomes even more adjuvant regarding generators of entire modules.

So, although a procedural generator "only" replicates a layout expert's best practice, it represents an effective instrument to capture practical know-how in a straightforward automatism, utilizing a design team's legacy portfolio of IC projects as a valuable resource of field-tested, silicon-proved layout solutions. While the input parameters of a procedural generator can be regarded as formalized layout design knowledge allowing to customize the generator's output for a specific design problem (see **User View** in Figure 3.13), the automatism in itself already contains an essential amount of layout design knowledge in a nonformalized way since the layout solution was preconceived in advance (see **EDA View** in Figure 3.13). This trait gives procedural generators the advantageous ability to consider design constraints implicitly, i.e., without the need to formalize them.

As an example, Figure 3.14 (a) shows an instance of a procedural generator creating a Differential Pair layout for two transistors A and B. Thereat, the generator implicitly takes care of all requirements that are crucial to achieve a highly accurate two-dimensional matching without having been explicitly told to do so. As discussed in Section 3.1.1.2, these can be divided into placement constraints and routing constraints:

- Placement:
  - the two transistors are interdigitated by being split into a cross-coupled AB/BA array of four devices (also known as a so-called Quad layout),
  - the devices are identical and are aligned according to a common centroid arrangement,
  - each device is flipped in a different way such that they are all oriented symmetrically to the overall center point,
  - the placement is highly compact, i.e., vertically the devices are placed as close as possible while preserving sufficient space for lateral wires, and horizontally the devices are abutted such that they even share their bulk pins to save space.
- Routing:
  - all routing wires run alongside the devices and thus not on top of them,
  - only two metal layers (metal1 and metal2) are utilized for the entire routing,
  - the overall routing scheme effectuates electrical symmetry and homogenous wire density,
  - every employed via has two clearance holes, i.e., no single-cut vias are used.

Notwithstanding the implicit consideration of all these constraints, one should observe that the routing of the transistor gates in the layout of instance (a) includes horizontal metal2 wires. For some semiconductor technologies this is not a problem, but for others it is critical regarding the already mentioned antenna effect. On this account, the generator provides a parameter by which the routing can be switched into an alternative variant exemplified in instance (b). Here, the transistor gates are automatically connected using only metal1, following the inventive manual routing solution with nonrectangular gate extensions known from Figure 3.11 (c).

Beside such powerful abilities in terms of constraint consideration, procedural generators go strong regarding several other aspects that have been identified as weaknesses of optimization algorithms in Section 3.1.1. Above all, Figure 3.14 demonstrates the natural ability of procedural generators to create layouts wherein devices are simultaneously placed *and* routed. Furthermore, procedural generators elude the need for an abstract mathematical problem modeling and design-methodical standards. Likewise, neither runtime nor randomness are an issue because the execution of the generator's command sequence proceeds fast and deterministic without resorting to heuristics.

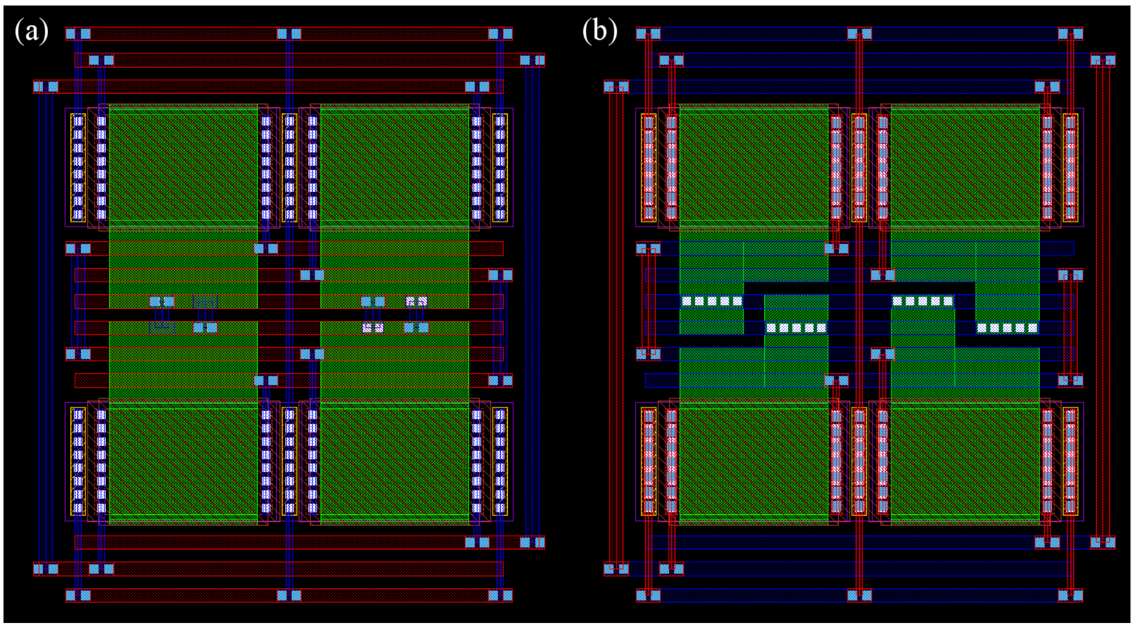


Figure 3.14: Instances of a procedural generator, creating a cross-coupled Differential Pair layout.

All these assets are based on the fact that every procedural generator targets one specific type of design component. Apart from primitive devices, procedural generators are particularly expedient for automating simple modules as defined in Table 2.1, i.e., basic circuits with high regularity in the layout (as is the case in Figure 3.14) for which feasible layout solutions are well-known from years of practical experience. This is why procedural generators are of much greater interest in industry than in academia, which in turn explains why –in contrast to optimization algorithms– the amount of publications on generator-based approaches is rather small. Beside the procedural generators themselves, a more elementary issue is the question which programming languages can be used for their implementation, as is about to be covered in Section 3.1.2.3.

3.1.2.3 Generator Programming Languages

In terms of computer science, one can distinguish between (1) *general-purpose languages* without particular support for generator programming, and (2) *domain-specific languages* [108] that have been specifically developed for that purpose.

(1) General-Purpose Languages

Today's semiconductor industry is dominated by the three IC development frameworks of the three market-leading EDA companies, and each of these frameworks features its own general-purpose language that can be used for generator programming.

Mentor Graphics' *Pyxis Custom IC Design Platform* provides AMPLE (Advanced Multi-Purpose LanguagE), a C-like scripting language which enables the coding of parameterized cells. In the *Galaxy Design Platform* by Synopsys, the interpreted high-level programming language Python is used for the implementation of so-called PyCells. The Cadence *Virtuoso* design environment has the proprietary scripting language SKILL[3] for PCell development [109]. SKILL is based on the functional programming language Scheme, which itself is a dialect of the list-based programming language Lisp. SKILL has also been extended into another dialect called SKILL++ in order to support object-oriented programming.

SKILL-based PCells can be converted into Python-based PyCells using the programm Sk2Py [110]. Through the plug-in mechanism of OpenAccess it is also possible to include procedural generators written in C++ [111].

[3]SKILL is not an acronym – it is a name.

(2) Domain-Specific Languages

Domain-specific languages for generator programming have already come up in the 1970s, but only a few selected works shall be outlined here. With STICKS [112] (1978), a design component can be described as a symbolic *stick diagram* which is then compiled into a DRC-correct mask layout. Layla [113] (1985) is a Pascal-based language for the description of hierarchical technology-independent layouts, which is able to translate an existing layout into a Layla program. This *design by example* methodology can also be found in the interactive approach of [114] (1990), where annotations in a handcrafted layout allow to specify instructions for the parameterized module generator that is produced from that layout. The approach also supports the creation of *circuit generators* able to create transistor-level schematics.

With the concept of [115] (1992), a parameterized cell can be obtained from a fix example layout in a graphical way through the definition of stretch lines, repetition groups, and conditional inclusions. The work of [116] (1998) presents the description language MOGLAN and provides a graphical user-interface (GUI) for writing, translating, executing, and debugging module generators. A comprehensive survey of generator description languages and concepts from before the year 2000 is given in [117]. After that time, the charm of graphical generator development support has directed EDA attention from programming language concepts towards the realization of professional generator development tools (Section 3.1.2.4). Still, a domain-specific language can serve as the intellectual foundation for a graphical tool – an idea that has also been employed in the context of this work (as will be covered in Section 8.3).

Figure 3.1 indicates that graphical generator development tools are favored over mere domain-specific languages since they can accelerate the implementation of procedural generators more effectively. And as will be seen in Section 3.2.3, providing such tools to capture a design expert's layout knowledge also represents one essential stepping stone for the automation philosophy behind this thesis.

3.1.2.4 Generator Development Tools

The following works are professionally implemented generator development tools, most of which are offered by corporate EDA vendors as commercially available software products. Such tools are meant to simplify the programming of circuit and layout generators by fostering the development of parameterized cells through design experts, providing graphical programming facilities to relieve these design experts from the burden of textual codification effort.

Freescale's *PCell Compiler* [118] is referred to as an apparatus that interprets a so-called structure layout and compiles a parameterized cell from it. Apart from determining and analyzing shape relationships in the structure layout, this method creates links between related shapes and maps them into a shape tree wherein dimension properties of related shapes are automatically calculated via shape generation functions. The approach is implemented as a computer program product, but no information is given about any GUI support.

The IPGEN *1Stone Developer* [119] is part of the 1Stone tool set that enables the implementation of parameterizable layout generators based on a generic engineering model (GEM). The methodology allows to interactively design these generators –denoted as analog IP (intellectual property)– based on GEM code, with support for development, execution, testing, and verification. Through a virtual-grid symbolic layout principle, where all technology-related data is represented via variable design rules, the analog IP is technology-independent. Circuit generators can also be created with 1Stone.

SpringSoft's *Laker Custom Layout Automation System* [120] facilitates parameterized generators denoted as user-defined devices (UDDs). Layout shapes can be imported into a UDD script and then be equipped with "constraints" that apply certain "procedures" to the shapes (e.g., an Align constraint calls an Align procedure). The development of an UDD is largely menu-based, but some graphical (i.e., layout-based) programming is also supported. For example, the edges that are to be aligned with the Align procedure can be directly selected on the layout shapes.

The *HiPer DevGen* (High Performance Device Generator) [121] by Tanner EDA is a generation engine that automatically creates the layout of primitive devices and simple modules, based on design inputs, matching requirements, and manufacturing design rules for the specific semiconductor technology. It is not reported, inhowfar HiPer DevGen supports the development of custom generators. HiPer

DevGen is built upon Tanner's existing T-Cell architecture of parameterized cells, which provides some utilities for T-Cell programming. This is done by drawing layout shapes and configuring operations that are applied to these shapes using StretchPort lines and RepeatGroups.

AnaGlobe's *Geometric Object Layout Formula* (GOLF) [122] is an OpenAccess-based layout editor that features an interoperable PCell design environment. It facilitates the creation of hierarchical layout generators based on the composition of layout components whose attributes can be parameterized through the hierarchy. Power is added to the generators via a set of predefined geometrical operations, while new ones can be specified by writing own code in C++, Python, Perl, or the tool command language TCL. The GOLF design environment also offers functionalities for previewing, debugging, and documenting the PCell that is currently being developed.

Ciranova *PyCell Studio* [123] is a group of tools that use a special Python API for the development of PyCells. The tools comprise a layout viewer, a Python shell programming environment, an integrated development environment (IDE) with debugging capabilities, and a plug-in for OpenAccess. Since the Python language supports object-oriented programming, the methods of the Python API are organized in classes that include shape and instance classes, design classes, technology classes, geometric classes, utility classes, connectivity classes, and classes for higher-level functions such as contact rings. However, the IDE does not support graphical programming, so the PyCell development occurs through purely textual coding.

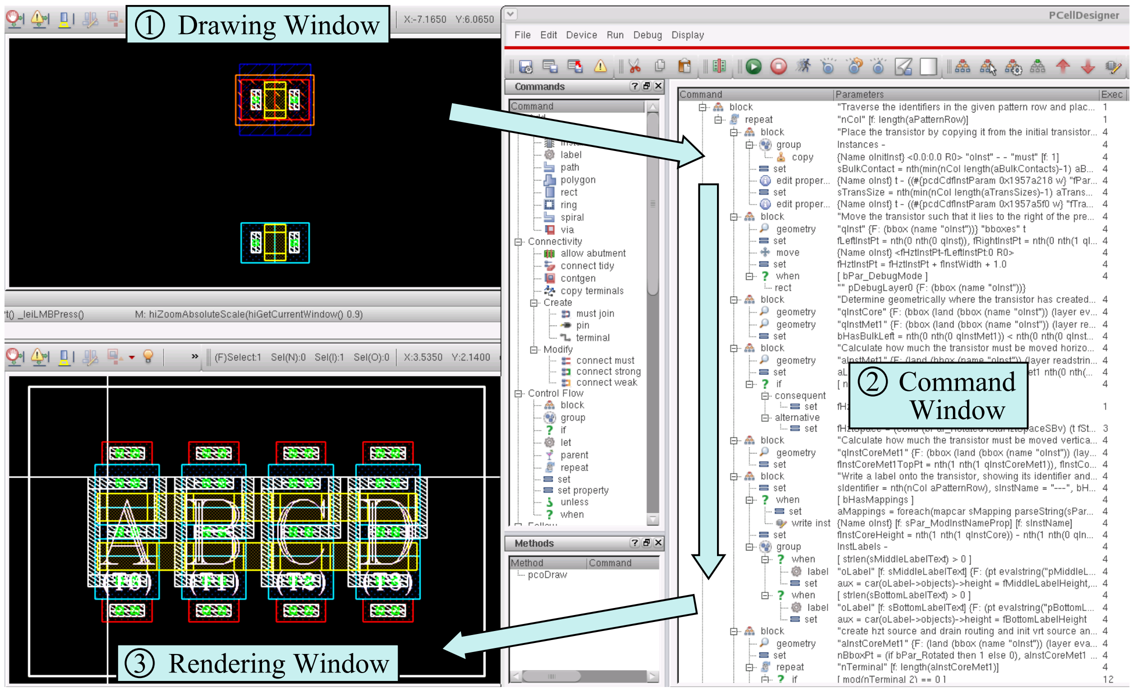
The *Berkeley Analog Generator* (BAG) [124] is an integrated framework for the development of circuit and layout generators in Python. It uses the Python API and the IDE of PyCell Studio and provides own helper classes (collections of template architectures and design routines) to ease the generator programming. For layout-related functions, these helper classes are referred to as layout styles. An example is the Array layout style, where an array of unit cells is generated according to a certain interdigitation pattern. The Array class can also be used to create flexible circuit structures such as a resistor string Digital-Analog Converter (DAC) with a variable number of bits. While BAG does not feature further development facilities in terms of graphical programming, its particular asset is that it addresses all steps of the design flow from testbench creation and circuit simulation to physical verification and extraction.

Fraunhofer's *IIP Framework* (IIP: Intelligent Intellectual Property) [125] provides an abstract programming interface –again based on Python– which can be used for developing object-oriented circuit and layout generators. A unique feature of the framework is that these generators can not only be developed in a highly technology-agnostic manner, but that they are also independent from the design environment. To employ the generators in a specific design environment, it is therefore necessary to implement an appropriate low-level interface which is able to access the respective design database via interprocess communication (currently, interfaces for the design environments of Cadence and Synopsys already exist). Another subtlety in that regard is that the generators are in fact not created as parameterized cells, but that they produce persistent library cells. Apparently, the IIP Framework neither comes with a GUI, nor does it enable graphical generator programming.

Cadence *PCell Designer* [126] is a layout PCell development platform based on an industrial inhouse methodology conceived at Bosch [127]. The PCell Designer GUI facilitates an advanced graphical PCell programming approach that consists of three windows shown in Figure 3.15. The (1) *drawing window* allows to draw layout shapes and to instantiate layout devices. These can be imported into the (2) *command window* where sophisticated GUI support helps to specify geometrical, arithmetical, logical, and other operations within a structured command tree that defines the execution flow of the PCell. The result of the PCell evaluation can be instantaneously previewed for any number of PCell samples instantiated in a (3) *rendering window*. Apart from debugging, documentation and deployment capabilities, PCell Designer enables

- working with hierarchically nested groups of shapes and subinstances,
- halo-crossprobing between the command window and the two layout windows,
- deriving PCells from other PCells in an object-oriented manner of inheritance, and
- articulating complex geometrical expressions with a dedicated *geometry query language*.

Although the beginnings of analog EDA date back to the 1980s, the history of professional generator development tools is not older than this millennium and still gains momentum. This observation reflects



```
Command                        Parameters                                                              Exec
block                          "Traverse the identifiers in the given pattern row and plac...          1
  repeat                       "nCol" [f: length(aPatternRow)]                                         1
    block                      "Place the transistor by copying it from the initial transistor...      4
      group                    Instances -                                                             4
        copy                   {Name oInitInst} <0.0:0.0 R0> "oInst" - - "must" [f: 1]                 4
      set                      sBulkContact = nth(min(nCol length(aBulkContacts)-1) aB...              4
      edit proper...           {Name oInst} t - ((#{pcdCdfInstParam 0x1957a218 w} "fPar...             4
      set                      sTransSize = nth(min(nCol length(aTransSizes)-1) aTrans...              4
      edit proper...           {Name oInst} t - ((#{pcdCdfInstParam 0x1957a5f0 w} "fTra...             4
    block                      "Move the transistor such that it lies to the right of the pre...       4
      geometry                 "qInst" {F: (bbox (name "oInst"))} "bboxes" t                           4
      set                      fLeftInstPt = nth(0 nth(0 qInst)), fRightInstPt = nth(0 nth(1 ql...     4
      move                     {Name oInst} <fHztInstPt-fLeftInstPt:0 R0>                              4
      set                      fHztInstPt = fHztInstPt + fInstWidth + 1.0                              4
      when                     [ bPar_DebugMode ]                                                      4
        rect                   "" pDebugLayer0 {F: (bbox (name "oInst"))}
    block                      "Determine geometrically where the transistor has created...            4
      geometry                 "qInstCore" {F: (bbox (land (bbox (name "oInst")) (layer ev...          4
      geometry                 "qInstMet1" {F: (bbox (land (bbox (name "oInst")) (layer re...          4
      set                      bHasBulkLeft = nth(0 nth(0 qInstMet1)) < nth(0 nth(0 qInst...           4
    block                      "Calculate how much the transistor must be moved horizo...              4
      geometry                 "a                                            layer readstrin...        4
      set                      aL                                            et1 nth(0 nth(...         4
      if                       [ n                                                                     4
        consequent
          set                  fH                                                                      1
        alternative
          set                  fH                                            aceSBv) (t fSt...         3
    block                      "Calculate how much the transistor must be moved vertica...             4
      geometry                 "qInstCoreMet1" {F: (bbox (land (bbox (name "oInst")) (lay...           4
      set                      fInstCoreMet1TopPt = nth(1 nth(1 qInstCoreMet1)), fInstCo...            4
    block                      "Write a label onto the transistor, showing its identifier and...       4
      set                      sIdentifier = nth(nCol aPatternRow), sInstName = "---", bH...           4
      when                     [ bHasMappings ]                                                        4
        set                    aMappings = foreach(mapcar sMapping parseString(sPar...                 4
        write inst             {Name oInst} [f: sPar_ModInstNameProp] [f: sInstName]                   4
      set                      fInstCoreHeight = nth(1 nth(1 qInstCore)) - nth(1 nth(0 qIn...          4
      group                    InstLabels -                                                            4
        when                   [ strlen(sMiddleLabelText) > 0 ]                                        4
          label                "oLabel" [f: sMiddleLabelText] {F: (pt evalstring("pMiddleL...          4
          set                  aux = car(oLabel->objects)->height = fMiddleLabelHeight,...             4
        when                   [ strlen(sBottomLabelText) > 0 ]                                        4
          label                "oLabel" [f: sBottomLabelText] {F: (pt evalstring("pBottomL...          4
          set                  aux = car(oLabel->objects)->height = fBottomLabelHeight                 4
    block                      "create hzt source and drain routing and init vrt source an...          4
      geometry                 "aInstCoreMet1" {F: (land (bbox (name "oInst")) (layer eva...           4
      set                      nBboxPt = (if bPar_Rotated then 1 else 0), aInstCoreMet1 ...            4
      repeat                   "nTerminal" [f: length(aInstCoreMet1)]                                  4
        if                     [ mod(nTerminal 2) == 0 ]                                               12
```

Figure 3.15: The three windows of the PCell Designer tool [126]: (1) drawing window, (2) command window, (3) rendering window.

a change of mind in EDA, encouraged by the insight that the practical benefit of procedural generators has not yet been exploited to the full degree. To do so, the biggest leverage is brought to bear by the advancement towards graphical generator programming concepts, letting designers automate their design routines in a straightforward way that hardly deviates from their daily business. This underlines the value of a designer's expert knowledge for analog layout automation.

Apart from graphical programming, the tools described above show that there are many other avenues for improving generator development (supporting both layout and circuit generators, hierarchical PCells, object-orientation and inheritance, technology-independence, design environment interoperability, export and import of IP, debugging and verification, etc.). The other side of the coin is found in the difficulty to meet all these opportunities with one single tool and to exploit all of them in practice without getting entangled in the details.

Even though significant progress in these directions is still expected to be made by further work on generator development tools, this can in the end "only" lower the implementation effort but does not exempt the generator developer from the duty to preconceive the generator in its entire variability and anticipate all design constraints in advance. Without overcoming that burden, the development of procedural generators reaches an inevitable limit as soon as advanced modules (in the sense of Table 2.1) are tackled. This obstacle is substantiated by implemented generator examples published in recent years, as follows in Section 3.1.2.5.

3.1.2.5 Implemented Generator Examples

Publication [128] presents a layout generator for a fully differential Operational Amplifier with folded cascode and class AB output stage. It consists of simple modules (again as defined in Table 2.1) which are assembled according to a *street principle* adapted from the structure of digital standard cells. A fix height –which can be specified through a parameter– is used for all of these modules while their width depends on the dimensions of their internal devices. The modules are positioned side by side across two lanes with their interconnections being drawn above, below, and in between the lanes. The generator has been developed with the 1Stone Developer tool. Also shown is an example of migrating another amplifier layout to a different semiconductor technology, but it is not made clear inhowfar that amplifier layout has been implemented as a generator.

In [129], a Current Mirror layout and a Capacitor Array layout is shown, produced via layout generators that facilitate interdigitated device arrangements. Furthermore, a layout generator for a single-ended Operational Amplifier is presented. Three different variants with different layout arrangements are depicted, but no precise information is given about the total variability of the generator. Finally, three different layouts for a single-ended single-stage Analog-Digital Converter (ADC) are depicted, as created by a dedicated generator. Regarding variability, again it is not reported how and to what extent the layout arrangement can be customized through parameters. The authors also mention the realization of circuit generators and hint at the implementation of parametric behavioral models within these generators. As above, the 1Stone Developer tool has been used for the generator development.

The approach of [130] provides a layout generator tool aimed at creating matrix-like layouts for analog basic circuits and arrays of passive devices. With a particular focus on device interdigitation, the tool consists of a pattern generator module for the automatic proposal of interdigitation patterns, a placement module able to construct an array of devices based on such an interdigitation pattern, and a routing module which facilitates different routing styles and variable wire widths. The creation of guardrings and dummy devices is also supported, but no further details are given about these features, nor about the implementation of the generator tool. Layouts for a Differential Pair and a Switched-Capacitor Integrator circuit are shown as examples.

Extending the work of [131], a highly flexible Current Mirror PCell has been developed in the scope of this thesis and presented in [132]. The immense variability of the PCell –created with Cadence PCell Designer– is attained by supporting a comprehensive set of degrees of freedom, which are:

- multiple topologies (simple, cascode, wide-swing),
- various transistor types (NMOS, PMOS, . . .),
- arbitrary device sizes (width, length, fingers, . . .),
- free scalability (number of devices, number of outputs),
- user-specified current mirror ratios (1:1, 2:3:4, . . .),
- multi-row interdigitation (AB/BA, ABA/BAB, . . .),
- different routing styles (symmetric, compact, . . .),
- detailed layout settings (spacings, wire widths, . . .).

Since it is neither possible nor practical to cover *all* degrees of freedom, finding the most profitable trade-off between parametrical customizability and PCell development effort is one major issue in this context. The PCell examples given in [132] represent only a marginal fraction of the total variability that is being covered. This demonstrates the respectable magnitude of functionality that can already be achieved with procedural generators for simple modules like this Current Mirror.

However, one has to realize that the development of such a module profits from the regularity in its layout arrangement and the identity of its internal devices. This means that many parameters (e.g., the device type or the device size) do not affect the essence of the resulting layout. Figuratively, it can be said that –for example– changing the number of devices only scales the layout in a kind of "linear" way since it adheres to the same generator behavior. In contrast, supporting different topologies is more disruptive because it typically requires an implementation of separate branches in the generator code. Yet, this is not problematic if only a small and finite set of topologies has to be covered, as is usually the case.

The situation is entirely different with advanced modules and larger blocks due to the irregularity in the arrangement of their submodules. This entails much more layout interdependencies between these submodules and thus blows up the generator's variability in a "nonlinear" way. To give an example, one may consider an advanced module such as in Figure 3.16 (a), consisting of ten simple modules (named M1 to M10). With a different sizing of the devices in module M1, the overall module constellation would not fit that well (b). Thus, making the layout more compact either asks for (c) a complete re-arrangement of all modules to eliminate dead space, or (d) turning some modules into another layout variant if the overall arrangement is to be maintained, e.g., due to certain placement constraints, or (e) both changing certain module variants and re-arranging the modules, e.g., to obtain a desired aspect ratio.

This example illustrates a fundamental problem: anticipating such contingent, application-dependent, high-level constraints (as opposed to the obligatory, type-specific, low-level constraints of simple modules) virtually compels a generator developer to preconceive the entire variability of the generator in

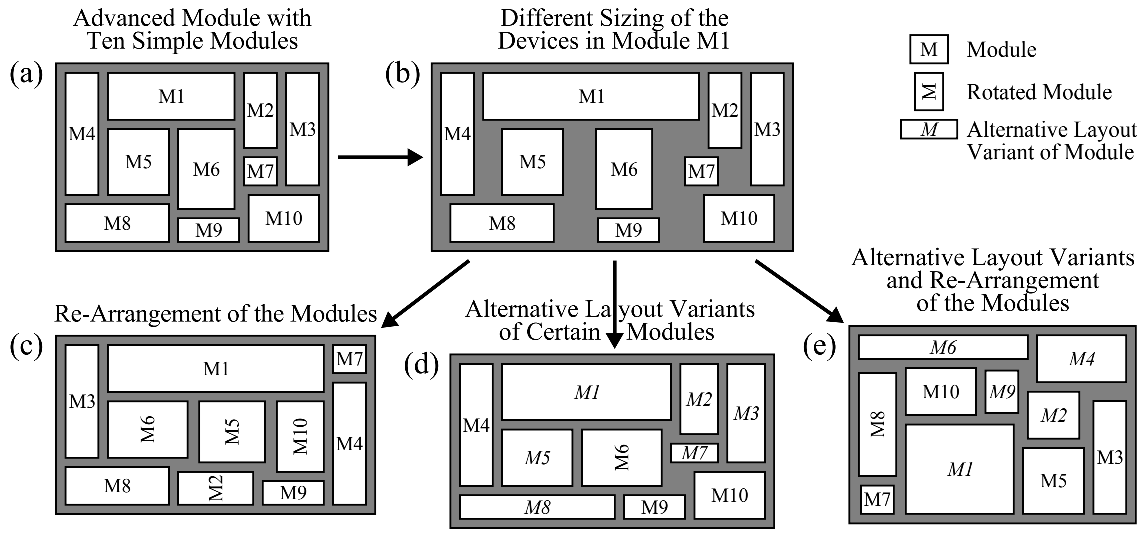

Advanced Module with Ten Simple Modules
(a)
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
Different Sizing of the Devices in Module M1
(b)
Module
Rotated Module
Alternative Layout Variant of Module
Re-Arrangement of the Modules
(c)
Alternative Layout Variants of Certain Modules
(d)
Alternative Layout Variants and Re-Arrangement of the Modules
(e)

Figure 3.16: Example showing the major degrees of freedom in the variability of an advanced module.

advance. The irony herein is that although such a constraint diminishes the potential solution space in the concrete case, it prevents a reduction of the solution space that is to be covered by the generator (and implemented by the generator developer) because the concrete case –and thus the prospective set of constraints– is not known at the time of development. Hence, the generator has to be prepared in advance for all possible situations that might be encountered during design.

In general, the variability of an advanced module consisting of several submodules must be able to cope with three basic requirements: (1) the device sizing given by the schematic circuit, (2) potential constraints for the positioning of the submodules, (3) the contour and size of the layout space available for the module, e.g., imposed by a Fixed Outline constraint. These requirements correspond to three major degrees of freedom that have to be supported by the module generator in accordance with each other: (1) different dimensions of the submodules due to all possible device sizes, (2) different arrangements of the submodules in all possible constellations, and (3) different layout variants of the submodules in all possible permutations.

With these degrees of freedom, the combinatorial growth quickly becomes immense: while the number of layout variants that need to be preconceived for the implementation of a simple module is discrete and finite, the variability of an advanced module is effectively continuous and infinite. So, this small step from the level of simple modules to the next higher level of advanced modules already is so drastic that the problem complexity escalates to an extent which defies a purely generator-based automation approach. This statement is backed up by the observation that the published examples of such advanced modules (e.g., the Operational Amplifier of [129] and the Switched-Capacitor Integrator circuit of [130]) do not mention which degrees of freedom are being covered in total so it can be assumed that only a small subset of the potential variability is supported. Just as well, reducing the degrees of freedom by sticking to a standard-height row-based placement as in the street principle of [128] is no feasible solution. So far, no attempt at establishing such “analog standard cells” has achieved evident success in practice.

In the end, it should be remembered that optimization algorithms find it particularly difficult to cope with the interdependencies between component layouts, placement and routing at device level (see Section 3.1.1.3) whereas these issues can be effortlessly mastered by procedural generators (Figure 3.14). On the other hand, procedural generators struggle with the interdependencies at module level when it comes to cover the immense variability stemming from the possible device sizings, different arrangements, and alternative variants of the modules (Figure 3.16) – a challenge where optimization algorithms score with the versatility of their self-intelligent solution finding. This exemplifies that the strengths of the two automation strategies may enrich one another, which fortifies the approach of this thesis.

Another strength of generator-based automation is elaborated in the work of [133]. It proposes a technique for the construction of procedural generators that are capable of producing layouts with 45-degree polygons. The geometric structures are described in a format called *parameterized Caltech intermediate form* [134] and an example of a parameterized two-turn octagonal spiral inductor is given. Drawing 45-degree layout structures is an ability scarcely found in algorithmic approaches (two exceptions being the routers of [135] and [136]). However, this ability can play an important role for certain applications, e.g., for the creation of transmission lines with 45-degree bends in high frequency designs (having the advantage that their electrical behavior, in contrast to that of right-angled bends, does not have to be especially simulated).

As already mentioned, a perennial issue in the context of procedural generators is how to conveniently migrate a procedural generator to a different semiconductor technology. This desire feeds the incentive to implement such generators as technology-independent as possible. For that purpose, the work of [137] proposes a generator programming interface based on a technology abstraction layer where technology-specific parts are separated from the (topo-)logical parts of a generator implementation. Another idea is presented in [138], where an abstract placement graph is extracted from the generator code at runtime and then utilized in a post-processing step to resolve technology-related layout issues according to a library of critical layout structures, without the need to change any line of code. Still, technology-dependence is often cited as a soft spot of procedural generators.

A preliminary work for this thesis has been done in [139]: Figure 3.17 shows a “Pad-over-Active”-aware PCell, where a bondpad area can be manually defined by mouse. The area is passed to the PCell as a pointlist and then processed in order to adjust the generated layout such that only the part of the top-level routing that belongs to the bondpad’s net runs below the bondpad area (as can be seen in the image, the routing finger in the middle –which belongs to another net– is shortened thereby). A generalization of this concept is known as *fluid shapes* and facilitates an arbitrary shaping of complex layout structures such as guardrings. An example of a fluid guardring PCell is displayed in [140]. The basic idea behind this approach (i.e., encapsulating complex data in a single parameter) is not necessarily limited to such structures but can be extended to consider a PCell’s entire neighborhood. For example, in [141] a set of layout devices is turned into a parameter value and passed to a PCell which re-creates the devices internally, measures their pin positions and generates an appropriate routing.

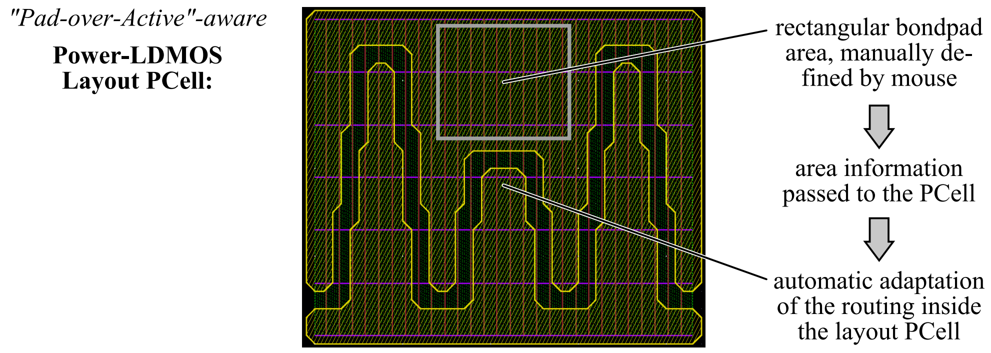

"Pad-over-Active"-aware
Power-LDMOS
Layout PCell:
rectangular bondpad area, manually defined by mouse
area information passed to the PCell
automatic adaptation of the routing inside the layout PCell

Figure 3.17: “Pad-over-Active”-aware PCell which processes geometric input (adapted from [139]).

That idea is also utilized and even further enhanced in this thesis to break up the seclusion of a procedural generator which results from being an enclosed library component and isolates every generator instance from its layout context. For a primitive device, this isolation is welcome, leaving the device’s predefined set of input parameters as a dedicated interface for controlling the device generator. However, this interface represents a real obstacle for an advancement towards higher-level modules, making it difficult to feed more complex data *into* a generator instance and missing the chance to get valuable information *out of* such an instance. This detriment not only bars the user from customizing the instance’s contents in a convenient way, but also precludes the implementation of more sophisticated concepts such as a multilateral communication between different instances.

Since this thesis concentrates on layout design, circuit generators are of rather little interest here. However, they can play a vital role for a seamless inclusion of layout generators into the design flow. This is because most SDL functionalities such as *Generate from Source*, crossprobing, and symbolic

LVS checking rely on a strict correspondence between schematic components and layout components.[4] Hence, to maintain a seamless SDL design flow, a module generator in the layout asks for a corresponding counterpart on the schematic side [142], and this is where circuit generators come into play.

Unfortunately, publications on circuit generators are rare. The work of [143] uses SKILL to implement a parameterized multi-conductor transmission line with a variable number of pins, as well as a buffer component of changeable buffer type. The authors point at the combinatorial amount of component variants that would have to be provided if no parametrization was utilized – the involved design effort, maintenance overhead, and hampered usability is obvious. In [144], an example of a SKILL-based inverter is presented, employing parametrization to create multi-fingered subtransistors via discrete devices, thus achieving more accurate simulation results through a more realistic consideration of parasitics.

For SDL design flow reasons, [145] provides a Current Mirror circuit PCell to accompany the layout PCell shown in [132]. Of course, the circuit PCell covers several topologies with the same extent of variability as its layout counterpart. Inventive means to *edit-in-place* and *flatten* instances of the circuit PCell are also demonstrated: the former enables an editing mode where subtransistors of an instance become temporarily accessible to enact modifications, the latter dissolves the instance by replacing it with its internal circuitry. In this context, it should be noted that a circuit PCell is in fact a pair of two PCells: a *schematic PCell* that generates the topological structure of the circuit, and a *symbol PCell* that describes its interface (pins, terminals) and visual appearance.

The topic of schematic and symbol PCells will again be picked up in Chapter 9 –for the incorporation of the developed automation methodology into the design flow– while the rest of this thesis remains focused on layout design.

3.1.2.6 Procedural Generators – Conclusion

As a résumé, the trait of targeting a specific circuit class gives a procedural generator the natural ability to take intricate design constraints into account in an implicit way, eluding the necessity to formalize them. Hence, a parameterized cell can –opposite to an optimization algorithm– be regarded as a “specialist”: it is able to produce layout results in full-custom quality, but despite the degrees of freedom it may offer, it is limited to a particular family of closely related circuit variants. In summary, the assets of procedural generators are:

- Re-use of expert knowledge on a higher level of abstraction.
- Variability is directly realized through the generator parameters.
- No demand for constraining and thus no risk of overconstraining.
- Ability to perform both placement and routing simultaneously.
- No need for problem modeling, design-methodical restrictions, or heuristics.
- Deterministic (even predictable) behavior and fast runtime.

As has been illustrated in this Section 3.1.2, procedural generators are suitable for automating simple modules. However, they already drop the ball when proceeding to the next higher level of advanced modules, obstructed by a couple of hindrances:

- Necessity for preparatory generator specification work, with the difficulty to determine the best trade-off between parametrical customizability and implementation effort.
- An efficient generator implementation relies on dedicated tools covering several concepts (e.g., graphical programming, support for hierarchy, object-orientation, geometrical operators).
- Additional overhead is required to make a generator as technology-independent as possible.
- Implicitly considering potential constraints that are not yet known during generator development relies on explicitly preconceiving and implementing the generator’s entire variability in advance.
- The variability of an advanced module is effectively continuous and infinite, rooted in the close interdependencies between all possible device sizes, arrangements, and layout variants of its submodules.

[4]Even though auxiliary constructs such as a one-to-many or many-to-one mapping between schematic and layout exist, hierarchical breaks of that kind are a constant source of trouble.

- Conventional parameter interfaces are a bottleneck for the transfer of more complex information (e.g., geometrical data) into, out of, and between generator instances.
- A seamless inclusion of layout generators into the SDL flow requires the implementation (and consequent usage) of corresponding circuit generators.

By now it should have been credibly shown how hard it is to get ahead with analog layout automation using either optimization algorithms *or* procedural generators. So, what about other approaches which incorporate both algorithmic *and* procedural aspects (Section 3.1.3)?

3.1.3 Other Approaches

This section is supposed to point out that all existing layout automation approaches basically belong either to the category of optimization-based approaches (Section 3.1.1) or to the category of generator-based approaches (Section 3.1.2). Although some layout automation tools look like hybrid approaches, on closer examination they always reveal a distinct emphasis on one of the two automation strategies and the respective manner of constraint consideration, which is either explicit or implicit.

As an example, one may contemplate the ModGen tool [146], that is meant to assist designers in the placement and routing of device groups which constitute simple modules. Regarding the placement, ModGen helps the designer by aligning all devices in a regular array, according to an interdigitation pattern that the designer provides via a dedicated pattern editor. As helpful as that may be, it is merely an editing aid that does not implicitly consider any placement constraints. The subsequent routing is performed with a traditional algorithmic auto-router. If routing constraints are to be considered, they need to be formally entered into an associated constraint manager GUI. Thus, the ModGen tool is plainly geared towards a purely explicit consideration of constraints.

Surely, most optimization-based placement approaches utilize procedural generators for components on the lowest design levels, and some of those approaches such as [147] even acknowledge that certain low-level duties like interdigitation should be handled by these generators rather than through an algorithmic consideration of explicit placement constraints. However, this understanding remains unsatisfied if the generators fall short of covering the required parametrical variability or if the optimization engine fails to exploit that variability to the full extent. And indeed, contemporary optimization-based approaches can only be observed to employ fairly trivial generators, so the workload –and more importantly: the constraint handling– clearly lies on the shoulders of the algorithm instead of being equitably apportioned between the algorithm and the generators.

In addition to such constraint-driven automation approaches, [148] identifies two other layout synthesis methodologies: (1) layout migration with retargeting, and (2) layout synthesis with knowledge mining.

The former methodology produces a new layout based on an existing legacy layout from a different semiconductor technology instead of creating the whole layout from scratch. This is done by extracting a symbolic template from the legacy layout to deduce placement and routing constraints from that handcrafted layout solution. Then, the layout is reconstructed in the new technology and optimized via compaction techniques that try to maintain the deduced constraints. Examples are given in [149], [150], [151], and [152].

Because this methodology is limited to a strict preservation of the circuit topology (except [153], which allows for certain variations), layout synthesis with knowledge mining investigates an entire repository of legacy circuits and layouts. By analyzing these legacy designs, a knowledge database is automatically filled with constraint information about subcircuits of the legacy designs. During synthesis, this information can then be applied to corresponding subcircuits of the new design, as done in [154] and [155] for example.

Although these two *template-based* methodologies aim at incorporating existent design expertise into the algorithmic layout creation, they do not facilitate the kind of re-use achieved with procedural generators (see Section 3.1.2.2). Instead, the informal expert knowledge contained in a given layout is extracted and translated into a formalized representation of constraints. This formalized expert knowledge is then passed to an algorithmic optimization engine to be processed in a conventional, purely explicit fashion.

Most algorithmic works in analog layout automation pursue the canonical way of optimizing a single candidate solution (see Figure 3.1), but there are also approaches that operate on an entire pool of candidate solutions. Such approaches constitute the branch of *population-based* optimization and one prominent technique of that branch is evolutionary computation [156], which has already appeared in Section 3.1.1.

Evolutionary computation is a subfield of *artificial intelligence* that puts forth algorithms inspired by Darwinian principles of evolution. As an example, the work of [157] employs a genetic algorithm for analog module placement, while the placement approach GASA [158] combines a genetic algorithm with Simulated Annealing, which was utilized in the work of [88]. A couple of other approaches that make use of evolutionary computation have also been referenced already ([38], [74], [84]).

Another popular technique among population-based optimization approaches is represented by methods of *swarm intelligence* [159]. Two widely recognized types of such so-called swarm algorithms are known as Ant Colony Optimization [160] and Particle Swarm Optimization [161], both of which have been applied in the floorplanning approach of [40] (see Table 3.1). In [162], a modified Particle Swarm Optimization algorithm is used for the area optimization of a two-stage amplifier.

Apart from these population-based approaches, only few other concepts from the field of artificial intelligence have also been considered in analog layout automation. As an example, [163] investigates the use of artificial neural networks for module placement while [164] presents a floorplanning approach based on a neural learning algorithm. Fuzzy logic, despite its application to many different problems, has been of almost no interest for layout design so far, the placement approach of [165] being one exception. Compared to layout automation, a much richer spectrum of optimization techniques has been employed for circuit sizing [166], including Ant Colony Optimization [167] and Particle Swarm Optimization [168].

3.1.4 Combining Optimization and Generation – Conclusions for the New Approach

All in all, it is fair to say that every layout automation approach existing so far can be classified as basically being either an optimization-based approach (where design constraints are taken into consideration explicitly), or a generator-based approach (where design constraints are taken into consideration implicitly). Distinctive traits are the versatility of the automatism in the former approach, and the quality of the resulting layouts in the latter. To get hold of both, the following suggestions for the undertaking of this thesis can be deduced from the findings of Section 2.5 and Section 3.1:

- Merge optimization and generation into a symbiosis where both respond to each other.
- Conquer the design problem by delegating and distributing design tasks in a balanced way.
- Take care of the high-level constraints and all low-level layout details at the same time.
- Exploit all degrees of freedom instead of reducing the solution space via standardization.
- Diminish the constraining effort as well as the preliminary generator development effort.
- Share the workload while performing the different design tasks in a highly concurrent fashion.
- Refrain from randomization but make the automation methodology completely deterministic.

However, as should be obvious by now, optimization algorithms and procedural generators represent such different automation strategies that the above suggestions for combining them involve a couple of problems (P) raising questions (Q) for which the following answers (A) are disclosed right away:

P: An algorithm reigns like a “dictator” whereas a generator works like a “slave”.
Q: How can the two parties join forces in a more “democratic” form of government?
A: Engage algorithmic optimization and generator-based layout creation in a bilateral relationship.

P: An algorithm finds its layout solution at runtime while that of a generator is preconveived in advance.
Q: How can the temporal and conceptual gap between the two forms of solution finding be bridged?
A: Let a generator’s set of preconceived layout solutions flow into the algorithmic solution finding.

P: An algorithm surveys the overall problem whereas a generator concentrates on a specific purpose.

Q: How can these two strategies, with their disparate perspectives, be made to meet in the middle?
A: Make the algorithm split and spread the decision-making across several individual generators.

P: An algorithm asks for diminishing the solution space, a generator exploits its degrees of freedom.
Q: How can the algorithm be enabled to handle the immense variability introduced by the generator?
A: Reduce the algorithm's authority and scope of responsibility by giving it only indirect powers.

P: An algorithm performs unforeseen actions but a generator can only cope with expected situations.
Q: How can the generator be enabled to manage influences that have not been anticipated in advance?
A: Equip the generators with the capability of reacting to changes of their layout context.

P: Algorithmic optimization proceeds repetitively, executing a generator is a straight and sealed process.
Q: How can the features of a generator be fully utilized during the flow of successive solution finding?
A: Let the generators interact with each other in repetitive cycles throughout the optimization.

P: An algorithm for NP-hard problems often involves randomization while a generator acts predictably.
Q: How can a combination of algorithmic optimization and layout generation be expected to behave?
A: The behavior of such an approach is supposed to be deterministic but not entirely predictable.

To fathom the prospects of combining optimization-based automation and generator-based automation, Section 3.2 looks at these two automation strategies from a more academic perspective.

3.2 An Abstract View on Existing Approaches: Two Fundamentally Different Automation Paradigms

The concrete view of Section 3.1 on the practical application of optimization-based approaches (Section 3.1.1) and generator-based approaches (Section 3.1.2) shows that they are quite reciprocal regarding the versatility of the automatism and the quality of its resulting layouts. Looking at these approaches from a more abstract perspective, it becomes apparent that optimization algorithms and procedural generators are not only different from each other in several aspects, but that they are downright complementary. This leads to the definition of two distinct *automation paradigms* (in Section 3.2.1 and Section 3.2.2) and to a vision of combining these two paradigms (in Section 3.2.3).

3.2.1 Optimization Algorithms: Top-down Automation

The following items reprise some essential characteristics of optimization algorithms, as already discussed in Section 3.1.1.1:

(1) An optimization algorithm cycles through an exploration-evaluation loop to repeatedly optimize a candidate layout.
(2) In this manner, the algorithm is meant to self-intelligently find the solution for a given design problem (thus re-inventing the solution anew every time).
(3) For that purpose, the algorithm works on an abstract mathematical model of the design problem.
(4) The problem modeling allows the algorithm to take certain design constraints into consideration explicitly.
(5) All design requirements must be expressed in a formal way (i.e., all relevant expert knowledge must be completely formalized) such that the algorithm can find a feasible solution at runtime.

Putting it straight, optimization algorithms rely on a complete formalization of the overall solution finding. As such, one can say that this strategy typifies a distinct automation paradigm, roughly outlined by the points above. That paradigm will be denoted as *top-down automation* in this thesis. Thus, the term top-down is not used in the usual sense of relating to a design hierarchy, but to denote the nature of automation, which must make sure to model the entire problem as a whole.

Optimization algorithms follow a distinct automation paradigm here denoted as top-down automation.

While certain aspects of the design problem can be modeled easily, the consideration of intricate layout details incurs a severe complication of the overall formalization, which commonly leads to qualitatively insufficient automation results. This loss of layout quality accompanies the qualitative penalties that are induced by resorting to design-methodical standards and by employing heuristics. With respect to the size of a layout problem, the quality losses are quite characteristic when compared against the best possible degree of layout quality (i.e., manual expert design). This characteristic quality curve is depicted in Figure 3.18 (a).

Being based on an abstraction of the design problem, top-down automation involves a significant initial loss of layout quality. On the other hand, this way of reducing the degrees of freedom enables such automatisms to handle layout problems of immense quantitative complexity. With increasing problem size, the quality decline continues further, but becomes more and more marginal. The overall *quality gap* can be tolerated in the digital domain, but dissatisfies the quality requirements of analog design.

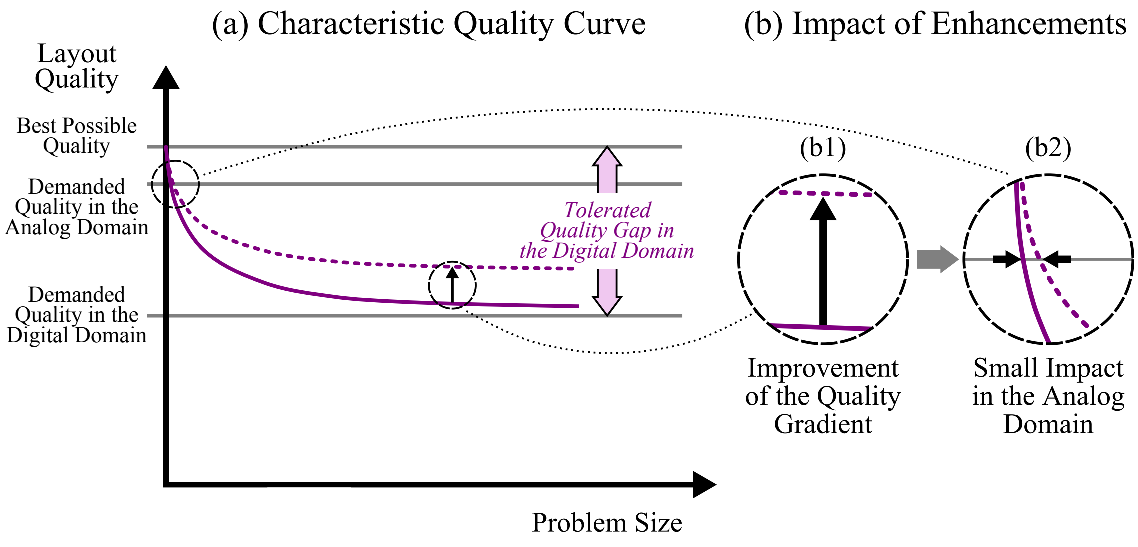

(a) Characteristic Quality Curve
(b) Impact of Enhancements
Layout Quality
Best Possible Quality
Demanded Quality in the Analog Domain
Demanded Quality in the Digital Domain
Tolerated Quality Gap in the Digital Domain
Problem Size
(b1)
(b2)
Improvement of the Quality Gradient
Small Impact in the Analog Domain

Figure 3.18: Quality characteristics of top-down automation.

Figure 3.18 (b) illustrates another aspect of this automation paradigm: a slight qualitative improvement of a top-down automatism practically has no effect on its usefulness in the analog domain. Even though such an improvement may noticeably lift the quality curve (b1), the initial quality loss prevents the automatism from becoming suitable for analog designs (b2). That finding is confirmed by EDA history: although optimization-based layout algorithms have been successively improved over the past three decades, no breakthrough can be seen in the industry.

This observation puts another spin on the use of the term *top-down*. A huge and very real problem for the industrial adoption of optimization-based automation approaches is their affinity to completely replace existing design flows in a *revolutionary* way. This “all or nothing at all” habit of imposing an optimization-based design methodology top-down does not find acceptance as long as that methodology fails to cover the design problem in its entirety and to achieve the demanded level of layout quality. The situation is quite different with procedural generators, as follows in Section 3.2.2.

3.2.2 Procedural Generators: Bottom-up Automation

Recapitulating Section 3.1.2.2, the subsequent enumeration (corresponding to the items listed in Section 3.2.1) sums up the trademarks of layout automation with procedural generators:

(1) A procedural generator follows a *straight sequence* of commands to draw a customizable layout. Thus, the generator does not cycle through a *repetitive loop* that optimizes a candidate layout.

(2) The generator only (re-)produces a layout *result*. The inventive task of thinking up an optimal layout *solution* is completely left to the human expert who implements the generator (preferably *re-using* existing design knowledge instead of *re-inventing* the solution anew).
(3) With the introduction of parameters, a procedural generator enables a *generalization* of the preconceived solution (which *increases* the degrees of freedom). This contrasts an *abstraction* of the problem (which *lessens* the degrees of freedom).
(4) On that basis, constraints can be taken into consideration *implicitly* (i.e., in a *nonformalized* way). This is done without the need to express these constraints *explicitly* (i.e., to provide them in a *formalized* representation).
(5) The entire set of solutions that is to be covered by the generator (i.e., the mathematical image of the generator, as introduced in Section 3.1.2.1) must be *preconceived in advance* by the generator developer. The generator itself merely mimics the expert's laborious drawing work, but does not help the design problem to be *solved at runtime*.

Regarding the nature of automation, a comparison of the above items with those in Section 3.2.1 spotlights that the characteristics of procedural generators are distinctly different from the characteristics of optimization algorithms. Hence, it is consequent to say that procedural generators follow a distinctly different automation paradigm. This paradigm will be denoted as *bottom-up automation* in this thesis. The term *bottom-up* is quite appropriate considering its well-known usage to denote processes that do not have a particular intent. For example, biological evolution is said to proceed bottom-up since life has no ultimate goal [169]. The same is true from the viewpoint of a procedural generator: in contrast to an optimization algorithm, a generator is not eager to solve a design problem – in fact, it does not even "know" about the problem itself.

Procedural generators follow a distinct automation paradigm here denoted as bottom-up automation.

The particular strength of bottom-up automation is the ability to satisfy complex, low-level design requirements implicitly. In practice, that ability is limited by the question to what extent a feasible layout solution can be predetermined and generalized in advance. Thus, compared to top-down automation, bottom-up automatisms have an inverse characteristic curve in terms of layout quality, as illustrated in Figure 3.19 (a).

For small problem sizes, the achievable layout quality is close to full-custom. However, larger problems lead to an increasing drop of layout quality due to the difficulty to anticipate all relevant design requirements and keep up sufficient parametric variability. So, bottom-up automation can ensure the demanded degree of layout quality up to a certain problem size, but still a significant *automation gap* has to be tolerated in the analog domain.

However, Figure 3.19 (b) illustrates the evident advantage that even a small improvement of the quality curve (b1) pushes the limits of automatable problem sizes to a perceptibly greater extent in the analog domain (b2). This particular trait is also disparate from the characteristics of top-down automation and underlines why industrial companies prefer to drive the development of bottom-up automatisms, since every contribution reveals an immediate gain in design productivity.

This aspect further reinforces the usage of the term *bottom-up*, because automatisms such as procedural generators can be introduced into present design flows in an *evolutionary* –instead of a revolutionary– fashion. In that way, existing design flows can be incrementally improved step by step, and are thus literally augmented from the bottom up. Beyond this obvious benefit for layout efficiency, such an advancement could –in the long run– turn out to be even more seminal than is apparent at first sight. This expectation is rooted in the belief that bottom-up automation may pave the way to a successful utilization of top-down approaches. In Section 3.2.3, advancing the degree of design automation bottom-up is sketched out as one of three major milestones on an envisioned path of combining the two paradigms.

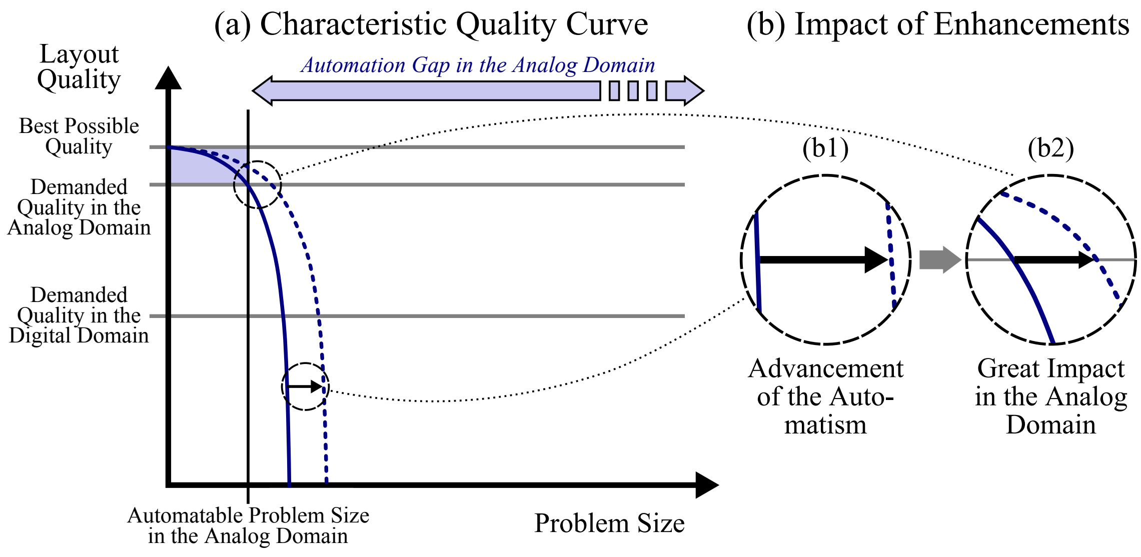


Figure 3.19: Quality characteristics of bottom-up automation.

3.2.3 Envisioning a New Automation Philosophy: Bottom-up Meets Top-down

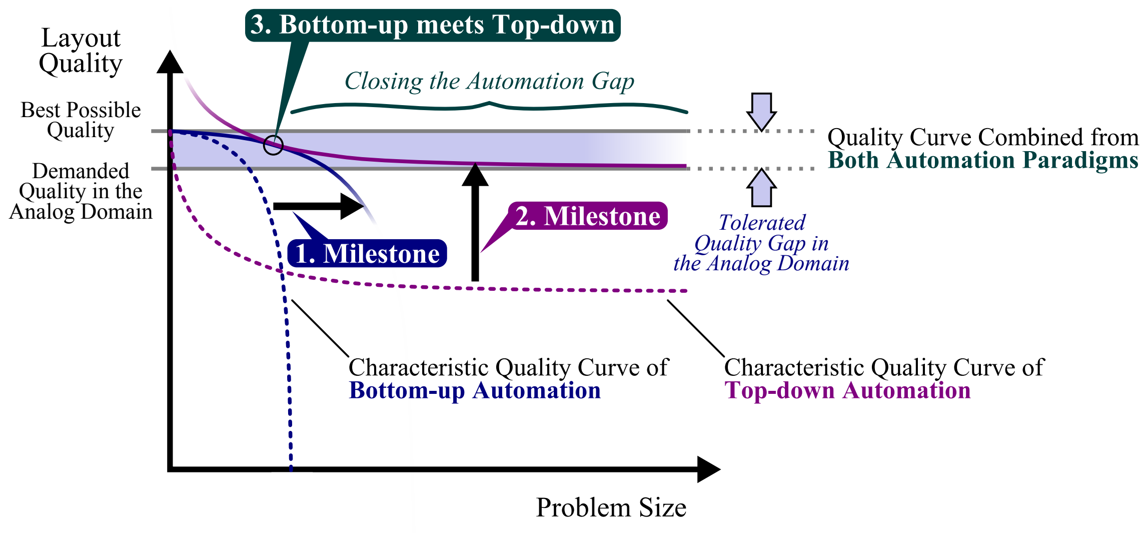
Summing up the discussion of Section 3.2.1 and Section 3.2.2, top-down automation and bottom-up automation exhibit quite converse automation characteristics. To put it more positively, the respective strengths and weaknesses of the two automation paradigms do not merely oppose each other but may rather complement one another. So, although EDA research considers procedural generators to be less worthwhile than optimization algorithms, the abstract view of Figure 3.18 and Figure 3.19 on the two automation strategies justifies that these should be granted equal scientific relevance for analog layout automation since both underlying paradigms do have their particular assets.

On that account, outreaching the largely concurrent work on optimization algorithms and procedural generators in academia and industry, this section pronounces a new mission for EDA: to explore the full potential of both paradigms and drive their convergence towards a novel *bottom-up meets top-down* design flow [170]. Presumably, a well-balanced combination of the two automation paradigms has much more potential for analog EDA than bottom-up or top-down approaches alone.

This assumption is backed by the characteristic curve in Figure 3.20, which is envisioned as the ultimate aim of a prospective EDA roadmap following five individual steps: (1a) provide tools for capturing expert design knowledge implicitly, (1b) transform the implicit expert knowledge into new bottom-up automatisms, (1c) integrate these automatisms into present design flows –compendiously, these three steps are meant to (1) promote bottom-up automation– in order to (2) tailor top-down approaches to the new bottom-up automatisms, and ultimately (3) combine the two paradigms. These five steps are now sketched out in greater detail.

Step 1a: Human expertise is essential for analog layout automation, but unfortunately layout designers are not accustomed to explaining their solution procedures in an explicitly formalized way. Therefore, step 1a is to develop innovative tools for capturing expert design knowledge implicitly. These tools should allow design experts to describe solution procedures for specific design tasks in an intuitive fashion which corresponds to the designers' mentality. The request to build effective bridges, that match the designers' way of thinking as closely as possible, is a great challenge and unfurls a fruitful, yet largely unexplored field for EDA.

Step 1b: Tools resulting from step 1a allow layout experts to transform their invaluable design knowledge into dedicated new automatisms. Hence, the purpose of step 1b is a further advancement of bottom-up automation in order to cover the design levels where top-down approaches typically fail to achieve the demanded degree of layout quality. This task raises the question, up to what levels of functionality bottom-up automation will have to proceed. As already mentioned, bottom-up



3. Bottom-up meets Top-down
Closing the Automation Gap
Layout Quality
Best Possible Quality
Demanded Quality in the Analog Domain
Quality Curve Combined from Both Automation Paradigms
Tolerated Quality Gap in the Analog Domain
1. Milestone
2. Milestone
Characteristic Quality Curve of Bottom-up Automation
Characteristic Quality Curve of Top-down Automation
Problem Size

Figure 3.20: Path towards the envisioned bottom-up meets top-down design flow.

automatisms are particularly suited for the layout generation of simple modules (see Table 2.1) that perform prime analog functions and display highly regular layout patterns.

Step 1c: Opposing the unfavorable affinity of new design methodologies to completely replace existing design flows, step 1c is to seamlessly introduce the newly developed automatisms of step 1b into present design flows in an evolutionary way. In [124], this suggestion of raising the automation degree bottom-up rather than fiddling about algorithmic synthesis ad nauseam has even been proclaimed as a paradigm shift. The development and integration of the new bottom-up automatisms immediately results in an improved characteristic curve as visualized in Figure 3.20 (1. Milestone). This curve marks the eventual impact of steps 1a, 1b, and 1c, thus representing the first major milestone for meeting bottom-up with top-down.

Step 2: The second major milestone –and goal of step 2– is to adapt existing top-down approaches to the capabilities of bottom-up automation, as seen in Figure 3.20 (2. Milestone). Bottom-up automatisms allow top-down approaches to delegate crucial low-level design tasks downwards, and therefore to concentrate solely on higher-level requirements, which simplifies the overall layout problem significantly. Thus, the continuation of research work on an explicit consideration of high-level constraints remains a vital task for future EDA, but the practical benefit of constraint-aware optimization will be much greater on these levels than on the lower levels, where constraint handling has become second nature to the designers.

Step 3: In step 3, the advanced bottom-up automatisms of the first milestone are to be joined with the adapted top-down approaches of the second milestone. This will result in the ultimate *bottom-up meets top-down* design flow, which represents the third and final milestone. As illustrated in Figure 3.20 (3. Bottom-up meets Top-down), this accomplishment of combining the strengths of both paradigms is expected to be a major breakthrough for EDA. According to that characteristic curve, a balanced coalescence of bottom-up and top-down automation should have the potential to close the automation gap in analog layout design *and* to maintain the level of quality that is uncompromisingly demanded in the analog domain.

Tapping the full potential of bottom-up automation, in order to cover the crucial design levels of analog basic circuits, can simplify the overall optimization problem significantly. This move mirrors a similar step that has been taken in the digital domain: the introduction of standard cells, which allow synthesis tools to operate on gate level rather than on transistor level. Equivalently, bottom-up automatisms can help to elevate the analog design flow to the module level. The pivotal difference is that the modules are not standardized but parameterized, in order to cover all necessary degrees of freedom. Yet, an open problem is how top-down approaches can be enabled to cope with that immense variability. From the outset, bottom-up and top-down automation are not offhand compatible with each other, which is why so far no existing EDA achievement has accomplished to provide a truly balanced combination of the two paradigms.

One additional remark should be made in this matter, since a look at Figure 3.13 suggests that a procedural generator already incarnates both paradigms: on the one hand, the generator contains nonformalized expert knowledge, and on the other hand, the generator is controlled by input parameters which can be regarded as formalized expert knowledge. However, a procedural generator still represents a pure bottom-up automatism because its set of layout solutions is entirely preconceived, providing parameters to –figuratively speaking– let the designer “choose” one specific instance from that set.[5] In other words, each permutation of parameter values maps to one particular layout variant in a predictable (not just deterministic) way. In contrast, the trademark of a potential *bottom-up meets top-down* design flow would be to produce layout results that can only be predicted in parts but not in their entirety.

To put it in a nutshell, the envisioned design flow implies to combine two fundamentally different automation paradigms, as this is supposed to be the key for a successful automation of analog layout design. The scientific challenge in pursuit of that philosophy is to answer the question, if and how the two different paradigms can be brought together. To be capable of determining the practical virtue of such an endeavor, Chapter 4 deals with the question, by what assessment criteria an analog layout methodology is to be judged.

[5]Apart from this, it can be said that the expert knowledge reflected by the setting of the parameter values is most of all circuit knowledge (primarily concerning the circuit topology and the device dimensions) rather than layout knowledge. Hence, these parameter values predominantly specify *what* is wanted – but, *how* to realize this in the layout is implicitly specified by the implementation of the procedural generator.

Chapter 4

Assessment Criteria for a Layout Methodology in the Analog Domain

Beware of the half-truth. You may have gotten hold of the wrong half.
Unknown Author

It is the pronounced purpose of the work presented in this thesis to be not only scientifically valuable, but practically useful as well. Therefore, this chapter discusses what assessment criteria an analog layout methodology must go by to find acceptance in practice. The term *layout methodology* is deliberately favored over speaking of an *automation approach* here because an assessment metric should also be applicable to a largely manual design flow, for example.[1] As will be shown in Section 4.1, the academic criteria that are traditionally consulted in literature do not tell the whole truth because they usually do not consider industrial practicability from a holistic perspective. For that reason, Section 4.2 works out a more expedient assessment chart which leads to the conclusion that the evaluation of an analog layout methodology is –equivalent to the qualitative complexity in the analog domain– not just a matter of *More Moore* but a rather complicated *More than Moore* issue, as will be summarized in Section 4.3.

4.1 Traditional Assessment Criteria

With the intention of facilitating objective comparisons, automation approaches are commonly evaluated through *benchmark circuits* (see for example [171]). While such an evaluation is quite substantive in the digital domain, for analog design that practice is not differentiated enough so far. Concerning floorplanning and placement algorithms, a set of benchmark circuits have been archived at the Microelectronics Center of North Carolina (MCNC) [172].[2] As listed in Table 4.1, the five MCNC benchmarks that are most often referred to for analog layout automation contain between 9 and 49 cells denoted as "building blocks".

As this denomination already indicates, these cells have fixed dimensions and therefore they do not adequately represent typical analog devices or modules that feature certain degrees of freedom (such as the number of fingers in a MOS transistor, as shown in Figure 2.1 on page 20). Hence, it is fair to say that these benchmarks are not even suited for the evaluation of analog *floorplanning* approaches (unless fixed dimensions are assumed for every floorplan block), but much less for analog *placement*. Although initiatives to define new analog benchmark suites have recently been contemplated [173], no progress can be seen so far.

[1]Unfortunately, some criteria only apply if some way of automation is included. That is why the subsequent paragraphs sometimes speak of automation approaches (and their automatisms). Otherwise, the more general notion of a layout methodology (and its mechanisms) is used.

[2]Meanwhile, these benchmark circuits have moved to the Collaborative Benchmarking Laboratory at North Carolina State University.

Table 4.1: Overview of the five most referenced MCNC benchmark circuits.

Circuit	Cells	Symm. Cells	Nets	I/O
apte	9	8	97	73
xerox	10	N/A	203	2
hp	11	8	83	45
ami33	33	6	123	42
ami49	49	4	408	22

The assessment criteria which are traditionally referred to in literature can be divided into criteria that obviously stem from the digital domain (Section 4.1.1) and analog-oriented criteria that are concerned with the question which constraints can be taken into consideration (Section 4.1.2).

4.1.1 Criteria Originating from the Digital Domain

A look at the many existing publications on optimization-based layout automation approaches reveals that these approaches are commonly compared according to the following three primary criteria:

- runtime,
- total area,
- total wirelength.

It goes without saying that these criteria have been simply taken over from the digital domain, but it is quite questionable how meaningful they are for analog design. Of course, in the digital domain these criteria matter enormously because their impact scales with the number of components on the chip. Thus, even the slightest reduction in runtime, area occupation, or wirelength that may be measured for a given test circuit, can make a truly noticeable difference when applied to industrial ICs with millions and billions of logic gates. And, regarding the ongoing progress towards upcoming technology nodes with ever-rising integration density, these criteria will remain increasingly relevant in the future.

In the analog domain however, one should honestly admit that such digital-inspired criteria are only secondary. For instance, present-day papers on the development of leading-edge automation approaches and their application to more or less representative analog circuits report runtime results in the range of a few minutes or even seconds. But despite the incessant pursuit of further performance improvements, such ambitions are beside the point when contemplating the fact, that an analog layout expert in an industrial design environment readily spends days and weeks to handcraft a full-custom layout block [174]. As already explained in Section 2.3.2, the downside for design productivity is tolerated as the lesser of two evils in order to attain the demanded degree of layout quality.

The situation is similar with respect to area and wirelength. Surely, these are two very important optimization goals – however, this is not just for predominantly quantitative reasons (as in the digital domain) but rather for qualitative reasons, namely *matching*. Hence, although literature on EDA advancements keeps on reporting increasingly better area and wirelength minimization in novel placement and routing approaches, the true merit of these achievements only becomes apparent when both placement and routing are performed together. Otherwise, layout solutions like the one depicted in Figure 3.12 (b) might be produced. And such layout solutions are not merely rejected in practice due to the large area and wirelength per se, but because of the poor matching that this detriment entails.

The bottom line of these ruminations is that such quantitative More Moore assessment criteria as traditionally drawn on in literature are not adequate enough to account for the true aptitude of an analog automation approach (or layout methodology in general). In essence, the obstacle for a solid assessment is equivalent to the "problem modeling dilemma" faced by optimization algorithms (as discussed at the end of Section 2.5.1): either the assessment metric is too elaborate to allow for a formal treatment, or it only concentrates on very few selected aspects and thus oversimplifies the matter.

4.1.2 Criteria Regarding the Supported Constraint Types

What EDA research got definitely right, is that quantitative comparisons alone do not have sufficient explanatory power in the analog domain, but that qualitative attributes must also be considered. In literature, this comprehension has manifested itself in the question which types of constraints an automation approach is able to handle. Unfortunately, such a valuation is still insufficient to adequately cover the complexity and diversity of the many design restrictions and design objectives in the analog domain. Only *one* missing constraint can already impair the functioning of the circuit even if all other constraints are satisfied. And furthermore, as described in Chapter 3, many design necessities are not even addressed by today's formal constraint representations.

The decision to translate abstract design requirements into more precise geometrical relations is comprehensible since it not just allows them to be conveniently targeted via optimization-based automation but also enables a formal verification of constraints. Yet, this formal ascertainability again involves a dilemmatic deviation from reality: comparing automation approaches based on the types of constraints they support does not tell whether the resulting layouts will really satisfy all functional requirements. In that regard, the absolute truth only lies in the final silicon. Theoretically, processing the layout into a measurable IC would be one requisite for a truly cogent benchmarking, but this is not feasible in practice. Instead, one may take a look at productive design flows, where layouts are signed off according to the expert's experience-based opinion that the chip will indeed work as expected.

So, equivalent to the *development* of an analog automation approach, performing a credible, objective, and thorough *comparison* with other layout methodologies in both quantitative and qualitative regard is a challenge of its own. While that subject might even fill an entire dissertation by itself, it is not the central topic of this thesis. To provide a pragmatic assessment approach without boiling the problem down to a few formally graspable criteria, Section 4.2 makes an attempt at giving the full picture of all aspects that can play a role in assessing a layout methodology for practical application. This compilation may serve as the basis for an informal but –more importantly– holistic assessment.

4.2 Relevant Assessment Criteria

Basically, a layout methodology's practical value for an industrial customer –in relation to the *status quo*, i.e., the currently established layout methodology– can be judged by two factors: (1) the methodology's impact on the customer's *design productivity* and (2) the degree of *layout quality* that it is able to achieve. Both factors in turn depend on multiple subcriteria which will now be discussed in Section 4.2.1 and Section 4.2.2 respectively. Therein, the industrial customer is presumed to be a semiconductor manufacturer with an in-house circuit and layout design department but without a proprietary business division for EDA.

4.2.1 Design Productivity

Most publications on analog layout automation approaches emphasize the assets of these approaches, forgetting to mention that their utilization is not really gratis but requires a certain kind of "investment". This is to say that the impact on design productivity is not only determined by the potential benefit but also by the involved costs. In the mindmap-like depiction of Figure 4.1, that benefit is referred to as *efficiency gain* while the costs can be comprised of what is denoted here as *effort* (labor costs)[3] and *expense* (financial costs). Section 4.2.1.1 and Section 4.2.1.2 go into greater detail. A formulaic expression for the calculation of layout design effort will be given by equation 8.2 in Section 8.3.4.2.

4.2.1.1 Effort and Expense

If a layout methodology includes some way of automation, the types of costs –i.e., effort or expense– partially depend on the focus of the chosen automation strategy. The following considerations distinguish between optimization-based automation using a commercial synthesis tool and generator-based

[3]Effort in the sense of labor costs is the product of (wo-)manpower and time (e.g., a person-month).

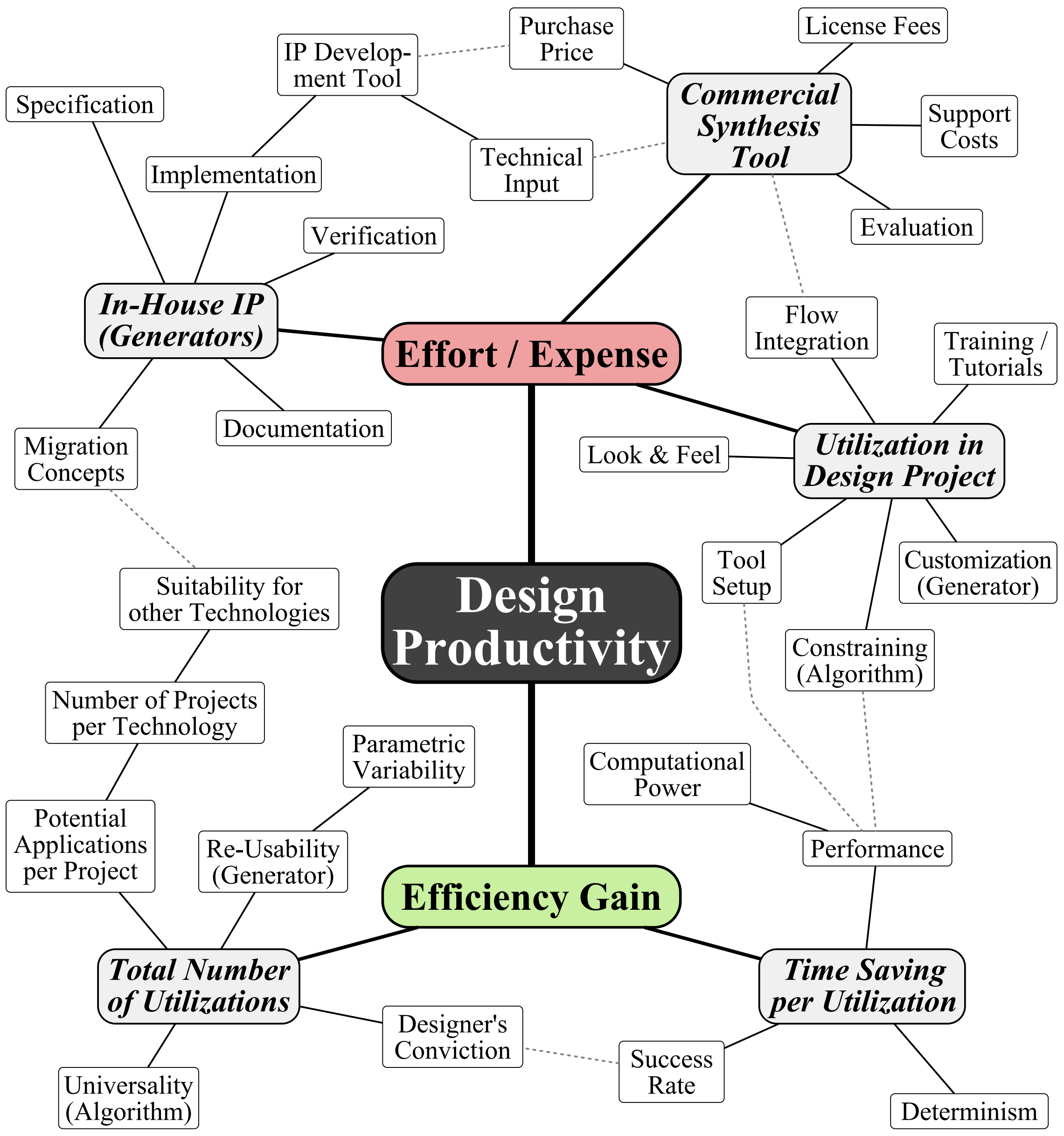


Figure 4.1: Assessment criteria for a layout methodology's impact on design productivity.

automation with in-house IP. Additionally, it should be noted that the utilization of such automatisms in a design project also involves some effort.

Commercial Synthesis Tool

In case the focus is on the use of optimiztion-based automation, the customer is supposed to acquire a dedicated commercial synthesis tool from an EDA vendor. This is because –without a proprietary business division for EDA– a semiconductor company usually cannot bring up the human resources to develop its own synthesis tool, nor to take over and continue a universitarian implementation. Such works have a greater chance of being adopted in industry if sold to an EDA vendor or professionalized through a start-up enterprise able to provide contractual troubleshooting, long-term tool maintenance, and future software improvements.

Buying a commercial tool often claims different kinds of expenses: on the one hand, there are the fixed costs of purchasing the tool, while on the other hand, it may be necessary to budget running costs such as license fees and support charges. Apart from these financial expenses, the decision of purchasing a commercial tool typically asks for a thorough evaluation on the customer's side. This can also involve a substantial amount of company-internal effort for fathoming potential applications, performing testcases and analyzing the results, as well as measuring and extrapolating the benefit for the company.

In-House IP (Generators)

If the automation focus is on the use of procedural generators, these are assumed to be developed by the customer as in-house IP that is specifically tailored to the respective product portfolio. Similar to the evaluation effort above, this first of all requires some specification effort for determining which generators are lucrative to be implemented, for detailing their layout content, and for defining their parametric variability. The parametric variability in turn dictates the effort for implementing a generator; however, that effort can be reduced by employing a dedicated development tool which may either be devised in-house or purchased commercially.

In the latter case, technical input can be given by the customer in order to guide the conception and ongoing enhancements of the development tool. Suchlike input might also be addressed towards further refinements of a synthesis tool, albeit to a lesser degree since algorithmic synthesis is farther away from the customer's expertise than IP development. Acquiring a development tool also calls for financial expenses, but these are expected to be less than those for a synthesis tool because IP development is a job for a few dedicated design experts during predevelopment (not for all layout designers in all design projects), and thus requires less licenses and less support.[4]

Apart from the specification and implementation effort, a procedural generator requires comprehensive verification in terms of DRC and LVS. The verification effort can be significant because these checks do not have to be applied to a single layout but to a multitude of parameterized layout variants to cover as much of the generator's parameter space as possible [175].[5] Additionally, the development of a procedural generator involves documentation effort for commenting its implementation. This can be helpful for later adjustments and for migrating the generator to another semiconductor technology. The effort for the latter can be reduced by implementing a generator as technology-independent as possible, accepting the downside that devising intelligent migration concepts causes certain overhead in advance.

Utilization in Design Project

Regardless of whether the customer's automation focus lies on in-house generators or on a purchased synthesis tool, each individual utilization of the respective automatisms in a design project also involves some effort. Initially, there is the fixed effort for integrating these automatisms into the design flow. In the case of generator-based automation, that issue is less problematic –though not negligible– because procedural generators are a natural element of the design flow anyway (at least when primitive devices are concerned, whereas higher-level module generators may introduce certain SDL problems due to hierarchical inconsistencies). In the case of a synthesis tool, the installation effort (or feasibility in the first place) depends on the tool itself: if the tool is not compatible with the existing design flow, additional interfaces such as OpenAccess (see [98]) might be necessary.

Next, it can be necessary to account for trainings and tutorials in order to teach designers in the handling of the automatisms. In the case of a procedural generator, that handling is denoted as customization and represents the task of providing appropriate parameter values to customize the resulting layout. This task requires an understanding of the parameters but is still relatively low compared to the constraining effort required to apply an algorithmic synthesis tool to a specific layout problem. In addition to this constraint formalization, there may be further setup effort in terms of tool settings, options, preferences, configurations, and problem-specific fine-tuning. That effort in turn depends on the look & feel of the tool which decides how intuitive its utilization is.

[4]The PyCell Studio [123] described in Section 3.1.2.4 can even be downloaded for free.

[5]Due to the growing capabilities of procedural generators, inventing formal verification methods has also become a topic of interest in EDA research [176].

4.2.1.2 Efficiency Gain

The overall efficiency gain of a certain layout automatism (or more generally: a mechanism of the layout methodology) can be calculated from the time by which the layout creation process is accelerated with each utilization (i.e., the saved effort), multiplied by the total number of its utilizations.

Time Saving per Utilization

The time that a layout designer saves with each utilization can be arithmetically obtained by subtracting the runtime of the automatism from the working time that the designer would have required without it. The runtime of the automatism is mainly determined by its performance. In the case of a conventional procedural generator, the runtime is negligible because it seldomly spans more than several seconds. In the case of an optimization algorithm where millions of computational operations need to be performed, the runtime is significantly higher and may noticeably depend on the available computational power. Furthermore, the performance can be heavily influenced by the setup of the tool and by the coherence of the constraining for a particular problem.

Inhowfar it is possible to make compelling statements about the runtime of an optimization algorithm can also be up to the question of whether the algorithm is deterministic or not. Another issue in this context is the success rate of the algorithm: if a probabilistic algorithm has to be applied n times to the very same problem (maybe with slightly different settings) before it produces an adequate solution (or a solution at all), then the total runtime is in fact n times as high. The report of [177] points out that it is important for algorithms to be deterministic such that predictability can be achieved in the design flow. Beside this practical aspect, there is also a psychological effect involved because designers have become increasingly skeptical towards automation approaches whose success rate is subject to “good luck” (as in the case of Monte Carlo algorithms).

Total Number of Utilizations

Such skepticism should not be underestimated since it can severely detract from a designer’s conviction and thus the personal willingness to utilize an automatism. Apart from that, the total number of utilizations is primarily given by the amount of potential applications per design project. In the case of an optimization algorithm, this applicability is correlated with the universality of the algorithm. In the case of a procedural generator, the decisive factor is the generator’s re-usability which in turn depends on the generator’s parametric variability. As an example, the investigations of [178] revealed that the Operational Transconductance Amplifier (OTA) is the most widely used analog circuit class in automotive IC designs. In that regard, it may be profitable to automate the OTA class with a procedural generator, but the generator’s re-usability is only brought to bear to the full extent if the generator covers the enormous topological variance of OTA circuits.

Of course, the total number of utilizations is not necessarily limited to one design project. Instead, an industrial customer has a sincere interest in exploiting the potential efficiency gain throughout all design projects in a semiconductor technology. To reckon the prospective benefit that can be expected in the long run, it is even worthwhile to determine an automatism’s suitability for other technologies. In the case of a procedural generator, that suitability goes hand in hand with the ease of migrating the generator to a different technology, which in turn relies on the effort that has been spent for devising intelligent migration concepts in advance. Thus, the network of assessment criteria with respect to *design productivity* comes full circle in Figure 4.1, but does not yet say anything about *layout quality*. This is a complicated matter in its own right and will be addressed in Section 4.2.2.

4.2.2 Layout Quality

As illustrated in Figure 4.2, assessment criteria for the quality of a layout design go far beyond what is covered by the types of constraints found in literature. Regarding this layout quality, one can discern two facets: one facet is defined by those quality aspects that affect the circuit’s physical *functionality*; the other facet is denoted as *consistency* and deals with the hierarchical organization of the layout. In other words, the first facet (discussed in Section 4.2.2.1) is only concerned with the mere geometrical mask data, while the latter facet (see Section 4.2.2.2) relates to the structural constitution of a layout, which can be quite valuable for the designer despite its irrelevance for circuit functionality.

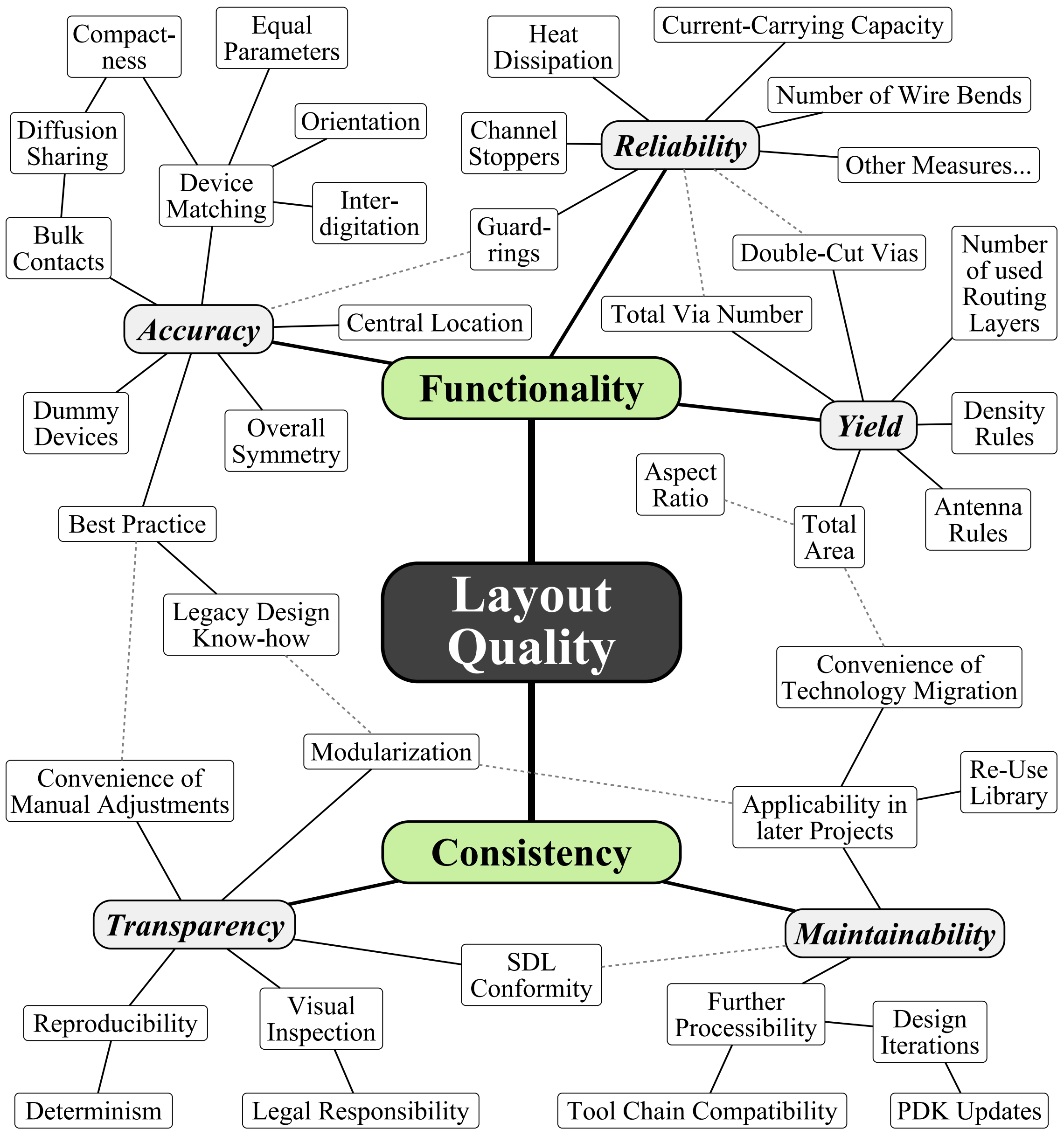


Figure 4.2: Assessment criteria for a layout methodology's aptitude regarding layout quality.

4.2.2.1 Functionality

In the following ruminations (heavily drawing upon [1]), functionality-relevant layout quality aspects are divided into three different categories concerning accuracy, reliability, and yield.

Accuracy

Attaining a properly functioning IC with accurate (and robust) signals under both nominal (and extreme) conditions is of utmost priority. This is where well-known constraints in terms of compactness, equal parameters, orientation and interdigitation are involved to achieve a good device matching. An important issue in this regard is to provide bulk contacts in the vicinity of the devices so the semiconductor substrate is brought to a defined potential. If the devices contribute their own bulk contacts, then it is suitable to abut the devices and let their bulk contacts coincide (as done in Figure 3.5 on page 36), because this so-called diffusion sharing is beneficial for compactness. For very high accuracy demands, it can also be necessary to insert dummy devices such that all of the functional devices see the same neighborhood.

In addition to these matching principles for the devices of certain modules that perform basic analog functions, it is often desired to arrange those modules in an overall symmetric fashion as well. Thereby, particularly critical modules are usually placed in the center of a layout block (or the chip) to avoid edge effects and to minimize piezoelectric disturbances. Apart from such measures, many design decisions for maintaining signal accuracy in the face of nonlinearities and parasitics are those being made implicitly – according to the best practice solutions that originate from years of experience and reflect the invaluable know-how which has already been obtained through a rich fund of silicon-proved legacy layouts.

Reliability

As semiconductor fabrication processes move towards denser technology nodes in the nanometer range, an increasingly delicate problem is posed by reliability failures which occur in the field and shorten the lifetime of a semiconductor product. To prevent or retard such faults that entail an irreversible physical damage to the IC, several techniques may be adhered to during layout design. For example, to prevent hot spots caused by the dissipation loss of a power transistor, its layout can be enlarged to reduce its ohmic resistance and distribute its heat dissipation across a greater area. Another issue is given by long-term effects such as the degradation of routing wires due to electromigration. This can be addressed by increasing a wire's current-carrying capacity via the wire width and by minimizing the number of wire bends since these bends are especially susceptible to electromigration.

Wherever poly or metal wires run above lateral NPN or PNP diffusions, an unwanted short can be caused by a parasitic field effect. This can be prevented via so-called channel stoppers, e.g., by increasing the vertical distance between the wires and the substrate to reduce the strength of the electric field, or by inserting additional diffusions that increase the doping concentration in the substrate and impede an inversion. Another event that leads to a malfunction of an IC can be observed if an isolating p-n junction accidentally starts conducting and thereby injecting minority charge carriers into the nearby region. This effect may lead to the formation of parasitic bipolar structures resulting in substrate currents and error currents in neighboring active regions. One layout measure to prevent this problem is to collect the diffusing minority carriers with additional p-n junctions which are denoted as guardrings. Such guardrings can not only be inserted for reliability reasons but also to protect sensitive circuitry from parasitic influences that detract from the circuit's accuracy.

Other reliability-critical effects ($\rightarrow$ and possible remedies) are: latchup, which leads to an escalating fault current due to positive feedback ($\rightarrow$ insert additional well and substrate contacts), hot electrons that are injected into the gate oxide of a transistor and shift its threshold voltage ($\rightarrow$ decrease the doping near the transistor's drain diffusion) and substrate debiasing, caused by voltage drops due to substrate currents ($\rightarrow$ insert an additional current-free net in star-wiring topology). Some problems that entail reliability faults cannot be parried with mere layout measures but also require attention already during circuit design. Examples include electrostatic discharge –ESD– (which requires special protection circuits), and dielectric breakdown (which has to be considered during circuit design by avoiding overvoltages). Both of these events can destroy the thin gate oxides of MOS transistors and lead to circuit failure in the field.

Yield

The third category of functionality-relevant layout quality aspects relates to yield losses, where chips instantly become nonfunctional due to variations in the fabrication process. With the exception of systematic process variations that affect entire wafers, many yield losses can be reduced through dedicated layout measures. As an example, using double-cut vias and minimizing the total via number is not only a reliability precaution with respect to electromigration, but has become a more and more important yield factor because it lowers the chance of open vias caused by particles and contamination. Similarly, being able to create a layout with as few routing layers as possible is a qualitative asset since less layers mean fewer processing steps and thus a lesser likelihood of inducing defects during the fabrication.

Some yield losses are not necessarily defect-induced but rather design-induced since they result from highly layout-dependent physical problems [179]. One such problem can be encountered in the fabrication process during layer planarization via chemical mechanical polishing: to avoid surface dishing due to overpolishing, certain density rules must be met in the layout. Another problem is the already mentioned plasma-induced gate oxide damage (discussed in the context of Figure 3.8 found on page 39). To prevent this so-called antenna effect, several remedies are available during layout design, enforced by dedicated antenna rules. As the names imply, density rules and antenna rules represent design rules –which are even covered by the DRC– but on the other hand, they are more intricate than other design rules (e.g., minimal wire width) and are therefore also akin to design constraints.

Apart from the possibility of reducing fabrication-induced yield losses through such specific measures as described above, the manufacturing yield can also be increased by minimizing the total area of an IC. If more dies can be fabricated per wafer, then the percentage of nonfunctional dies becomes smaller. The effect is in fact two-fold since this percentage comprises those dies that become defective due to inevitable fabrication parasitics, as well as the scrap dies that are unusable because of their location on the rim of the wafer. A further issue in this context is that the contour of a chip layout must satisfy a certain aspect ratio given by the chosen semiconductor package. This Fixed Outline constraint also applies to the shape of a layout block, which has been previously specified during floorplanning.

4.2.2.2 Consistency

Although physical functionality is the primary concern in IC design, a layout's consistency is also a highly important qualitative facet. To understand this criterion, one may imagine the extreme case of a layout that is given as a completely flat cell where any modifications can only be made on polygon level. Without providing any (nonphysical) meta-data about the logical organization, hierarchical structure and parameter values of the modules that constitute the layout, such a scenario severely detracts from the overall layout quality in two regards: transparency and maintainability.

Transparency

The notion of transparency is meant to capture several aspects. First and foremost, it deals with the question of how well a designer can see through a layout that has –for example– been created by a synthesis tool. Feeling insecure about such a layout solution is more than a matter of discomfort: as long as no formal verification method is capable of checking all design constraints in an automated way, a visual inspection by the human expert remains indispensable for evaluating the quality of a layout solution in terms of functionality. Signing off such a layout without assuring oneself of its correctness (to the best of one's knowledge and belief) not only risks technical failure in the field but can even become a serious issue of legal responsibility.

Another aspect in that context is reproducibility, which in turn relies again on determinism. On the one hand, this allows a designer to retrace and thus gain more insight into the automatisms by which a layout was produced. On the other hand, reproducibility also offers the possibility of re-running the same procedure or algorithm with some slightly different options, in order to effectuate selective improvements. Volatile solutions which are not reproducible should at least allow the designer to conveniently enact manual adjustments, e.g., to let best practice experience flow into the final solution where the automatism failed to include such knowledge. For that purpose, it is advantageous if the layout is organized in familiar modules as known from previous legacy designs. Furthermore, this modularization is supposed to match the corresponding schematic circuit, such that schematic and layout are completely SDL-conform.

Maintainability

SDL conformity is not only a matter of immediate transparency, but can also be quite instrumental in maintaining a design throughout subsequent design steps (and beyond). In this regard, maintainability indicates how well a layout and its corresponding schematic can be further processed (worked upon) until the final mask data is taped out. This requires compatibility with the other routines of the tool chain, e.g., for performing parasitic extractions that are then backannotated to the schematic for re-simulation. Since an IC design seldomly turns out to be “first time right”, a certain degree of structural orderliness is expected in the layout and the schematic to allow for swift design iterations without unpleasant complications. This argument is not to be shrugged off too carelessly, since modern design environments have become so convoluted that any inconsistencies in the design can easily lead to netlisting difficulties on the schematic side and verification problems on the layout side, for example.

Design iterations can also be necessitated by PDK updates, whereby design rules and design components (i.e., the behavior of the PDK's primitive device generators) can change before the design is finished. Here, the consistency of the layout is pivotal for handling such a PDK update. If the layout contains the affected design components as parameterized instances, these can be re-evaluated according to their new behavior; otherwise, if for example the layout is completely flat (i.e., has no hierarchy), the mere polygons do not reflect the update's changes in the layout and therefore require an utter re-design. If design rules have changed, it can be extremely helpful if the layout is hierarchically organized via parameterized cells on device and module level. This allows to correct DRC errors of certain modules in one fell swoop by revising only the respective master PCell, since this revision is then automatically taken over by all PCell instances throughout the entire layout.

After tape-out, the maintainability of a design can be even more rewarding with respect to re-use, considering the design's applicability in later IC projects. In many cases, only specific parts of the design are of interest, which again emphasizes the importance of appropriate modularization. With the intention of fostering re-use, industrial design teams decide to gather certain prototypical modules and store them in dedicated re-use libraries for potential applications in other designs. This practice may not only be directed to designs of the same semiconductor technology, but also in farsighted anticipation of future projects with denser integration – driven by the desire to put more functionality into an IC without increasing its total area. This is where the consistency of a layout can have a dramatic impact on the convenience of migrating the design to upcoming technology nodes. In the best case, only the primitive devices of the old design need to be substituted by the new PDK components while the remaining design entities adapt themselves to the new technology in a more or less automatic way. Of course, this is still wishful thinking, but the organizational structure of a layout can provide some valuable groundwork here nonetheless.

4.3 Summary

For practical application, an analog layout methodology has to be both technically feasible and economically viable. Unfortunately, traditional assessment criteria commonly referred to in literature are largely More Moore oriented and do not adequately cover all relevant aspects. For that reason, this chapter provided a more comprehensive overview of criteria that need to be taken into account for a thorough assessment of any layout methodology. The considerations distinguish between two basic factors: the impact on the design productivity of a potential customer and the layout quality that can be achieved. All in all, it is determined that assessing the value of a layout methodology requires to answer four elementary questions:

- Design productivity:
  - How much effort and expense must the customer put into the methodology?
  - What does the customer gain from the methodology in terms of efficiency?
- Layout quality:
  - Is the resulting layout good enough to fulfill the desired circuit functionality?
  - How well does the layout structure resemble the consistency of a manual design?

As the enormous amount of criteria behind these questions indicates, giving firm answers is anything but trivial. This is no surprise because one could say that making an objective assessment is not just a question of More Moore, but a quite difficult More than Moore challenge which reflects the qualitative complexity of analog design. This of course also implies that no generally-admitted, formal assessment metric can be conceived since the emphasis on (or relevance of) certain aspects always varies from case to case. Nevertheless, having unfurled an expedient chart of assessment criteria in Section 4.2, the preceding discussion can act as a good starting point and will be revisited for evaluating the layout automation methodology which is about to be developed in Part II of this thesis.

Part II

The Methodology

Chapter 5

Clarification of the Task

When people will not weed their own minds,
they are apt to be overrun by nettles.
Horace Walpole (British politician)

This chapter reprises the considerations of Part I in a nutshell to capture the essence of the task that is to be treated in this thesis. As was discussed in the previous three chapters, that task exhibits three different tenors: (1) the technical aim to achieve, (2) the scientific challenge behind that aim, and (3) the practical ambition of this work.

5.1 Technical Aim

Chapter 2 explicated that the problem of *analog layout* is a severe bottleneck in the design flow of *integrated circuits* because automation attempts have persistently failed to satisfy the high quality demands of practical analog ICs or are too narrowly focused on specific applications. In industrial environments, analog layout design is still done in a largely manual fashion today and heavily relies on human expertise to cope with the *More than Moore* character of the analog domain. That invaluable *expert knowledge* of seasoned layout engineers is pivotal for satisfying the diverse host of intricate *design constraints* that need to be taken into account to obtain the desired *functionality*. While some design constraints can be concisely expressed through *formalized* constraint representations, many requirements are simply communicated in a *nonformalized* way due to the difficulty and effort for describing them formally.

Existing automation approaches can be basically divided into *optimization-based* approaches and *generator-based* approaches. The former automation strategy works algorithmically and is capable of considering formally expressed design constraints *explicitly*. In contrast, the latter automation strategy works procedurally and can consider design constraints only *implicitly*, but without the need to formalize them. So far, no automation approach supports a well-balanced consideration of *formalized and nonformalized* constraints, although both kinds of constraint representation are equally indispensable to cope with the *qualitative complexity* of the analog design problem. For that reason, the technical aim of this thesis is to combine the advantages of the two automation strategies into a novel *layout methodology* able to consider design constraints both *explicitly and implicitly*.

5.2 Scientific Challenge

In Chapter 3, the working principles of optimization algorithms and procedural generators have been illustrated. An *optimization algorithm* works by iterating through a *repetitive loop* of solution space exploration and candidate evaluation. Thereby, a proposed candidate layout is successively refined with respect to explicitly expressed optimization goals as well as confinements of the solution space. Due to the problem complexity, optimization algorithms usually focus on one specific design task (floorplanning, placement, or routing). A *procedural generator* follows a *straight sequence* of commands to create a customizable layout, depending on the generator's input parameter values. Procedural generators for layout design are commonly implemented as *parameterized cells* (PCells) that can be used by being *instantiated*. In contrast to an optimization algorithm, a procedural generator has the natural ability to perform both placement and routing simultaneously.

A characteristic trait of optimization algorithms is that they translate the layout problem into an abstract mathematical model and optimize exactly the modeled aspects – not one thing more. Examining that strategy, this thesis identifies a distinct underlying paradigm denoted as *top-down automation*. In contrast to optimization algorithms, which self-intelligently *find* a layout *solution* at runtime, a procedural generator merely (re-)produces a layout *result* while the actual solution is *preconceived* by the human expert who implemented the generator. So, procedural generators follow a fundamentally different automation paradigm here denoted as *bottom-up automation*. While optimization algorithms rely on an *abstraction* of the problem (to *lessen* the degrees of freedom) and repeatedly *re-invent* the solution anew, procedural generators pursue the *generalization* of a known solution (to *increase* the degrees of freedom) which facilitates a parameterized way of layout *re-use*.

Relying on a complete formalization of the problem at hand, optimization algorithms are *universalists*: they can be flexibly applied to a wide range of circuits, but concede losses in layout quality (due to the inconvenience of having to *formalize all* design requirements, and also because of the algorithmic inability to take all these design requirements into consideration). In contrast, procedural generators are *specialists*: they produce layouts in full-custom quality but are limited to specific design tasks like automating a certain device type or circuit class (and may still require significant predevelopment effort for covering sufficient parametrical variability and to *anticipate all* design requirements within the range of the respective design task in advance). Hence, the strengths and weaknesses of top-down and bottom-up automation precisely complement each other. This insight suggests, that converging the two paradigms towards a new *bottom-up meets top-down* flow may not only facilitate a consideration of both formalized and nonformalized constraints, but could be the key to *close the gap* of analog layout automation one day. However, the heterogeneity of top-down and bottom-up automatisms poses the scientific challenge of investigating how these two *fundamentally different paradigms* can be combined at all.

5.3 Practical Ambition

Apart from establishing the theoretical foundations for a substantially new *layout methodology*, this thesis also intends to underline its practical usefulness. For that purpose, the field is plowed in two regards: not only to prepare the ground for the philosophy of an innovative *bottom-up meets top-down* flow, but also to straighten out some misconceptions from the past 30 years of analog EDA. In particular, this adverts to the mistake of comparing analog automation approaches in a More Moore style, as is traditionally done in literature. Chapter 4 argued that a thorough assessment adheres to many diverse criteria and is therefore a nontrivial More than Moore issue. In short, evaluating a layout methodology is supposed to consider both its impact on a customer's *design productivity* (which may not only yield an *efficiency gain* but also demands some *effort* and *expense*) as well as the degree of achievable *layout quality* (which has two facets referred to as *functionality* and *consistency*).

Based on these ruminations, the practical ambition of this work is to demonstrate its implementation in practice and to assess its true value for industrial application. Knowing about the tremendous complexity of analog layout design (and the unfortunate habit of previous EDA achievements to overestimate oneself), it is only fair not just to point at the assets of the new methodology but also to give an honest account of its shortcomings and to concede for which scope of problems it is really suitable. While providing an objective comparison of analog layout approaches is difficult in general, this is particularly true for the thesis at hand because its merit is not merely to extend some existing automation technique in one specific regard, but to pioneer an innovative, rather unconvential, and quite interdisciplinary multi-agent approach. Building upon the basic idea of distributing the layout design tasks in a decentralized way, Chapter 6 introduces the phenomena and the concepts from which that systemic approach can be deduced, before Chapter 7 describes the complete methodology in detail.

Chapter 6

An Interdisciplinary Approach – Preliminary Considerations

The future is already here – it's
just not very evenly distributed.
William Gibson (American-Canadian writer)

This chapter provides some preliminary considerations for an understanding of the interdisciplinary approach that will be taken by the layout methodology proposed in this thesis. First of all, Section 6.1 articulates the basic idea of tackling the analog layout design problem by *distributing* the design tasks across autonomously interacting, parameterized layout components. Next, Section 6.2 calls attention to three seminal terms in that context: (1) *decentralization*, (2) *self-organization*, and (3) *emergence*. For historical and didactical reasons, these three terms are then elucidated in reverse order: Section 6.3 illustrates the phenomenon of emergent behavior, while Section 6.4 discusses certain principles of self-organization, until Section 6.5 presents a relevant spectrum of existing models and implementations of decentralized systems. Finally, Section 6.6 alludes to the adaptation of these topics to the analog layout design problem, thus directly segueing into the description of the novel methodology in Chapter 7.

6.1 Divide and Conquer – Distribute and Conquer

As already discussed in Section 3.2, optimization algorithms and procedural generators follow two fundamentally different automation paradigms. But in one respect, it can be said that they share a common drawback: the determination of the layout solution is effected in a totally *centralized* way. Either, the algorithm has to solve the entire design problem all alone, or the layout solution must be completely preconceived by the design expert who develops the generator.

In the case of an optimization algorithm, one central optimization engine takes care of every design decision, thus performing the solution finding in a "dictatorial" fashion. However, the burden of having to consider all design constraints explicitly, leads to the *quality gap* shown in Figure 3.18 (page 61). Dividing the design problem *top-down* into separate placement and routing steps –as it is done in the digital domain– can surely reduce the problem complexity, but is obviously not a feasible pragmatism in the analog domain due to the strong interdependence of the different design tasks (as underlined by the example in Figure 3.12 on page 44).

In the case of a procedural generator, trying to realize a high-level circuit module with a powerful layout PCell, the developer of the PCell must be "prophetic" enough to foresee all design eventualities in advance. This, and the difficulty of covering the entire variability with a reasonable amount of development effort leads to the hunched-down curve and the *automation gap* in Figure 3.19 (page 63). Of course, the design problem can be unraveled hierarchically, using generators as building blocks to form more complex modules in a *bottom-up* fashion. Still, such an approach has not yet shown to be profitable enough above the level of *simple modules* (as defined in Table 2.1).

This thesis proposes a more concerted divide-and-conquer approach: a system, where layout design tasks are *distributed* across multiple autonomous layout components which are able to interact with each other to solve the design problem by themselves. The idea is motivated by the observation that such *decentralized systems* can outperform "classical" approaches when targeting problems of high complexity [180]. While it has been criticized that the term *complexity* is often used without giving a clear definition [181], one widespread view in *complexity theory* is that the complexity of a system is associated with the number of system components and the connections by which they are interrelated [182].

In [183], that view is phrased as: "in order to have a complex you need: 1) two or more distinct parts, 2) that are joined in such a way that it is difficult to separate them". This notion of a *complex system* is quite reminiscent of the *qualitative complexity* in analog layout design, where the design components are mutually so perturbing and the design constraints so heavily entwined, that it is not generally possible to decompose the problem in the conventional, purely reductionist *top-down* fashion, nor to assemble the solution in a constitutive, building-block-based *bottom-up* style. Instead, a *decentralization* approach is much more promising to tackle the problem – especially when taking a look at how remarkably interesting and seemingly intelligent decentralized systems can behave (Section 6.2).

6.2 Decentralization, Self-organization, Emergence

Nature demonstrates some striking examples of decentralized organisms, such as the flocking behavior exhibited by a group of birds: although each bird chooses its own course itself, their aggregate activity can lead to impressive cohesive motions [184]. Most prominently, as illustrated in Figure 6.1 (taken from [185]), large birds can often be observed to migrate in a V-formation, which is not only fascinating to watch but has some vitally important benefits such as reduced predation risk [186] and aerodynamic gains [187] (which also led to an adoption of such behavior in air forces: the so-called echelon flight formation for military aircraft [188]). Again, it should be emphasized that the collective flocking behavior arises –without central guidance– from the local actions of the flock's individual members. The birds manage to *self-organize*, wherein the joint achievement of the flock goes beyond the capabilities and the grasp of each individual animal. This form of *suprasummativity*[1] exemplifies an intriguing phenomenon called *emergence* [191], which can not only be encountered in nature but also in many other fields, as will be illustrated in Section 6.3.

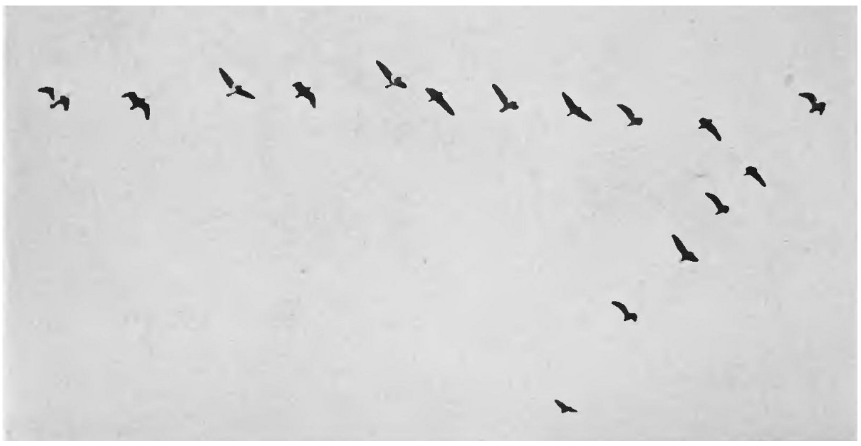


Figure 6.1: Flock of blue geese in V-formation, photographed at the Mississippi Delta [185].

If emergence occurs in a decentralized system, it can lead that system from an initially entropic state into a form of overall order. Such a process represents one acknowledged notion of *self-organization* [192] (although, providing a universal definition of self-organization is again a debate of its own [193]). An interesting question is, under which circumstances self-organization can occur – not only to explain

[1]The term *suprasummativity* originates from *Gestalt psychology* [189], where it denotes the principle, that the human mind acquires meaningful perceptions by forming a global whole that is independent of its parts in the perceptual system [190].

certain observable occurrences of self-organization but also to learn how self-organizing systems can be built to solve practical problems [194]. The latter intention is also pivotal for the work in this thesis, because the interaction of the layout components has to be engineered such that it evolves into a flow of self-organization, turning an initially disordered candidate layout into a "perfectly ordered" (DRC-clean, LVS-correct, and constraint-compliant) layout result. Research on the principles of self-organization is still rather juvenile, but many appealing theories have been formulated since the 1940s, particularly in the field of *cybernetics*. Notable works that are relevant to the topic of this thesis will be presented in Section 6.4.

A characteristic trait of decentralized systems is *distributed control* [195], which means that authority is allotted to multiple subsidiary entities of the system. With respect to layout automation, the benefit of suchlike decentralization is not just to share the workload, but rather to break down the complexity of the design problem – or, in other words, to meet the problem complexity with a system that is itself very complex (in the sense of [182] and [183] above). To achieve this, an important preparation is to decouple the system entities from the total complexity of the overall problem by making them autarkic (i.e., self-sufficient) and letting them interact with each other using only local information, without knowledge about the global state of the system [196]. In general, this also implies that the behavior of each entity is to be based on very few and very simple rules [197]. The relationship between such rules on the *microscopic* level and the resulting *macroscopic* behavior (terms used by [198]) are investigated in many existing models and implementations, some of which can be subsumed under the term *artificial life* (A-Life) [199]. Relevant achievements, from simulators of biological organisms to artificial systems that have been specifically designed for solving mathematical problems, will be covered in Section 6.5.

Interestingly, the close relatedness of the three topics emergence, self-organization, and artificial life (which will be detailed in the subsequent three sections), becomes especially apparent in the following formulation: "The theoretical focus of *A-Life* is the central feature of living things: *self-organization*. This involves the spontaneous *emergence*, and maintenance, of order out of an origin that is ordered to a lesser degree." [200]. The following Section 6.3 shows some illustrious examples of emergence.

6.3 Emergence: A Natural Phenomenon

The idea of a whole that is greater than the sum of its parts, dates back to Aristotle and the time of the ancient Greeks [201], but it was not until 1875 that the term *emergent* was coined by the English philosopher G. H. Lewes regarding certain effects of a chemical reaction [202]. These two perceptions already indicate that the definition of emergence has an *epistemological* aspect and an *ontological* aspect [203]. Epistemologically speaking, emergence is an attribute denoting that the global properties of a system cannot be reduced to the properties of its lower-order parts [204]. For example, the color of gold cannot be derived from examining a single gold atom [205]. In ontological terms, emergence is a process whereby new, collective structures evolve over time through interactions of simpler subunits [206]. This phenomenon accounts for the observation that intelligent behavior can arise from interacting entities that are individually not "terribly smart" [207]. The two aspects of emergence have lead to a distinction between different forms of emergence, as will be discussed in Section 6.3.1.

6.3.1 Forms of Emergence

In taxonomies for emergence, the epistemological meaning above relates to what has been denoted as *nominal emergence* [208] (Figure 6.2, left). Nominal emergence can further be divided into *intentional* and *unintentional* emergence [209]. As an example for intentional emergence, the function of a machine is an emergent property of its components (e.g., the ability to tell the time is an emergent property of a clock's inner gears and wheels). Unintentional emergence mainly refers to statistical quantities of aggregations with many identical particles (e.g., the volume of a gas). While the epistemological meaning of emergence has, above all, incited a philosophical debate about the overuse of reductionism in physics research [210], the ontological notion is not only more relevant for the work in this thesis, but also more spectacular (as the examples in this section are about to show).

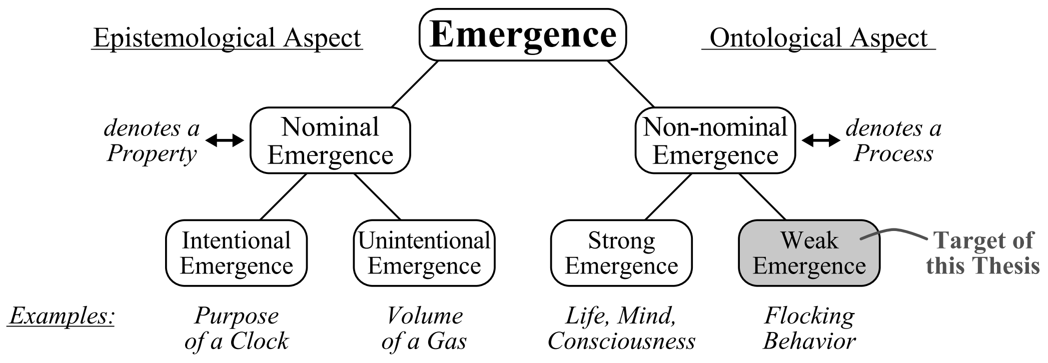


Figure 6.2: Overview of the different forms of emergence as commonly classified in literature.

The ontological, i.e., *non-nominal*, form of emergence is characterized by a dynamic behavior which is unexpected and unpredictable, whereat philosophers like to distinguish between *strong* and *weak* forms of non-nominal emergence [211] (Figure 6.2, right). Strong emergence (also denoted as *radical* emergence [212]) refers to processes that cannot even *in principle* be derived from lower-level processes [213]. Often-cited examples are life, mind, consciousness, and culture (also see [214]). However, the existence of strong emergence is contentious, with critics pointing to the obvious lack of evidence [215]. Weak emergence refers to processes where the macroscopic behavior is completely determined by the causal dynamics of its parts, but can only be predicted by performing a complete simulation [216] (which is often not possible because not all details about the parts and their dynamics are known [211]).

While the idea of strong emergence remains subject to philosophical discourses (surrounded by a lot of controversy), it is the understated notion of weak emergence that attracts scientific interest, mainly in complex systems theory [217], and also represents the target of this thesis. Speaking of weak emergence, the following real-life examples are by definition "weak" – but phenomenologically quite astonishing nonetheless. For that reason, the remainder of this thesis simply uses the term *emergence* to denote that form of non-nominal emergence which is so relevant for the presented work.

6.3.2 Emergence in Biology

Emergent cohesive motions of animate beings are not only exhibited by flocks of birds (as in Figure 6.1), but can also be observed in many other aggregations of congeneric creatures such as shoals/schools[2] of fish [218], herds of mammals [219], swarms of insects [220], or crowds of people [221] (see Figure 6.3, with pictures from [222], [223], [224], and [225]).[3] Behavior of that kind is collectively denoted as *swarm behavior* or *swarming*. As such, these two terms do not only apply to swarms of insects, but to animals in general and also to inanimate entities. A highly topical example for the latter case is swarm robotics [227], a field of interest that is predicated on biological studies of natural swarm behavior. Likewise, swarming has also been the inspiration for the population-based Particle Swarm Optimization technique discussed in Section 3.1.3.

A captivating example of intelligent swarm behavior is given by ant colonies such as in Figure 6.4 (image taken from [228]), especially in respect of the ants' collective ability to find the shortest path between their nest and a nearby food source [229]. As explained in [230], every wandering ant deposits a pheromone trail which in turn attracts other ants that intensify the pheromone trail. Since pheromones evaporate over time, the amplification of the pheromone density correlates inversely with the distance that the ants need to travel down a path and back again. Thus, the discovery of the shortest path *emerges* automatically from the individual movements of the ants. This pheromone-guided ant behavior has also found its way into computational optimization: it served as the blueprint for the Ant Colony Optimization algorithm (again, see Section 3.1.3).

[2]Shoaling means that some fish stay together and form a social group, swimming around arbitrarily. Schooling means that the fish are all swimming in the same direction.

[3]This kind of emergent behavior cannot be attributed to groups of canids because they involve dominance, e.g., an alpha wolf among a pack of wolves [226].

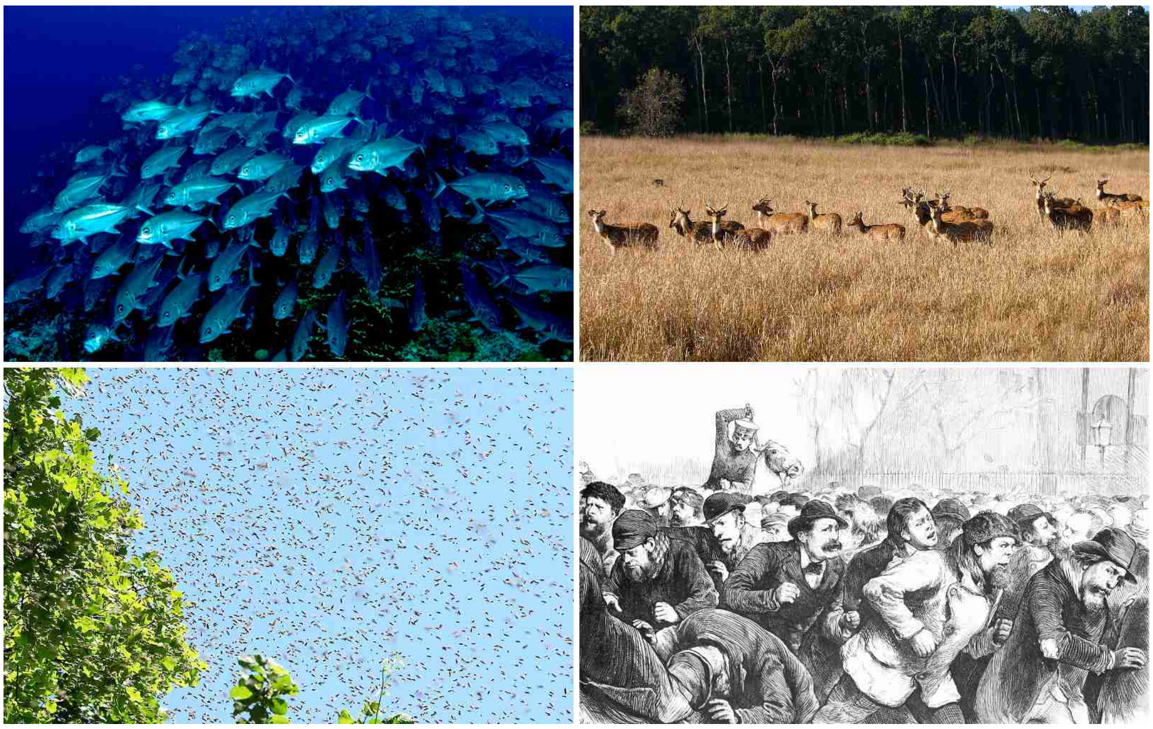


Figure 6.3: Emergent cohesive motions of animate beings: a school of jacks [222], a herd of deer [223], a swarm of bees [224], and a crowd of humans [225].

Figure 6.4: Safari ants foraging for food on the Chogoria route of Mount Kenya [228]. The shortest path to the food source emerges from the aggregate pheromone deposition of the ants.

6.3.3 Emergence in Physics

Emergent behavior can not only be found among sentient beings, but also on atomic and molecular levels. One shining example is crystallization, the process where microscopic particles aggregate in a thermodynamic solid state and form highly ordered lattice structures. In nature, crystallization is nicely demonstrated by snowflakes, where due to subtle differences in crystal growth conditions –depending on temperature and humidity– polymorphic crystal structures with different geometries abound [231]. Despite the immensely wide variety of sizes and shapes, the aggregation of the ice molecules often leads to an emergence of dihedral symmetry in symmetry group D_6 (i.e., six-fold radial symmetry) on the macroscopic scale [232]. This phenomenon is also shown by the magnified snowflake exemplar in

Figure 6.5 (photography by [233]), which has six variously shaped ice arms grown, thus constituting the characteristic stellate appearance well known from snowflakes.

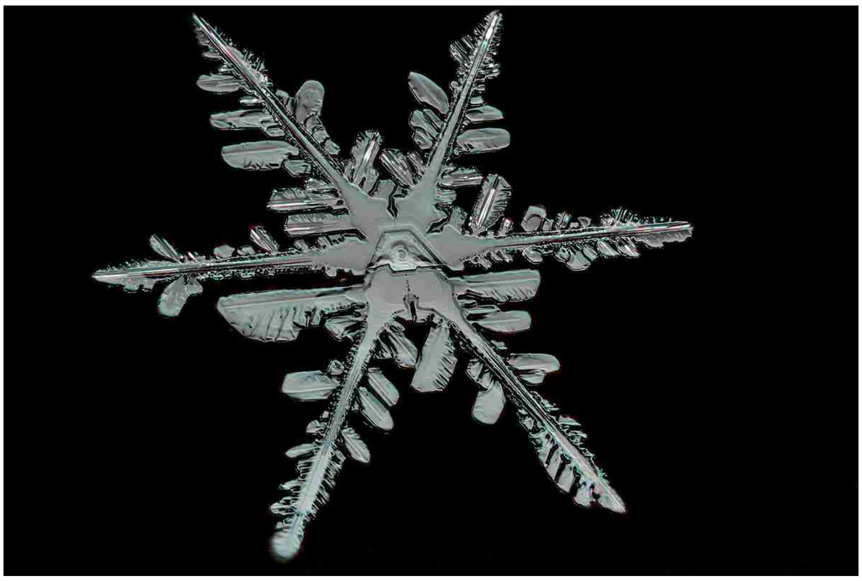


Figure 6.5: Stellate snowflake with dihedral symmetry [233] – an example of emergence in physics.

While crystals are structures of solid matter, emergence can also be encountered in liquids. For instance, one of the most commonly studied emergent phenomena in fluid mechanics is Rayleigh-Bénard convection [234]: in a thin layer of liquid which is slightly heated from the bottom, the temperature gradient leads to an upward conduction of thermal energy and thus an upwelling of lesser density fluid from the bottom layer, which –at a certain point– spontaneously becomes ordered on the macroscopic level (referred to as *spontaneous order*). This results in the appearance of rotating convection cells with regular patterns such as hexagons, eyes, and traveling waves (see samples in Figure 6.6, extracted from [235]). The formation of these so-called Bénard cells cannot be predicted due to their high sensitivity to the system's microscopic initial conditions. Suchlike behavior is denoted as *deterministic chaos* [236] (also known as the *butterfly effect*) and is subject to *chaos theory*. Another popular example of deterministic but chaotic behavior can be seen in the response of a double-rod pendulum [237].

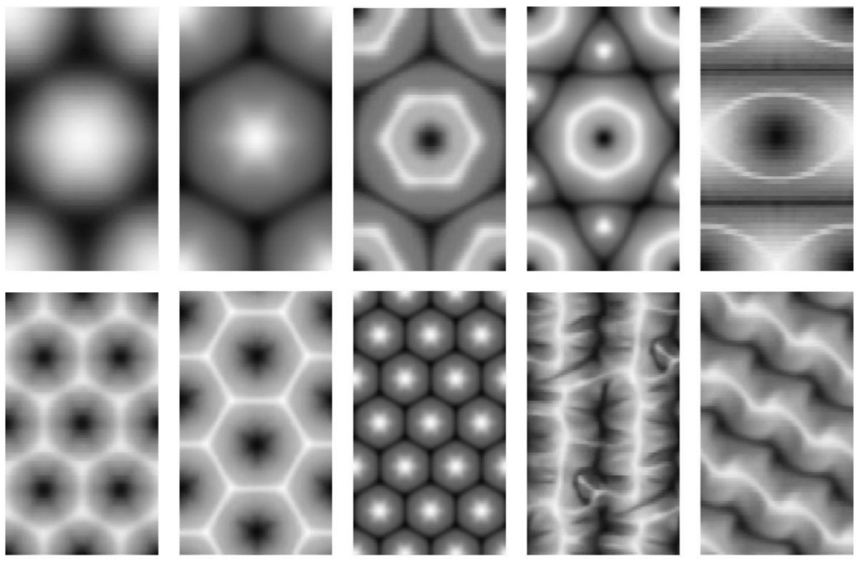


Figure 6.6: Samples of spontaneous order in liquids, emerging via Rayleigh-Bénard convection [235].

6.3.4 Emergence in Mathematics

Some geometric figures exert fascination for being self-similar at all scales. Such patterns are referred to as *fractals* and can also be considered as examples of emergence [238]. A fractal's infinite self-similarity emerges from a simple rule or operation that is applied *recursively*. While fractals can be readily found in nature (see, for instance, a Romanesco broccoli), they are a particular topic of scientific interest in mathematics – namely, in the branch of fractal geometry. One prominent fractal shape is the so-called Koch snowflake [239], which is constructed by drawing an equilateral triangle and then recursively modifying each side of the triangle by dividing the side into three segments and buckling the middle segment into another triangle. Figure 6.7 (created by [240]) shows a colored example of such a Koch snowflake (exhibiting the same radial symmetry as the real snowflake encountered in Figure 6.5).

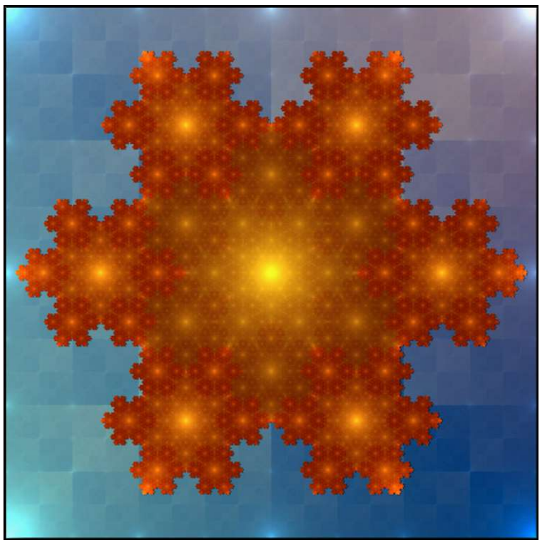


Figure 6.7: Fractal shapes like this Koch snowflake [240] can be regarded as examples of emergence.

Apart from attaining geometric beauty, fractal structures have also been used for engineering purposes. For instance, they find practical application in the design of fractal antennas as employed in telephone and microwave communications [241]. Due to their recursive pattern generation, fractals are commonly said to be self-replicative. Interestingly, it was work on self-replicating systems in the 1940s that led to the origination of a concept known as the *cellular automaton*. As will be described in Section 6.5.1, cellular automata are discrete models used in mathematics –but also in many other fields of science– to simulate dynamical systems and investigate emergent phenomena. The thematic relatedness of these topics is further confirmed by the fact that *recursivity* is considered to be one principle of self-organization, as Section 6.4 is about to touch upon.

6.4 Principles of Self-organization

The terms *emergence* and *self-organization* are sometimes used synonymously (e.g., see the debate in [242]), but they actually denote two different concepts and can exist in isolation [243] (although, as [244] puts it, this can be considered a matter of definition). As an example, a tornado is an emergent weather phenomenon but it would be presumptuous to call its appearance a product of self-organization. Vice versa, a group of castaways on a deserted island might democratically self-organize themselves, without any ranking of leadership, in order to survive – in that case, the ultimate purpose (i.e., survival) is only *nominally* emergent. The crucial question is, what ingredients are in general required –or helpful– to achieve a flow of self-organization, especially with the intention to provoke a form of emergence which is not merely nominal. Section 6.4.1 first introduces four basic constituents of self-organizing systems before Section 6.4.2 to Section 6.4.8 elaborate on seven self-organization fundamentals discussed in literature. Inhowfar all these ingredients can also be found in the layout methodology worked out by this thesis will be subject to Section 7.5.

6.4.1 The Basic Constituents of Self-organization

Referring to [245], which contemplates self-organization from a managerial point of view, a company team involves –apart from (1) the team itself, i.e., a group of interacting *workers*– three more constituents in order to self-organize: (2) a set of behavioral *rules* by which the workers may act, (3) *pressure* to get the group going, and (4) clear *goals* that are made known to the group. Urged on by these goals, one can say that there is a perpetual cycle of pressure being exerted on the workers and thus stimulating them, with the workers constantly trying to satisfy that pressure by acting according to the given rules (while each worker may simultaneously attempt to pursue his or her own individual goals). These four constituents, illustrated in Figure 6.8 (a), can in general be found in any self-organizing system. And as will be shown in Section 7.1.1, they also represent the pillars of the designated layout methodology – however, with one subtle reservation, because if the goals are completely known to the workers, achieving these goals is only an intentional, *nominally* emergent property.

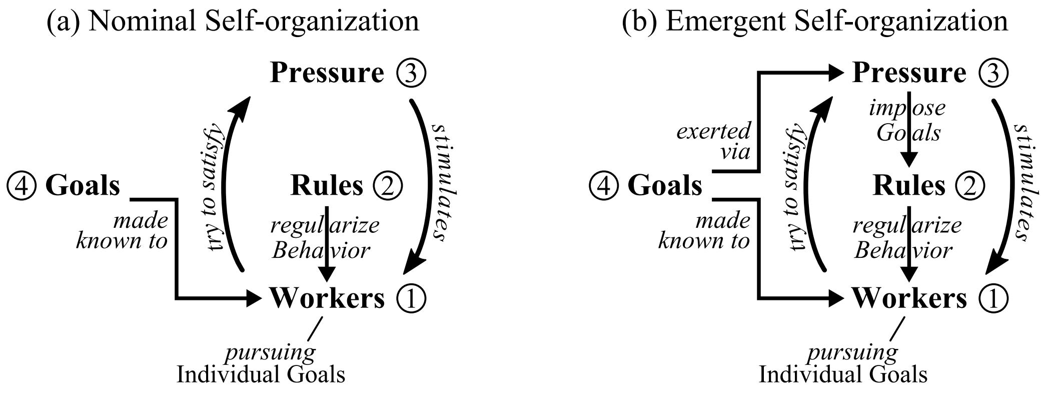


Figure 6.8: The basic constituents of self-organization: (1) workers, (2) rules, (3) pressure, (4) goals.

The situation is different in insect societies such as ant colonies (see Section 6.3.2), where labor is said to be *distributed*: such a system is accredited with the advantage that individual workers do not need to share information about how to achieve the desired collective aim [246]. Although shared knowledge can be necessary in a cooperating company team, the ant colony is better off without it: assessing the global state of the system would even be disadvantageous for the colony because "seeing the whole" is both perceptually and conceptually overburdening for any single ant [247]. In that case, detaching the workers from the colony's overall goal is even beneficial and yields an emergent form of self-organization which is not merely nominal. This poses the dilemmatic question, how suchlike emergent behavior can be aroused to solve a specific problem without plainly setting the ultimate goals. As will be covered in Section 7.1.1, this thesis takes an approach –indicated in Figure 6.8 (b)– where the achievement of the goals is not effectuated all directly (i.e., by making them completely known to all workers) but also by imposing the goals via the pressure and thereby steering the workers' actions in an indirect fashion.

6.4.2 Operational Closure and Structural Coupling

Exerting pressure to evoke emergent behavior means to put a self-organizing system "on the spot" by changing its environmental conditions. If the system is unable to influence these conditions, its only possible response is to accommodate itself to the new situation by changing its own internal structure. A system capable of maintaining itself in this way, can be called *autopoietic*.[4] According to [249], autopoietic systems are *operationally closed*, i.e., neither does the system alter its environment nor does the environment directly participate in the systems internal operations. For that reason, *operational closure* entails that the system must rely on self-organization (at least from the environment's perspective).

In contrast to *allopoietic* systems, the connection between an autopoietic system and its environment is not a causal chain with inputs and outputs, but a latent relationship referred to as *structural coupling* [250]: the environment cannot directly manipulate but only irritate the system to spur a self-produced

[4]The term *autopoiesis* was coined in cognitive biology to explain the nature of living organisms [248].

change (which, over time, is denoted as *structural drift*). At the same time, an autopoietic system is said to be *cognitively open*. This *cognitive openness* denotes that the system does not exist in isolation but is receptive to events in its environment and able to adapt itself to environmental perturbations [251]. Such a system, which is capable of increasing its chance of survivability in a turbulent world by accommodating itself to a changing environment, is also called a *complex adaptive system*.

6.4.3 The Edge of Chaos

The work of [252], which examined the behavior of cellular automata (see Section 6.5.1) in 1990, identified a phase transition from highly ordered to highly disordered dynamics which was likely to produce *emergent computation* (because near that phase transition, the cellular automaton had the greatest potential for the support of information storage, transmission, and modification). The transition region, located in the vicinity of the border between stable and unstable automaton behavior, became known as the *edge of chaos* (in [253] initially denoted as *onset of chaos*). Figure 6.9 (adapted from [254]) illustrates the edge of chaos as the order parameter regime where high-level structures of a system appear on the macroscopic scale. The *spontaneous order* of these structures is said to be *metastable* because it is usually short-lived and can be rapidly replaced by disorder due to disturbances of that balanced state.

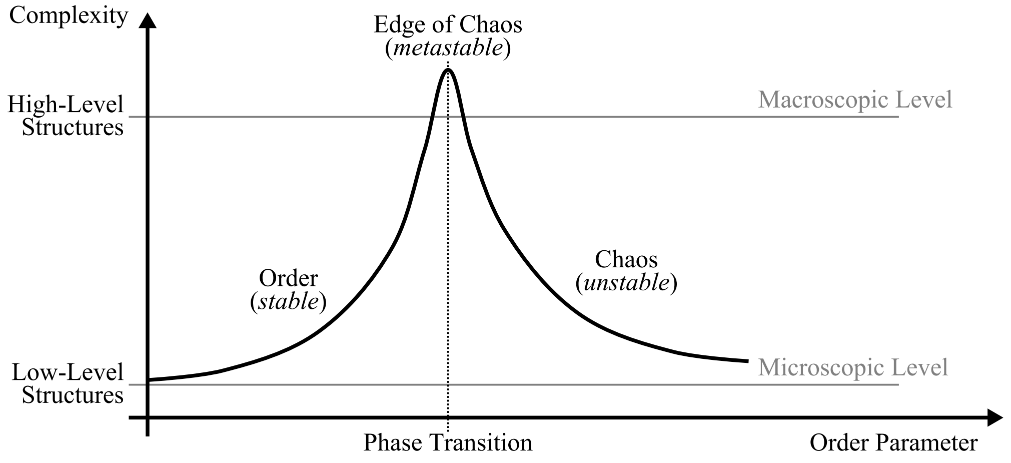


Figure 6.9: Operating at the edge of chaos [254] is considered as one principle of self-organization.

Until today, the phrase *edge of chaos* has also been adopted in various other fields of science (including physics, biology, and sociology) to describe the observation that many systems operate in a region between order and chaos, where the capabilities of the system are maximal [255]. For example, regarding creativity in cognitive systems, it is alleged that innovation occurs right at the edge of chaos [256]. On this note, but in more general terms, complexity theorists note that the proximity of a system's operating point to the edge of chaos is decisive for the evocation of emergent behavior and self-organization [257].

6.4.4 Recursivity and Feedback

A constant background theme of self-organization is *recursion* [258]. It is often put on record that a system's ability to reach a stable –or metastable– configuration (also denoted as an *attractor*) involves some kind of *recursivity*. In the context of *systems theory*, recursivity means that the output of some operation serves as the input for its next operation, i.e., cause and effect mutually influence each other [259]. In [260], such a recursive system is called a *nontrivial machine* and described as a *finite state machine*: it has the property that the response for a certain stimulus is not necessarily the same when the same stimulus is applied at a later time. Thus, in contrast to a *trivial machine* (which effectively denotes a combinatorial circuit, i.e., a system whose output depends only on its input), the behavior of a nontrivial machine is history-dependent and analytically unpredictable even though it is synthetically deterministic. This analytical unpredictability means that an experimenter –without knowledge about the inner workings of the machine– is not able to "crack the code" just by observing sequences of input-output pairs. According to [261], all autopoietic systems (Section 6.4.2) are organized recursively.

Recursivity is commonly understood as *feedback*, whereat two types of feedback are discerned: *positive feedback* and *negative feedback*. Regarding the latter, control cycles for error-controlled regulation are based on a negative feedback loop which counteracts a system variable's deviation from the target value (the so-called setpoint). In contrast, positive feedback (sometimes also referred to as self-enhancement, amplification, or autocatalysis) reinforces fluctuations of a variable and thus promotes changes in a system. For that reason, positive feedback can result in a snowballing effect that may even destroy the system (*resonance disaster*), but interestingly it can also lead to an equilibrium instead [262]. As an example, one may consider the foraging behavior of ants (see Section 6.3.2): the pheromone deposition is continuously increased by attracting more ants that deposit further pheromones. However, suchlike stabilization in fact also involves negative feedback in the end: an exhaustion of resources [263]. In this example, when the entire ant colony ultimately follows the pheromone trail, there are no more ants left to amplify the pheromone density any further. In complex self-organizing systems, there usually are several interlocking feedback loops, both positive and negative.

6.4.5 Stigmergic Interaction

While *operational closure* and *structural coupling* (Section 6.4.2) designate a form of oblique correlation between a system and its environment, a similar relationship called *stigmergy* [264] can also be encountered among the entities that interact within the system. Stigmergy is a mechanism of indirect coordination whereby entities in a decentralized system communicate with each other not directly but via modifications of their environment [265]. Stigmergy was first observed in social insects, with one example (already given in Section 6.3.2) being the way ants exchange information by laying down traces of pheromone. Accordingly, algorithmic Ant Colony Optimization techniques involve a deposition of "virtual pheromones" [266].

Another –nonbiological– example is provided in [267], where self-organizing traffic lights stigmergically co-control each other via the car density of the traffic that they regulate. This example, displayed in Figure 6.10 (adapted from [267]), illustrates that stigmergy is not only an integral part of collective intelligence in natural organisms, but that it is also considered to be an expedient principle for designing self-organizing systems [268]. For instance, the work of [269] implements stigmergic interaction between a group of mobile robots whose movements are coordinated through local configurations of their environment.

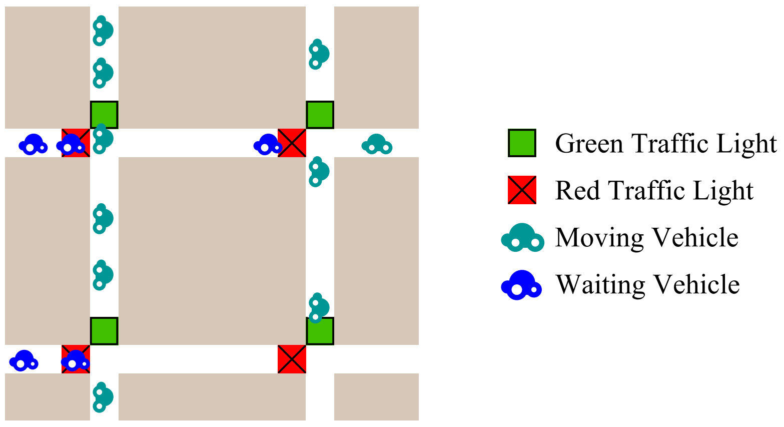


Figure 6.10: Stigmergy: the traffic lights do not communicate directly, but via the regulated traffic [267].

6.4.6 Reducing Friction and Promoting Synergy

One term that regularly appears in the context of emergence and self-organization is *synergy* [270]. There even exists a dedicated scientific branch called *synergetics* [271] (the science of cooperation), but still, the term *synergy* is ambiguous. Often, synergy is paraphrased with the Aristotelian formulation that the whole is greater than the sum of its parts [272]. In that case, the term is synonymous to the *epistemological* understanding of emergence (see Section 6.3.1). In [273], synergy is referred to as the combined effect of an interaction, so the term is used in the *ontological* sense of emergence. Sometimes, synergy does not denote the effect of an interaction but the interaction itself (e.g., in [274]). Despite this equivocality, there seems to be consensus in two regards. First, synergy always indicates a form of cooperation. Second, while emergence rather represents a long-term phenomenon resulting from many interacting entities, the term synergy can apply to a single (inter-)action.

Mirroring both of these aspects, the author of [275] understands synergy as a correlation or concourse of action. In this spirit, but with a focus on the design of self-organizing systems, the author of [276] speaks of synergy when the action of an entity is not only beneficial for the entity itself but also for the other entities (and thus for the entire system). So, a *synergistic* action amounts to a "win-win" situation. If an entity increases its own satisfaction at the cost of the other entities, this is denoted as *friction*. The main premise of the work in [276] is that reducing friction and promoting synergy in a self-organizing system will result in better performance. This is backed by the observation, that evolutionary processes also tend towards synergistic relationships on the level of genes and genomes [277].

6.4.7 The Virtue of Selfishness

As already mentioned in Section 6.4.1, it can be advantageous if the interacting entities of a self-organizing system have only local information and elide any global knowledge about the system's overall state. Carrying this notion a bit farther, one might suggest to design the individual entities not only as *ignorant*, but even as *selfish* [278], so as to dumb down their decision-making. Intuitively, it may appear doubtful to assume that a group of such egoistic individuals could interact in a way that is beneficial for the system as a whole. However, a strong case for this idea of egoism can be made by studying nature once again. In particular, the *selfish herd theory* [279] states that –in contrast to previous beliefs– the gregarious behavior of animal species is not owing to mutual benefits of the animals but results from selfishness: in the face of predators, each animal tries to put other conspecifics between itself and the predator, which inevitably leads to an aggregation of the herd –as for example seen in Figure 6.11 (photo by [280])– and to a reduction of predation risk for the entire group. Thus, the aggregate behavior of that population is not rooted in cooperation but in competition.

Figure 6.11: According to the selfish herd theory, animals aggregate out of selfishness (photo: [280]).

The characteristics of such competitive settings are mathematically investigated in *noncooperative game theory* (see Section 6.5.2), where egoistical *players* are called *self-interested*. However, as [281] points out, self-interested does not necessarily mean that the players want to cause harm to each other, but that they act according to their individual preferences and within their own scope of action. Regarding the contemplations of Section 6.4.6 about *synergy*, this remark connotes that –although synergy implies cooperation instead of competition– the selfishness of an entity can still allow for synergistic actions. An equivalent notion can also be found in economics, where the metaphor of Adam Smith's *invisible hand* describes that society can benefit from the self-interested efforts of individuals [282].

6.4.8 Law of Requisite Variety

A herd's reaction to predatory danger (as discussed in Section 6.4.7) exemplifies the pivotal question which escorted all of the preceding ruminations in this Section 6.4: how apt is a self-organizing system in responding to disturbances that emanate from its environment? One fundamental answer to that question is found in Ashby's *law of requisite variety* [283]. By Ashby's definition, *variety* denotes the total number of potential distinct states of a system. In a more general sense, variety is the repertoire of actions available to a system. Ashby's law (also known as the first law of cybernetics [284]) says: the larger the repertoire of actions available to a system, the larger the variety of environmental disruption the system is able to compensate.

Ashby's law of requisite variety can be understood as a condition for dynamic stability under external perturbations, and is also considered as a simpler version of Shannon's Tenth Theorem [285]. Two further notions shall be brought up here: first, according to Ashby, variety is a measure for the complexity of a system [286], and second, Ashby's law has also been phrased as "variety absorbs variety" [287]. These two notions elucidate the statement that was made in Section 6.2 about the approach taken by the methodology in this thesis: to address the complex problem of analog layout with a system that is itself very complex. In other words, variety of action is required by the methodology to cope with the many degrees of freedom and the diversity of constraints that are so characteristic of the analog IC domain.

6.5 Models of Decentralized Systems: A Form of Artificial Life

Inspired by the phenomenological occurrences of emergence (see Section 6.3) and with respect to the oft-enunciated principles of self-organization (Section 6.4), various models of *decentralized systems* have been proposed since the middle of the 20th century, targeting the examination of emergent behavior as well as the practical utilization of self-organizing structures. And although no approach specifically addresses the problem of layout automation in analog IC design, it is worthwhile to survey a couple of relevant achievements. Therefore, this Section 6.5 presents an overview of existing concepts in that field, particularly concentrating on those ideas that have influenced the work of this thesis (as will be indicated in each of the following subsections and comprehensively summarized in Section 7.5). It has already been mentioned that a couple of implemented (i.e., computer-based) models of decentralized systems imitate natural biological processes and are, for that reason, often referred to as *artificial life* [288]. A famous example –the *Game of Life*– will be given in the following Section 6.5.1, which covers the concept of *cellular automata*.

6.5.1 Cellular Automata

The concept of cellular automata, suggested by Stanislaw Ulam and taken on by John von Neumann for his work on self-reproducing systems in the 1940s [289], can be considered the first exemplification of artificial life. A cellular automaton is a space- and time-discrete mathematical model, given as an n-dimensional lattice of cells each of which behaves like a *finite state machine*. At any point in time, all cells are in one of a finite number of states, and with every time step, each cell changes or keeps its state according to a simple *transition function* that involves the state of the cell itself and the states of its neighboring cells [290]. In their most illustrative form, cellular automata have two dimensions (represented as an orthogonal grid of adjacent squares), each cell switches between two states, and a cell's neighborhood is comprised of the cell's eight surrounding cells.

One example of such a cellular automaton is *Conway's Game of Life* [291], which was devised in 1970 and attracted much interest in and beyond academia. In Conway's Game of Life, every cell is in a state of being "alive" or "dead" and –depending on its neighboring cells– either dies (due to isolation or overpopulation), lives on (to the next generation) or becomes alive (as if by reproduction) at the next time step. The fascination with the Game of Life emanates from its visualization of emergent behavior, because –based on this simple set of rules– quite interesting patterns arise after several generations. These include static patterns (still lifes), periodic patterns (oscillators), moving patterns (gliders, spaceships), self-copying patterns (replicators), patterns that emit other patterns (guns, rakes, breeders), and patterns that take a large number of generations to stabilize or vanish (methuselahs). Figure 6.12 shows some examples of such patterns as well as a Garden of Eden (an orphan pattern that has no predecessor and can therefore not appear beyond the automaton's initial configuration). Apart from the surprising ways, in which such complex configurations evolve over time, a scientifically interesting aspect of the cellular automaton is its computational power: since it is possible to construct counters and logic gates by letting gliders interact with each other, Conway's Game of Life is *Turing complete* [292].[5]

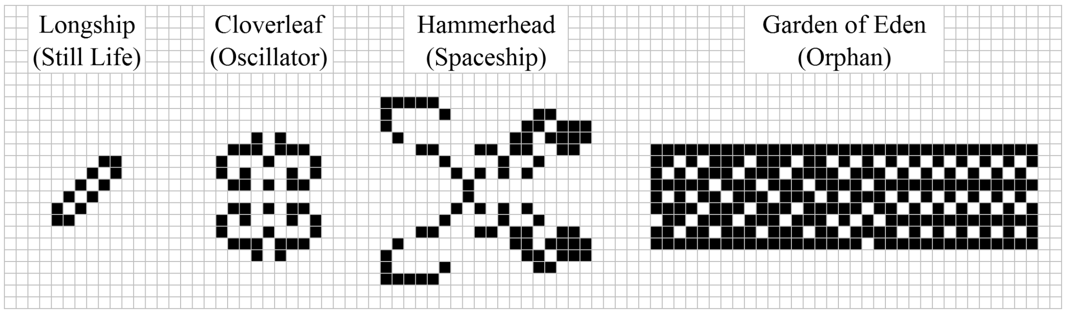


Figure 6.12: Examples of patterns in Conway's Game of Life, a two-dimensional cellular automaton.

Another Turing complete system that can be described as a cellular automaton is *Langton's Ant* from 1986 [293]. It simulates step-wise orthogonal movements of a single virtual ant whose decisions are based on two simple rules. Over thousands of steps, these rules lead to complex behavior, as the ant can be observed to first create plain symmetric patterns, which later become chaotic and irregular, until the ant's motion eventually leads to the emergence of a straight, diagonal, indefinitely repeating ant trail. Similar to Conway's Game of Life, Langton's Ant is not only interesting for its capability of universal computation (proven in [294]), but for the evolvement of captivating patterns such as spirals and snowflake-like, hexagonal structures (in a generalized concept known as *Turmites*). Langton's Ant can be implemented within *Wireworld*, a cellular automaton first presented in 1987 [295]. The cells in Wireworld cover four different states and behave with respect to four very trivial rules. Like the previously mentioned examples of cellular automata, Wireworld is computationally universal, despite the simplicity of its state transitions. It is particularly designed to simulate electronic circuits and has even been used to implement a Turing complete computer.

Until today, many types of cellular automata have been developed and are being investigated for various practical applications such as image recognition, task scheduling, and cryptography [296]. Apart from that, their computational capabilities and the diversity of emergent patterns soon inspired scientists to draw certain parallels between cellular automata and the complexity of natural processes. Already in 1969, Konrad Zuse proposed that even the evolution of the universe itself is, at heart, digitally computed by some sort of cellular automaton, presuming that physical laws are not continuous by nature but discrete [297]. In 2002, Stephen Wolfram published an extensive study of cellular automata to pronounce their importance for science in general, supposing huge potential for providing insight into the behavior of complex systems in all kinds of fields [298].

The bearing of cellular automata for this thesis is plain: just like the simple interactions of an automaton's cells leads to the emergence of "living" higher-level structures, the designated layout methodology attempts to let the overall layout solution emerge from the dynamics of interacting lower-level layout components. And as will be seen in Section 7.5, some particular aspects of cellular automata (e.g., including an entity's local areal neighborhood into its decision-making, or changing the "state" of an entity) can also be identified in the novel layout methodology.

[5]In computability theory, Turing completeness signifies that a system can be used to simulate a Turing machine (which is an abstract mathematical computation model). This means, that such a system is generally capable of performing all data manipulation tasks which are also accomplishable by real-world computers.

It is interesting to note that the main inventor of cellular automata, von Neumann, was also one of the founders of modern *game theory*, as follows in Section 6.5.2 (by the way, Conway's Game of Life is also referred to as a zero-player game).

6.5.2 Game Theory

One key discipline to model the self-interested decision-making of entities in a decentralized system is game theory [299], which is the mathematical study of strategic interaction among independent rational *players* who try to maximize their so-called *payoff*. Game theory primarily serves the examination of *coalitional* or *competitive* behavior in such situations and the prediction of probable *outcomes*. Commonly, this is achieved by identifying so-called *solution concepts* which allow game theorists to deduce *equilibrium strategies* for each player. The most influential solution concept in game theory is the *Nash equilibrium* [300], where no player can increase his or her own payoff by unilaterally choosing another strategy. The payoff is usually represented by a *utility function* which –in short– maps a player's preference for a possible outcome to a real number.

Game theory distinguishes between many different types of games. Considering the criteria that can also be recognized in the work of this thesis, a game can be

- a *cooperative game* (where groups of players can form coalitions) or a *noncooperative game* (where each player makes individual choices),
- a *discrete game* (where all players choose from a finite set of strategies) or a *continuous game* (where strategies are chosen from an infinite continuous set),
- a *symmetric game* (where the payoffs for a strategy are independent of the player, as in Figure 6.13) or an *asymmetric game* (where changing the identities of the players changes their payoffs, as if the players had taken on different roles),
- a *constant-sum game* (where each player's gain/loss comes at the expense/benefit of the other players) or a *non-constant-sum game* (where the aggregate gains and losses are not the same for every outcome),
- a *simultaneous game* (where all players choose their strategy without knowing about the other players' choices) or a *sequential game* (where players decide one after another, having –at least some– knowledge of the other players' prior choices),
- a *perfect-information game* (a sequential game where a player always knows all choices previously made by the other players) or an *imperfect-information game* (a sequential game where a player may have only partial or no knowledge of earlier choices).

If a game is played multiple times, the game being repeated is called the *stage game*. A game, that is played a finite and known number of times, is denoted as a *finitely-repeated game*, whereas a game that is played infinitely often or a finite but unknown number of times is referred to as an *infinitely-repeated game*.

Games in game theory can be represented in various ways. The most fundamental game representation is the *normal form* because many game-theoretic settings can be reduced to normal-form games. Simply put, the normal form allows to represent the payoffs for a set of n players, depending on the vector of chosen actions (called an *action profile*), in an n-dimensional payoff matrix. A textbook example of a game that can be represented in normal form is the well-known *Prisoner's Dilemma* [301] (depicted in Figure 6.13). Since the normal form does not support any notion of sequence, sequential games are often reasoned about in the so-called *extensive form* representation. An example of such an extensive form game, where the successive actions of the players are represented in a rooted game tree, is *Centipede* [302]. Both the Prisoner's Dilemma and the Centipede game illustrate the paradoxical case that two players might choose not to cooperate although both would be better off by doing so.

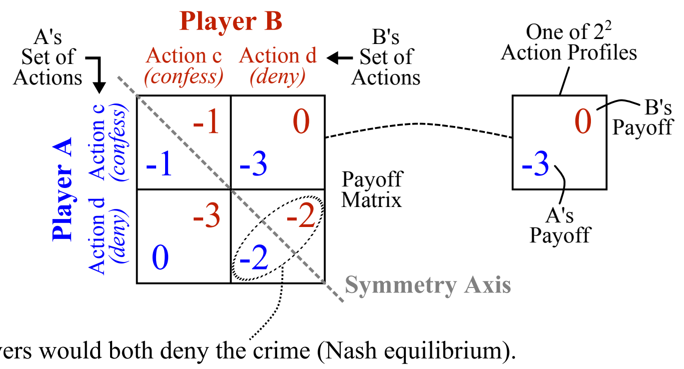
Game theory is used to model real-life situations of conflict and coordination. In the 1940s, game theory was initially developed to understand economic behavior [303], but today it also plays an important role in disciplines as diverse as political science, biology, psychology, philosophy, and computer science. With respect to this thesis, game theory is interesting since the decision-making of the interacting layout components in the novel layout methodology can also be regarded from a game-theoretical

Description

Two criminals A and B are imprisoned, not able to communicate with each other.

- If A and B confess the crime, then both remain imprisoned for 1 year.
- If A confesses and B denies the crime, then A is imprisoned for 3 years and B is freed.
- If A denies and B confesses the crime, then B is imprisoned for 3 years and A is freed.
- If A and B deny the crime, then both remain imprisoned for 2 years.

Normal-Form Representation



➡ Two purely rational self-interested players would both deny the crime (Nash equilibrium).

Figure 6.13: The Prisoner's Dilemma – a noncooperative, discrete, symmetric game in normal form.

point of view. However, these layout components are no mathematical conceptions of abstract players but concrete computational entities – such as the entities used to build *multi-agent systems*, which will be the topic of the subsequent Section 6.5.3.

6.5.3 Multi-Agent Systems

Multi-agent systems [304] are a relatively new field of computer science that followed a 1980s trend from classically centralized approaches of artificial intelligence towards distributed control. A multi-agent system is defined as a network of loosely coupled *agents* that interact with each other in a given environment. Therein, an agent [305] is a computational entity such as a software program or a robot (with the corresponding environment being a virtual space or a real physical setting, respectively). Such an agent is commonly accredited with the following characteristics:

- the agent is able to perform actions upon its environment,
- the agent is autonomous in its decisions about how to act,
- the agent has a particular degree of intelligence (and often mobility),
- the agent has a limited viewpoint with only incomplete information about its environment,
- the agent is rational in that it adheres to a goal-directed behavior following clear preferences.

According to [306], agents can be grouped into the following five classes: simple reflex agents, model-based reflex agents, goal-based agents, utility-based agents, and learning agents. As a side note in reference to Section 6.5.1, multi-agent systems are related to cellular automata with respect to the consideration that both implement a *complex system* (in the sense of [182] and [183], as mentioned in Section 6.1), albeit at two ends of the spectrum: compared to each other, cellular automata are complex systems with simple agents, whereas multi-agent systems are simple systems with complex agents [307].

In [308], multi-agent systems are denoted as concepts of distributed artificial intelligence that complement artificial intelligence and artificial life. Although multi-agent systems can (and are usually meant to) exhibit some form of collectively intelligent behavior, they must be distinguished from certain methods of *swarm intelligence*, which are investigated in the context of mathematical optimization and thus represent an algorithmic sub-branch of artificial intelligence. As already discussed in Section 3.1.3, a method of swarm intelligence such as Particle Swarm Optimization (for which a lot of different applications exist [309]) is a population-based optimization technique where the solution space is explored by multiple *particles*, each of which represents one candidate solution to the optimization problem. This is a fundamental difference towards a multi-agent system, in which every agent contributes to solving a global problem but is not itself a solution to the problem.

While the idea of multi-agent systems has not gained widespread recognition up until the mid-1990s, interest in the field is now growing significantly, to some extent spurred by the initiated technical evolution towards the Internet of Things and Industry 4.0 [310]. Today, the usefulness of multi-agent systems is actively examined in a large host of diverse applications, such as aircraft maintenance [311], supply chain management [312], and energy resource scheduling [313], just to name a few. With the thesis at hand, the problem of layout automation is added to that palette since the interacting layout components of the new layout methodology can –according to the five characteristics above– also be understood as agents. However, an extraordinary trait therein is that each of these agents is not only meant to help in solving the actual layout problem but in fact represents a part of its solution.

It should be mentioned that the term *multi-agent system* is sometimes confused with the notion of an *agent-based model*, although the two concepts are not the same – as will be explained in Section 6.5.4.

6.5.4 Agent-based Models of Collective Motion

There is a considerable overlap between multi-agent systems and agent-based models, but a subtle distinction can be made [314]: multi-agent systems are understood as an engineering approach (i.e., to solve particular problems), whereas agent-based models rather have a share in analytical science (such as in simulating the behavior of biological organisms). While agent-based modeling can be used to imitate a wide variety of decentralized systems in numerous domains, this section is particularly focused on models of collective motion, as exhibited by ensembles whose members perform spatial moves. Such models, in which the environment represents a geographical space, are called *spatially explicit* [315]. That conception also applies to this thesis, regarding the intention to let autonomous layout components interact with each other inside a layout cell.

The first suchlike agent-based simulation model was *Reynolds' Boids* from 1987 [316], an artificial life program that animates the flocking behavior of birds (or "creatures" in general). As illustrated in Figure 6.14, Boids implements a so-called *distributed behavioral model* where the collective, polarized motion of the flock emerges from three simple *steering rules* obeyed by each individual flockmate (referred to as a *boid*: a bird-like "bird-oid" object). In [317], these steering rules are denoted as *separation* (steer to avoid crowding flockmates), *alignment* (attempt to match the heading and speed of flockmates), and *cohesion* (steer to move towards the average position of flockmates). As in a cellular automaton, the steering rules involve a boid's local-only neighborhood, such that each flockmate reacts solely to nearby neighbors in its immediate vicinity.

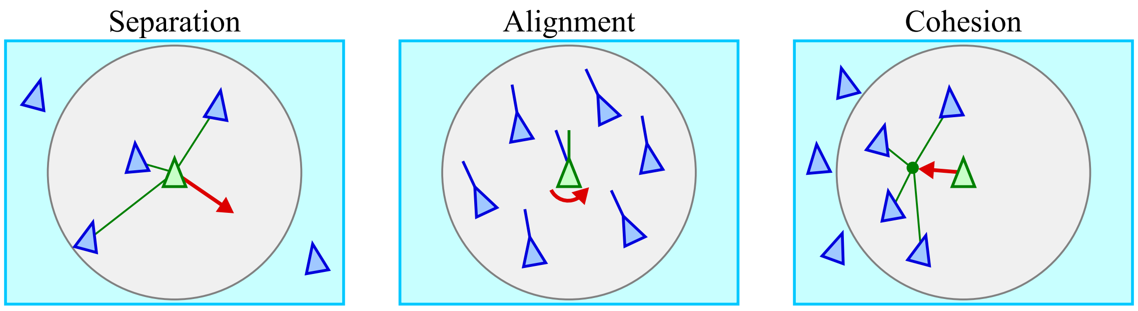


Figure 6.14: Depiction of the three simple steering rules in Reynolds' Boids (adapted from [317]).

Concerning the definition of such a neighborhood, a basic question is whether the environment of the boids' flock is two- or three-dimensional (as in [318]). Apart from this dimensionality, the neighborhood is further defined by its extent – a property for which two prevalent models can be consulted [319]. In the so-called *metric distance model*, a boid is only affected by those of its neighbors that are located within a certain range. This accounts for three different, concentric, circular zones (zone of repulsion, zone of alignment, zone of attraction) which correspond to the three steering rules above. In contrast, the *topological distance model* lets a boid pay attention to a fixed number of its closest neighbors, regardless of their respective distance. A hybrid metric-topological composition model is presented in [320].

Several behavioral extensions of Reynolds' original Boids program have been proposed by others. For example, the authors of [321] complement the alignment rule (see above) with a force that facilitates a change of leadership. The work of [322] provides an additional rule called escape and incorporates the effect of fear. This idea can also be found in [323], where the conception of Boids is adopted for the emulation of artificial fishes which are driven by three mental state variables: fear, hunger, and libido.

Emotions and sensations of this kind have also inspired certain facets of the work in this thesis. Another interesting aspect in that regard is memory, which has for example been worked into the animal group formation model of [324] and will also be an element of the layout methodology that Chapter 7 is about to present.

Encouraged by Boids, spatially explicit agent-based models can also be found in other disciplines. Examples include the cellular-automaton-based Nagel-Schreckenberg model for freeway traffic simulation [325], as well as the Vicsek model of nonbiological *self-propelled particles* (also referred to as self-driven particles) in physics [326]. A perseverative issue in suchlike agent-based modeling is the question of what the simplest possible sets of rules are that lead to the emergence of synchronized, collective motion [327]. However, it is important to note that in contrast to a flock of birds or a convoy of cars, the interacting components in the layout methodology provided by this thesis are not supposed to exhibit a unidirectional laminar movement but an overall flow that can rather be described as a motion of centripetal yet turbulent convergence.

For simulating the behavior of decentralized systems, a plethora of general-purpose agent-based simulation toolkits –both proprietary and open source– are readily available (with a comprehensive survey being given in [328]). As an example, the Java Agent Development Framework (JADE) [329] is a software framework for the development of intelligent agents in Java, based on the Foundation for Intelligent Physical Agents (FIPA) standards for hetereogeneous and interacting agents. However, tools such as these are of no further interest here due to the specialized topic of this thesis, for which a tailor-made implementation had to be provided (as will be covered and demonstrated in Chapter 8).

6.6 Adaptation to the Problem of Analog Layout Design

Incited by the ideas portrayed in this chapter, the thesis at hand strives to address the task of analog layout automation by implementing a system of interacting layout components with distributed control. The conceptual approach behind this methodology is highly interdisciplinary, drawing from (A) the developments of *decentralized systems*, (B) the investigations of *self-organizing structures*, and (C) the observations of *emergent behavior* discussed in the preceding sections:

(A) The composition of the proposed system follows the idea of *decentralization*. Hence, the setup of the taken approach is in line with other models of artificial life as covered in Section 6.5.
(B) The interaction demeanor of the system's individual layout components is carefully worked out in an endeavor to pay respect to the principles of *self-organization* described in Section 6.4.
(C) The behavior of the overall system is supposed to let full-custom quality layouts emerge from its execution, mirroring the natural phenomenon of *emergence* as illustrated in Section 6.3.

The detailed elaboration of the new layout methodology is about to follow in Chapter 7. Therein, the overview of Section 7.1 describes the composition of the system (A), while the interaction demeanor of the system's layout components (B) is subject to Section 7.2, Section 7.3, and Section 7.4. As already mentioned, parallels to existing models of decentralized systems as well as the incorporation of the presented principles of self-organization into the work of this thesis are summarized in Section 7.5. The emergent behavior of the system (C) will be depicted in the examples of Chapter 8.

Chapter 7

The Methodology: Self-organized Wiring and Arrangement of Responsive Modules

In our narrow house of boards, bestride
the whole creation, far and wide;
move thoughtfully, but fast as well,
from heaven through the world to hell.
Johann Wolfgang von Goethe: Faust
(translated by Walter Kaufmann)

This thesis proposes an analog layout automation methodology where layout components interact with each other in a *decentralized* flow of *self-organization* that leads to *emergent* behavior. As will be discussed in this chapter, the layout components –denoted as *responsive modules*– are autonomous, mobile, rational, computational entities (and thus *agents*, in accordance with the definition in Section 6.5.3). On that basis, the presented methodology neither implements a purely optimization-based *top-down* approach nor a purely generator-based *bottom-up* approach (as defined in Section 3.1.1 and Section 3.1.2), but represents a multi-agent system which teams the two automation strategies. The new methodology is henceforth referred to as *Self-organized Wiring and Arrangement of Responsive Modules* (SWARM) [330], wherein the terms arrangement and wiring denominate the tasks of placement and routing.[1] SWARM is meant to be used by layout designers for the creation of analog layouts that fit within a given, sufficiently large, rectilinear outline. As will be discussed here (and demonstrated in Chapter 8), the SWARM methodology is able to include both *formalized* and *nonformalized* expert knowledge into the automation by effectuating an *explicit* and *implicit* consideration of design constraints.

7.1 Overview of the SWARM Methodology

As will be outlined in Section 7.1.1, SWARM is composed of three core concepts that are dexterously correlated with each other. To give an impression of how these concepts are supposed to engage in a purposive interaction, Section 7.1.2 sketches the designated self-organization flow in a superficial example. In Section 7.2, Section 7.3, and Section 7.4, each of the three concepts will be discussed in a section of its own. Section 7.5 completes this chapter with some final remarks about the conception of SWARM, discerning it from optimization algorithms and then also taking a retrospective look back at Chapter 6.

7.1.1 The Three Core Concepts of SWARM

The left part of Figure 7.1 reprises the depiction of Figure 6.8 (b) from Section 6.4.1, stating that a truly emergent self-organizing system requires four basic constituents: (1) a group of interacting *workers*,

[1]Semantically, arrangement and wiring are thus used as synonyms for placement and routing here, but the choice of diction is deliberate. This is not only for the sake of the SWARM acronym but mainly because –in the parlance of EDA– the terms placement and routing are commonly understood as optimization-based automation techniques.

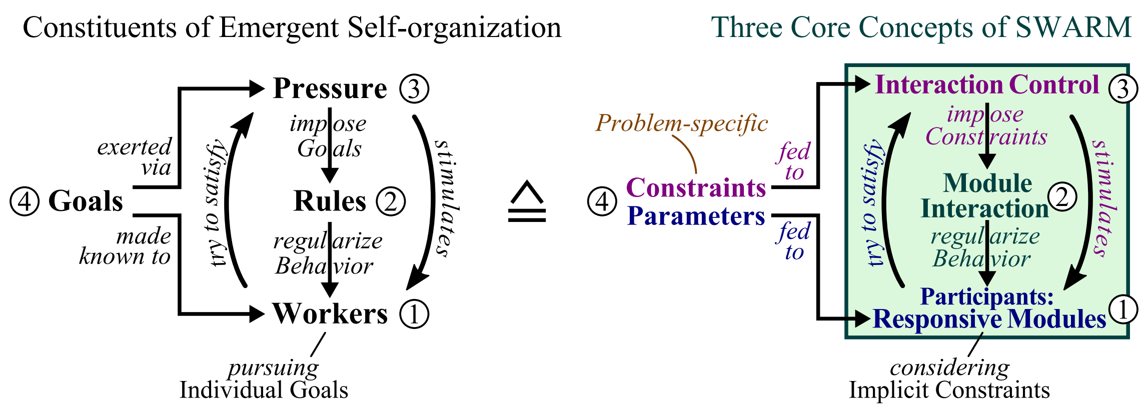


Figure 7.1: The constituents of emergent self-organization and the corresponding SWARM concepts.

(2) a set of behavioral *rules*, (3) the exertion of *pressure*, and (4) overall *goals* that can be directly communicated to the workers and indirectly inducted via the pressure. As illustrated in the right part of Figure 7.1, all these constituents can also be found in SWARM. Excluding the *goals*, which are always specific to the particular layout problem (i.e., the respective design restrictions and design objectives), the SWARM system itself is thus composed of three problem-independent core concepts:

Responsive Modules: The concept of the already mentioned *responsive modules* is SWARM's analogon to the *workers* in Figure 6.8. As Section 7.2 will discuss in greater detail, a responsive module is a *context-aware* procedural generator that can act on its own behalf within its design environment.

Module Interaction: SWARM allows a set of responsive modules to interact with each other and to arrange themselves inside a given layout territory. During this *module interaction*, each module acts according to a set of relatively simple behavioral *rules*, which will be the topic of Section 7.3.

Interaction Control: To steer the module interaction towards a compact and constraint-compliant layout arrangement, SWARM implements a supervising *interaction control* organ that exerts *pressure* by stimulating the modules and by imposing the design restrictions and objectives (i.e., the *goals*) onto the module interaction. The interaction control organ will be addressed in Section 7.4.

Before going into the three core concepts in detail, Figure 7.2 shows the layout automation approach of SWARM, as opposed to the working principles of optimization algorithms and procedural generators (see Figure 3.1 and Figure 3.13 respectively). SWARM combines these two automation strategies, since each responsive module has the *bottom-up* capabilities of a procedural generator, while the interaction control organ provides a *top-down* perspective on the design problem. Consequently, the design restrictions and objectives that can be committed to SWARM comprise both module parameters as well as formalized constraints.

On that basis, high-level constraints are imposed by the interaction control organ to be then *explicitly* considered during the module interaction, while each individual module simultaneously takes care of its innate low-level constraints *implicitly*. To underline the decentralized character of this module interaction approach, the layout that emerges from SWARM via self-organization is denoted as the layout *outcome* (taken from game theory and contrasting the terms layout *solution* in Figure 3.1 and layout *result* in Figure 3.13). Section 7.1.2 provides an exemplary depiction of SWARM's self-organization flow.

7.1.2 Depiction of SWARM's Self-organization Flow

Although various kinds of decentralized systems have been developed in the past decades (see Section 6.5), SWARM is the first suchlike system whose interacting entities are layout modules. As already done in Figure 7.1 and Figure 7.2, these interacting modules are subsequently denoted as *participants* in order to discern them from the miscellaneous notions of entities found in other existing systems, some of which are listed in Table 7.1.

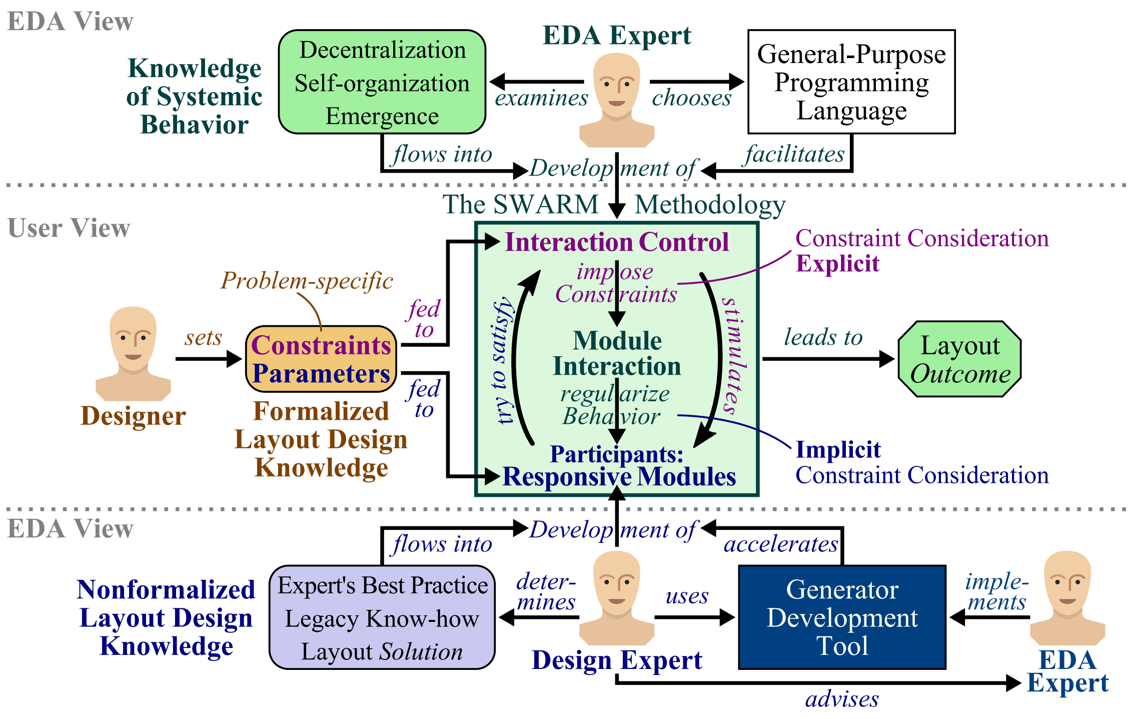


Figure 7.2: Working principle of the SWARM system, as opposed to Figure 3.1 and Figure 3.13.

Table 7.1: Denomination of interacting entities in selected examples of decentralized systems.

Name	Year	Description	Entities
Conway's Game of Life [291]	1970	Two-dimensional Cellular Automaton	Cells
Centipede [302]	1981	Extensive Form Game (Game Theory)	Players
Reynolds' Boids [316]	1987	Agent-based Simulation Model	Boids
Particle Swarm Optimization [309]	1995	Population-based Optimization Algorithm	Particles
JADE [329]	2001	Java Agent Development Environment	Agents
SWARM [330]	2015	Multi-Agent Layout Automation Approach	Participants

The self-organization in SWARM is structured as a so-called *run* which, in its most elementary form, follows the flow of control depicted in Figure 7.3. Given a set of participants (responsive modules), SWARM takes an initial module *constellation* that may for example be template-based, handcrafted, or randomly created. Next, SWARM takes the user-defined *zone* Z (which demarcates the available layout territory) and centers the outline of Z on the bounding box of the initial constellation. After this *initialization phase*, the supervising control organ enlarges Z such that its area is significantly greater than the sum of the participants' individual areas (see Section 7.4). Then, the *self-organization phase* starts with a so-called *round* of interaction, where each participant is allowed to perform an *action* (see Section 7.3). By performing an action, a participant undergoes a *transformation*: it may move inside the layout zone Z, may also rotate around its center, and may even deform itself into a different layout variant with nominally equal electrical behavior (see Section 7.2). The actions in one round are meant to emulate a concurrent behavior but are in fact executed sequentially.

Multiple rounds of interaction are repeated until no action at all is performed within one round. This situation is denoted as a *settlement* because each participant has *settled* at a definite position. If the module constellation is *viable*, which means that all participants are in a legal and satisfying position, then Z is slightly tightened to induce further rounds of interaction and thus another settlement (otherwise, the participants failed in finding an appropriate arrangement, and the SWARM run gets aborted). This tightening-settlement cycle continues until Z is *tight*, which means that Z has reached its original, user-defined size that marks the available layout space. The last settlement ends the SWARM run and represents the layout *outcome* of the overall self-organization. In the end, a *finalization phase* gives the opportunity to perform some dedicated post-processing on the obtained layout. This may involve auxiliary tools or can be accomplished manually.

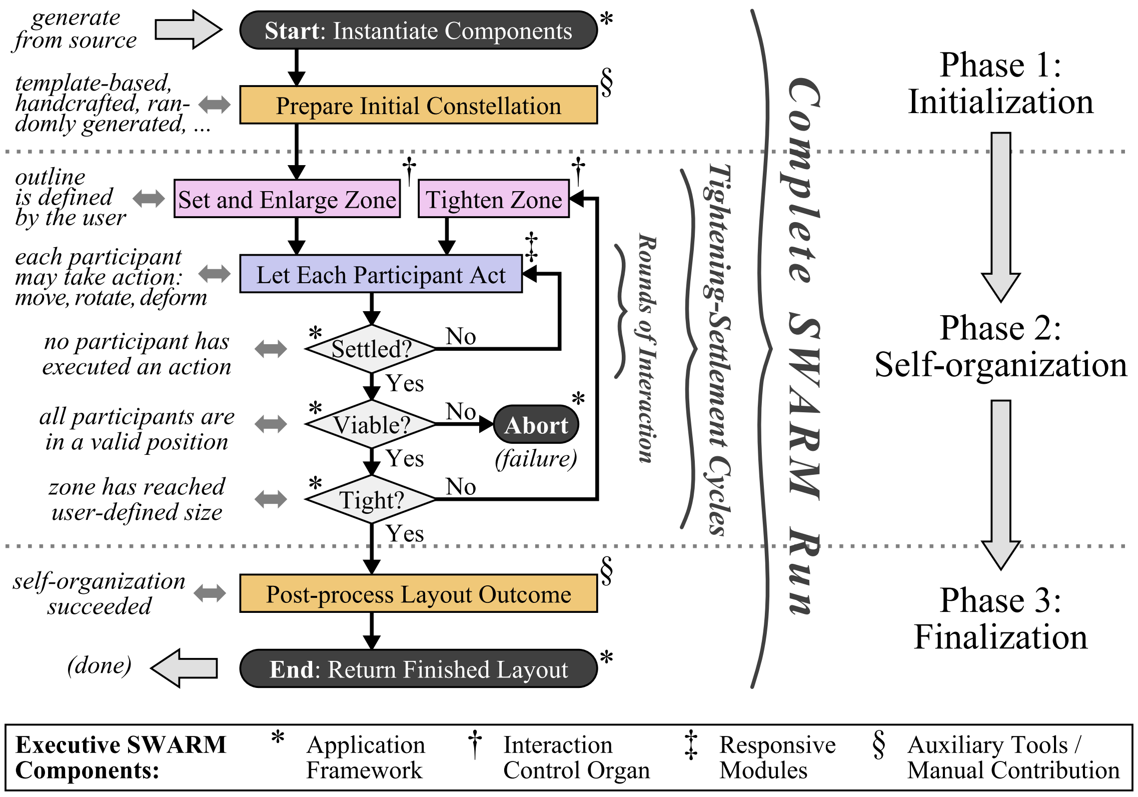

generate from source
Start: Instantiate Components *
template-based, handcrafted, randomly generated, ...
Prepare Initial Constellation §
outline is defined by the user
Set and Enlarge Zone †
Tighten Zone †
each participant may take action: move, rotate, deform
Let Each Participant Act ‡
no participant has executed an action
* Settled?
No
Yes
all participants are in a valid position
* Viable?
No
Abort *
(failure)
Yes
zone has reached user-defined size
* Tight?
No
Yes
self-organization succeeded
Post-process Layout Outcome §
(done)
End: Return Finished Layout *
Rounds of Interaction
Tightening-Settlement Cycles
Complete SWARM Run
Phase 1: Initialization
Phase 2: Self-organization
Phase 3: Finalization
Executive SWARM Components:
* Application Framework
† Interaction Control Organ
‡ Responsive Modules
§ Auxiliary Tools / Manual Contribution

Figure 7.3: Flow of control in a SWARM run (initialization, self-organization, and finalization phase).

Figure 7.4 provides an exemplary depiction of a SWARM run. The example shows seven participants being successively goaded towards a compact arrangement inside a rectangular layout zone. This illustration brings a vivid biological analogon to mind, because SWARM's idea of autonomous layout modules that perform areal movements across the plane of an increasingly tightened layout territory, imitates the roundup of livestock. For instance, a shepherd drives a group of sheep together just be encircling them (affecting the group like a predator – also see Figure 6.11), thereby leaving it up to every single animal to find its individual place among the herd. In SWARM, the participating layout modules can be thought of as sheep, with the supervising control organ taking on the role of the shepherd.

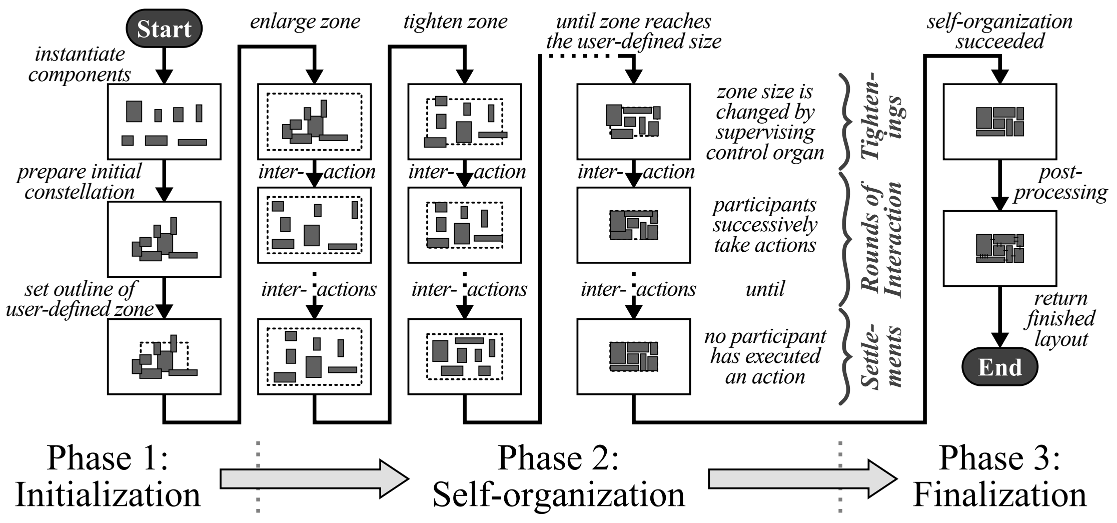

Start
instantiate components
prepare initial constellation
set outline of user-defined zone
enlarge zone
tighten zone
until zone reaches the user-defined size
inter- action
inter- actions
zone size is changed by supervising control organ
participants successively take actions
until
no participant has executed an action
Tightenings
Rounds of Interaction
Settlements
self-organization succeeded
post-processing
return finished layout
End
Phase 1: Initialization
Phase 2: Self-organization
Phase 3: Finalization

Figure 7.4: Exemplary depiction of a SWARM run, visualizing the analogy to the roundup of livestock.

During the module interaction, the contour of Z is employed as an explicit Fixed Outline (see Section 3.1.1.2) constraint. In that regard, the outline of the layout zone represents a strict confinement, but with the approach of making Z progressively smaller, it is thus used to pursue a basic optimization goal: minimizing the total layout area. In a similar way, SWARM is also able to address the other major optimization goal commonly found in analog layout design: minimizing the total wirelength. This issue is not depicted in Figure 7.4, but will be broached in Section 7.3.1. Also not shown in the example above is the fact that a participant is not necessarily a single module, but may be an *association* of multiple components. This aspect is the central topic of the following Section 7.2 about the first core concept of SWARM.

7.2 Responsive Modules

The *responsive modules* in SWARM are realized as procedural generators, enhanced with a couple of additional capabilities: they can react to changes of their environment (Section 7.2.1), they can be used to administrate a group of design components (Section 7.2.2), they can be imposed onto each other in a hierarchical fashion (Section 7.2.3), and they are able to assume a different layout variant from an exhaustive set of possible alternatives (Section 7.2.4).

7.2.1 Context Awareness

As already said, the layout modules in SWARM interact with each other as *agents*. This presupposes an ability to sense, reason, and act. Since a procedural generators is a piece of software, it can be readily implemented in a way that facilitates intelligent reasoning. But in their common form, procedural generators are conceptually not able to access their environment. Therefore, sensing and acting requires to enhance the conventional concept of procedural generators with a specific feature, subsequently referred to as *context awareness*. Context awareness allows an *instance* of a procedural generator (1) to *read* information from its design context and (2) to *modify* that context, including –self-referentially– the instance itself. With these abilities, the instance can realize a *responsive* design entity that reacts to environmental changes like a *reflex agent* does as shown in Figure 7.5 (a).

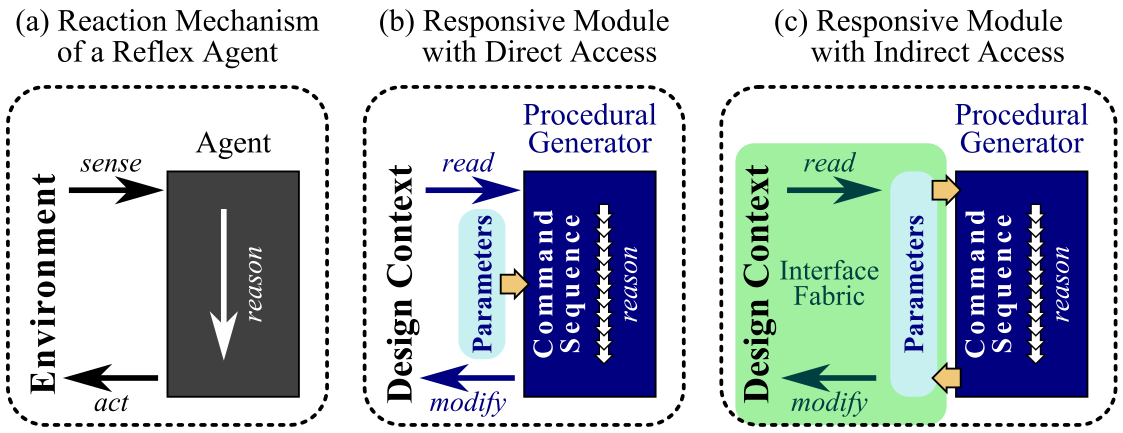


Figure 7.5: Via context awareness, procedural generators can react to environmental changes as agents.

The most straightforward way of implementing context awareness is to equip a procedural generator with *direct access* to its design context, as illustrated in Figure 7.5 (b). However, such an endeavor requires a fundamental extension to the IC design framework itself, which cannot be accomplished without additional efforts on the vendor's side. An initial implementation of that approach has led to the concept of *FR PCells* [331], but this concept is still in its infancy and no official release for productive use is available so far.

Therefore, as shown in Figure 7.5 (c), SWARM takes a different approach which facilitates *indirect access*, abusing a generator's parameters to communicate information between an instance and its design context, as also done in [139] and [140] (see Section 3.1.2.5). For that purpose, SWARM implements a dedicated *interface fabric* which reads data from the design context, encapsulates it in a single parameter value, and passes that value to the instance. After extracting and processing the context data, the instance also writes its response into a single value and stores it in a parameter (which is thus not used as an input parameter but as an output parameter). The interface fabric parses that response value and –if necessary– modifies the design context on behalf of the instance (as will be seen in Section 7.2.2). In contrast to a FR PCell, the instance must be actively re-evaluated if its design context is changed.

7.2.2 Governing Modules

Based on context awareness, a procedural generator for a *simple module* (in the sense of Table 2.1) can be easily realized as a single (i.e., self-contained) responsive layout module. This is feasible, if a corresponding generator such as a *circuit PCell* is available on the schematic side. If instead, the schematic circuit is flat, i.e., comprised of *primitive devices*, then the *Generate from Source* step (see Section 2.1.4) places corresponding devices in the layout.

One possibility to get to the module level on the layout side is presented in [332], which performs a circuit structure recognition in the schematic diagram to identify functional units and replace the respective devices in the layout with parameterized modules. A drawback of that technique is the problem to maintain SDL-conformity since the 1:1 device correspondence between schematic and layout has to be sacrificed for this purpose. A better method in that regard is made possible by the powers of context awareness, as it allows for using a responsive module in the layout to "manage" a group of devices. A responsive module of that kind is referred to as a *governing module*.

7.2.2.1 Temporary Context Duplication

If the context awareness of a governing module is implemented via indirect access, the need to gather a large regiment of detailed geometrical information (e.g., the individual pin positions, gate finger shapes, and bounding boxes of multiple transistors that need to be "managed") and encapsulate that information in a single parameter value may be cumbersome. A more elegant idea, which can be understood as *temporary context duplication* is presented in [141] (and is also pursued in this thesis): the module re-instantiates the devices internally so it can then determine all relevant geometries from these "clones" (which may be deleted afterwards). For that approach, the module requires only the type, the parameter values, the location, and the orientation of each external device. Based on that information, the utilization of a governing module generally follows an *adoption process* –written as a sequence $A = (Ab, Ad, Am, As)$– which is basically comprised of four operations (although not all of these operations are mandatory in every case):

(1) *Absorption*: "create an image of the outside" (symbol Ab)
The module reads the *external* device information (in the case of direct access) or receives it from the interface fabric (indirect access), and re-instantiates the devices internally.

(2) *Adaptation*: "measure and calculate inside" (symbol Ad)
The module determines all relevant device geometries and creates its *internal* layout as per these geometries, thus computing certain correcting quantities. The internal devices can then be deleted.

(3) *Amendment*: "modify the outside" (symbol Am)
The module transforms (moves and/or rotates) each *external* device according to the computed correcting quantities. If the context access is indirect, the transformation data is encapsulated in a parameter value and then put into effect by the interface fabric.

(4) *Assimilation*: "harmonize the inside with the outside" (symbol As)
The module repeats operations (1) and (2) in order to adjust its *internal* layout with respect to the new arrangement of its *external* devices (again, the internal devices may then be deleted). Now, the module and its context are supposed to be in perfect conformance with each other.

As an example, Figure 7.6 illustrates the utilization of a governing module which is based on the Differential Pair procedural generator from Figure 3.14. Initially, there are four transistors in the layout, as column **Context** shows. In (1), the module absorbs these transistors and instantiates them internally. Column **Module** displays the inner layout of the module instance. In the design, the module with its internal transistors lies precisely on top of the external transistors, as can be seen in column **Design**.

Next (2), the module determines the pin positions of all internal transistors, connects them by creating wires and vias following a preconceived routing scheme –thereby also computing the correcting quantities mentioned above–, and then deletes the internal transistors such that only the *wiring* remains inside the module. Afterwards (3), the module pushes the external transistors together according to the correcting quantities computed before.[2] In that *positioning*, the correcting quantities reserve sufficient vertical space to accommodate the wiring between the transistors, which clarifies why the correcting quantities were calculated in conjunction with the previous creation of the wiring. Finally (4), the module generates its layout anew to harmonize the wiring with the modified design context. In the end, the external transistors and the wiring inside the module constitute an optimal Quad layout.

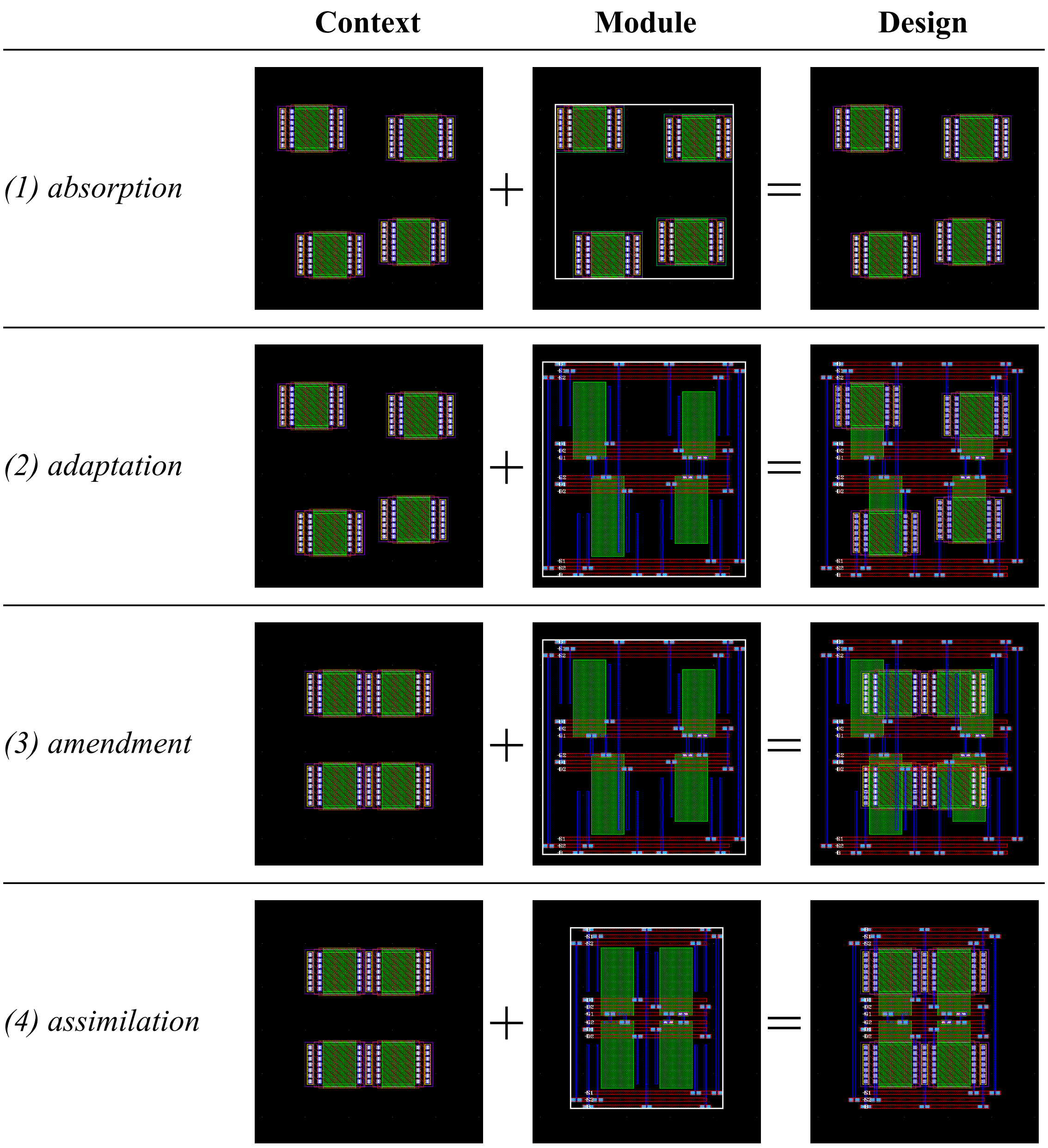


Figure 7.6: Adoption process of a governing module, exemplified for a Differential Pair (Quad layout).

[2]In this example, the basic interdigitation of the transistors –specified by their initial relative locations (1)– is not changed through the amendment operation. Instead, the transistors are only compacted (according to a module-specific, preconceived, simple, row-based compaction scheme).

7.2.2.2 Co-transformations in a Governing Module

With temporary context duplication, context awareness is not only utilized during the self-organization phase, but already in the initialization phase of a SWARM run: if the given schematic is flat, a governing module can be imposed on each group of devices that needs to be managed as a compound functional unit. From then on, it is the governing module's duty to care for these "children" it has adopted. That is to say, a transformation of the entire module during the self-organization is in fact a set of correlated transformations (*co-transformations*), which involves the transformation of the governing module itself and a corresponding transformation if its adopted children. So, if the governing module moves or rotates when interacting with other modules, it has to make sure that its children are also moved and rotated accordingly. To deform itself into another layout variant, the governing module repeats the adoption process and moves/rotates its children thereby. If the deformation includes a deformation of the children, the children are deformed first, and then the governing module repeats the adoption process.

Figure 7.7 gives an example of such a deformation, pertaining to the Differential Pair module above. Starting off with the previous module variant from Figure 7.6, the co-transformation of the adopted children consists of –first– deforming each of the four transistors from a 1-finger variant into a 2-finger variant (0), and –second– moving the transistors according to their new layout geometries, which is done in (1)–(3) as part of the adoption process. Then, the co-transformation of the governing module is to deform itself by assimilating itself in (4), which represents the last operation of the adoption process and sees the overall module layout turn into a 2-finger variant of the Quad.

7.2.3 Module Associations

Like in the examples of Figure 7.6 and Figure 7.7, it is usually the case that the positioning of a module's adopted devices strongly depends on the designated wiring. In that situation, it is suitable to perform both of these tasks with one single, *panfunctional* governing module, as it is done by the Differential Pair module above. If there is not such a strong dependence, two separate modules can be employed for the two different tasks.

Figure 7.8 shows a Current Mirror that involves three governing modules. One module is a pure Positioning module which does not create any physically relevant layout shapes and is therefore denoted as a *meta-module*. So, the adoption process A_{Pos} of that module includes only an absorption and an amendment operation (for which the correcting quantities are already computed during the absorption). This adoption process can be written as $A_{\text{Pos}} = (Ab_{\text{Pos}}, -, Am_{\text{Pos}}, -)$ to indicate that there is no adaptation and no assimilation involved (as is characteristic of a meta-module). Another module –referred to as a Wiring module– provides the connectivity. The third module is an Info module that writes device identifiers onto the transistors to display how they have been interdigitated. The latter two modules do not perform amendments (which eliminates the need for assimilations), so their respective adoption process A_{Wir} and A_{Inf} only consists of an absorption and an adaptation operation (see Figure 7.8). The entire construct, for the reason of involving more than one governing module, is called a *module association*.

7.2.3.1 Supreme Commanders

In every module association, one of its governing modules has to be implemented as a *supreme commander* which rules over the other modules (denoted as *associated modules*) and maintains the consistency of the entire construct. Thus, if the module association is about to perform a transformation, it is in the supreme commander's authority to apply the respective co-transformation to its adopted children and associated modules. If the transformation includes a deformation, the supreme commander also has to trigger the adoption process for each associated module once again. As it is the case in Figure 7.8, these are not supposed to perform amendments, but if they do, then they need to trigger the adoption process for the supreme commander anew (which again affects the entire module association). Attention must be paid in order to avoid an infinity loop here. This responsibility is left to the module developer, as well as the question which module is destined to act as supreme commander: normally, it is the module that performs the primary design task (as defined by the developer). If –instead of a module association– only one panfunctional governing module is involved, that module is by definition a supreme commander.

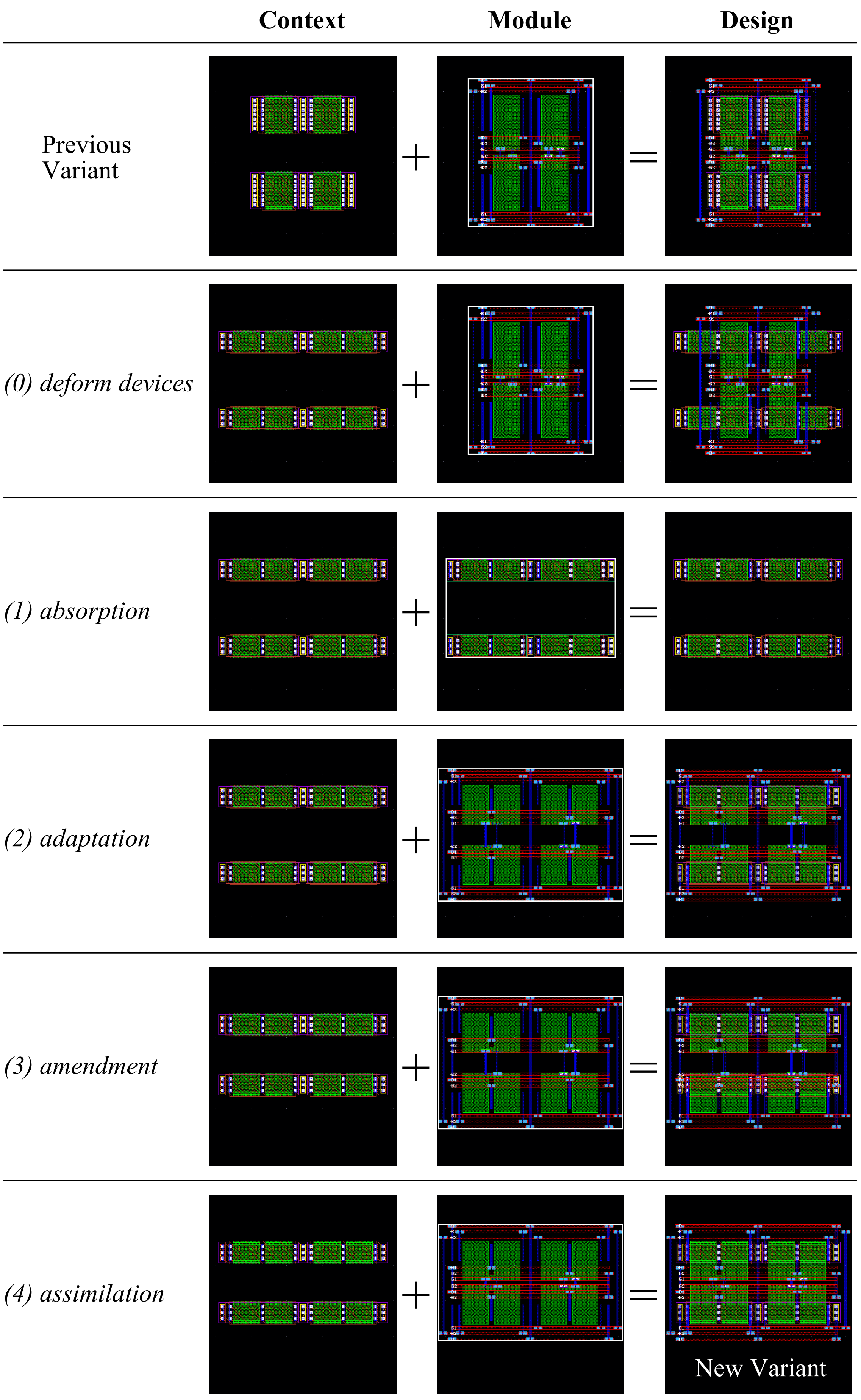


Figure 7.7: Exemplary deformation of a governing module, involving a set of co-transformations.

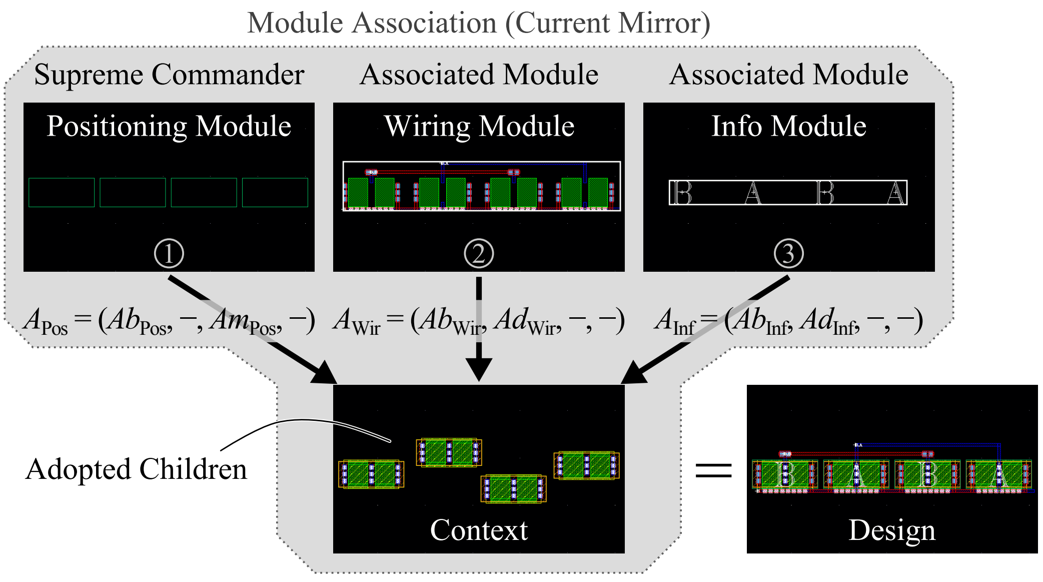

Module Association (Current Mirror)
Supreme Commander
Associated Module
Associated Module
Positioning Module
Wiring Module
Info Module
B A B A
①
②
③
$A_{\text{Pos}} = (Ab_{\text{Pos}}, -, Am_{\text{Pos}}, -)$
$A_{\text{Wir}} = (Ab_{\text{Wir}}, Ad_{\text{Wir}}, -, -)$
$A_{\text{Inf}} = (Ab_{\text{Inf}}, Ad_{\text{Inf}}, -, -)$
Adopted Children
Context
=
Design

Figure 7.8: Example of a Current Mirror module association consisting of three governing modules.

The Current Mirror layout from Figure 7.8 is reprised in Figure 7.9 to illustrate a deformation of the module association. In this example, the deformation of the module association is based on a rotation of its adopted children. Initially (a), the context of the module association is made up of the four aligned transistors from the design of the previous variant. To perform a deformation, the supreme commander –here: the Positioning module– first applies the appropriate co-transformation to its adopted children, which in this case means to rotate each device by 90°. Then (b), the Positioning module adopts the devices anew (via A_{Pos}) to move them closer together and align them. This represents the co-transformation of the supreme commander, who furthermore cares for the co-transformation of its associated modules. That means, the Positioning module triggers their respective adoption process again (A_{Wir} and A_{Inf}), which leads to the new layout variant: a Current Mirror design whose transistors are rotated "upright", as opposed to the previous layout variant of the module association, where the transistor channels run "across" the design. These two transistor orientations will again be picked up in Section 7.2.4.2.

7.2.3.2 Hierarchical Module Associations

A module association may not only be comprised of primitive devices and governing modules, but can also incorporate other module associations. Thus, module associations can be hierarchically imposed onto one another to build larger design components, as Figure 7.10 illustrates with an example. Initially (a), there are two groups of transistors in the design, with seven devices per group. Next (b), two Current Mirrors are created from each group of transistors according to the adoption process known from Figure 7.8. As indicated in column **Module Hierarchy** (Figure 7.10 again), there are now two module associations, each of which contains three governing modules. All three of these governing modules adopt the seven transistors, with the Positioning module representing the supreme commander of its module association. In (c), the two module associations are adopted by the Positioning module of a Symmetric Current Mirror Pair (marked with an asterisk to distinguish it from the other Positioning modules), which has the purpose of keeping two Current Mirrors next to each other side by side.

The amendment operation Am^{*}_{Pos} of this adoption process A^{*}_{Pos} effectuates a movement and a rotation of the two Current Mirrors, which is applied to the two supreme commanders as their respective co-transformation. Each of the supreme commanders then also cares for the corresponding co-transformation of its adopted children and of its associated modules. Since the amendment operation does not include a deformation, the entire set of co-transformations involves only translational moves and rotations (such that –in contrast to Figure 7.9– the transistors on device level need not be adopted once again by the governing modules in the module association). The obtained layout is a symmetric arrangement of the two Current Mirrors as shown in column **Design** of Figure 7.10.

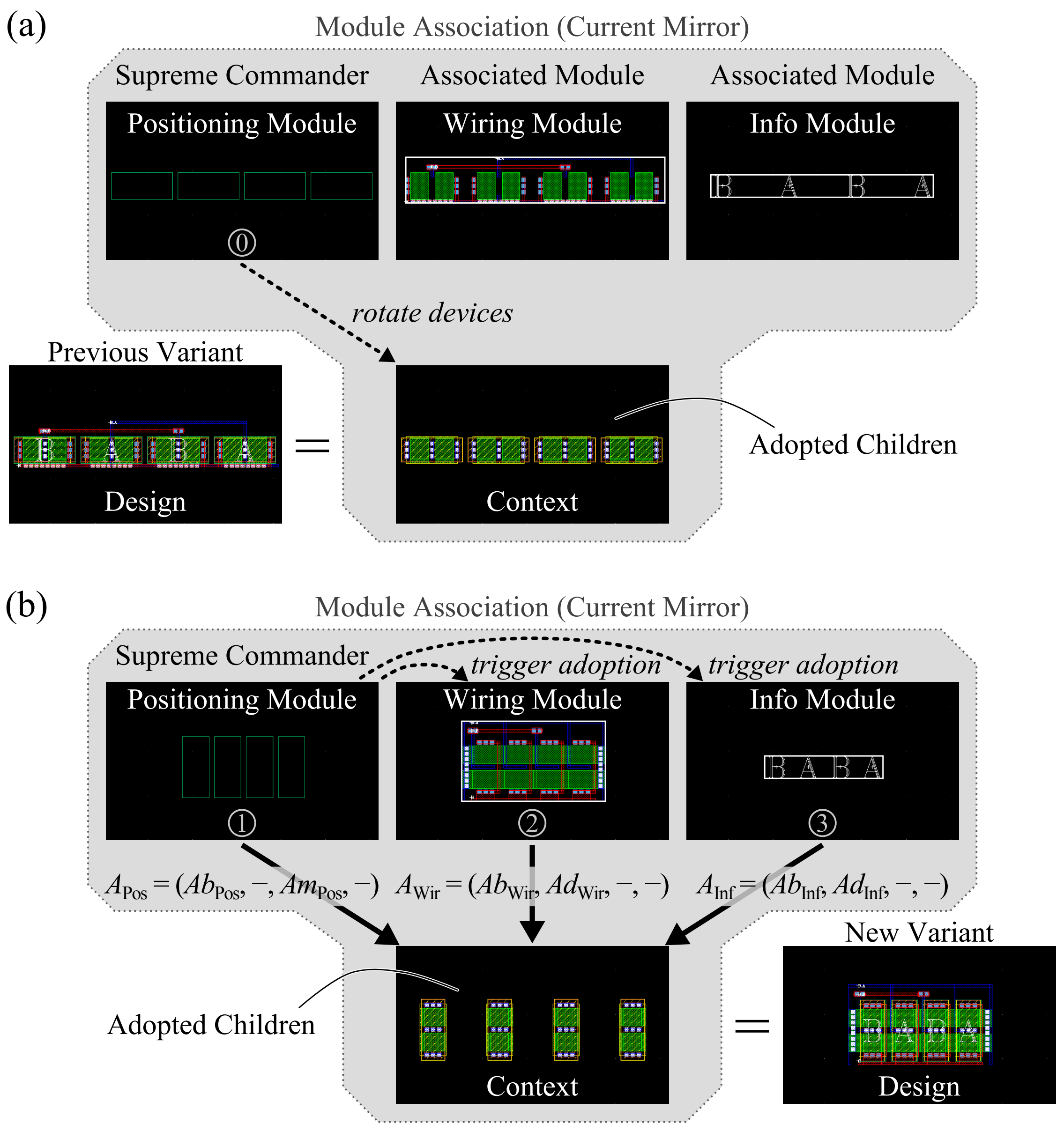


Figure 7.9: Exemplary deformation of a module association via (a) device rotation and (b) re-adoption.

7.2.3.3 Co-transformations in a Module Association

Table 7.2 lists the possible transformations that a module association can undertake by performing a *movement* M, a *rotation* R, and/or a *deformation* D. These transformation elements are subsequently referred to as *morphisms* Φ. Further shown in the table are the involved co-transformations that the adopted children (indexed I), the supreme commander (indexed II), and the associated modules (indexed III) need to undergo. As should be clear from the discussion so far, each of these co-transformations can also consist of different morphisms. To mark the order, in which the individual morphisms are performed, two symbols are introduced. A diamond $\diamond$ indicates that two morphisms are commutative, i.e., a morphism Φ_a can precede or succeed another morphism Φ_b. In contrast, a triangle $\triangleright$ denotes a noncommutative order: $\Phi_a \triangleright \Phi_b$ says that Φ_a cannot succeed but only precede Φ_b.

It it important to note that each rotation R in a co-transformation has the same pivotal point as the rotation of the module association. This is to say that the components of the module association are not rotated around their individual points of origin. Instead, the pivotal point is always the center point of the entire module association. By the same token, that center point is also supposed to be identical before and after a deformation D. Items in square brackets represent an optional morphism, while A denotes a re-adoption for the purpose of deforming the module association. Deformations are also reflected by the

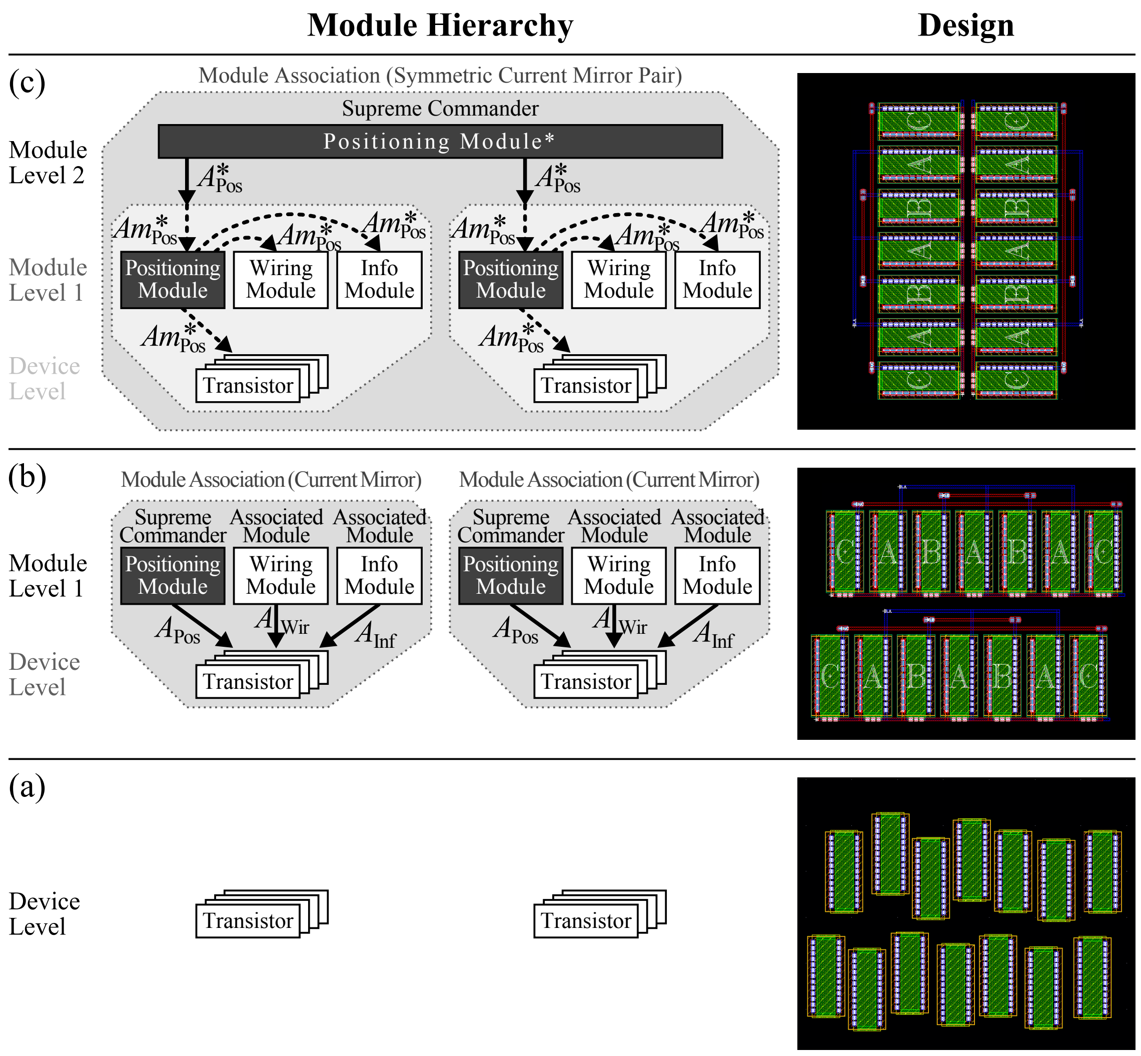


Figure 7.10: Example of a hierarchical module association with (a) 14 transistors on device level, which are adopted by (b) two Current Mirror module associations, which are in turn adopted by a (c) Symmetric Current Mirror Pair.

Table 7.2: Possible transformations of a module association with corresponding co-transformations. Items in square brackets are optional morphisms, while A denotes a re-adoption. A diamond $\diamond$ denotes commutativity, whereas a triangle $\triangleright$ signifies noncommutativity.

Transformation of the Module Association			**Co-transformations**		
Movement	Rotation	Deformation	Adopted Children (I)	Supreme Commander (II)	Associated Modules (III)
M	-	-	$M_I \diamond M_{II} \diamond M_{III}$		
-	R	-	$R_I \diamond R_{II} \diamond R_{III}$		
M	R	-	$(M_I \diamond R_I) \diamond (M_{II} \diamond R_{II}) \diamond (M_{III} \diamond R_{III})$		
-	-	D	$(([R'_I] \diamond [D'_I]) \triangleright M'_I) \triangleright A_{II} \triangleright A_{III}$		
M	-	D	$(M_I \diamond (([R'_I] \diamond [D'_I]) \triangleright M'_I)) \triangleright A_{II} \triangleright A_{III}$		
-	R	D	$(R_I \diamond (([R'_I] \diamond [D'_I]) \triangleright M'_I)) \triangleright (R_{II} \triangleright A_{II}) \triangleright (R_{III} \triangleright A_{III})$		
M	R	D	$(M_I \diamond R_I \diamond (([R'_I] \diamond [D'_I]) \triangleright M'_I)) \triangleright (R_{II} \triangleright A_{II}) \triangleright (R_{III} \triangleright A_{III})$		

appearance of M', R', and D': these are morphisms of the adopted children that serve the deformation of the module association, as was exemplified in Figure 7.9. So, a movement M' of the adopted children is not equal to the movement M of the module association.

At first glance, it might be surprising to see that in some cases a movement M of the module association leads to a corresponding movement M_{II} of the supreme commander and a corresponding movement M_{III} of its associated modules, while in other cases it does not. The latter can be observed in those cases where the transformation of the module association also contains a deformation D. It is in those cases, that the supreme commander and its associated modules perform a re-adoption A_{II} and A_{III} to account for the deformation. And since this re-adoption occurs *after* movement M has already been applied to the children that are to be adopted, the new locations of these children immanently manifest themselves in the supreme commander and the associated modules without the need to perform explicit movements M_{II} and M_{III}.

Using the notation $\Phi_a \succ \Phi_b$ to express that morphism Φ_a is followed by morphism Φ_b, the deformation of the Differential Pair module in Section 7.2.2.2 and the deformation of the Current Mirror module in Section 7.2.3.1 can be formally expressed in the style of Table 7.2. Concerning the first of these two examples, the visualization in Figure 7.7 helps to see which co-transformations are involved in the deformation of the Differential Pair:

$$D'_I \succ M'_I \succ A_{II}. \tag{7.1}$$

Since the Differential Pair is realized with a single governing module, it does not feature any associated modules and therefore the deformation does not involve a re-adoption A_{III}. Regarding the second example, the deformation of the Current Mirror in Figure 7.9 can be expressed as:

$$R'_I \succ M'_I \succ A_{II} \succ A_{III}. \tag{7.2}$$

In both examples, M'_I is in fact performed by the amendment operation of the supreme commander's adoption process, while the important aspect about A_{II} is its assimilation operation. In the latter example though, A_{II} is moot because the supreme commander is a pure Positioning module (i.e., a meta-module). Still, both expressions are in line with the sequence of co-transformations articulated in Table 7.2.

7.2.3.4 Coordinate System Issues

The preceding ruminations reveal that amendments do not only play an important role during the initialization phase of a SWARM run, but whenever a module deforms itself into another layout variant. Now, one should remember that for every amendment, the respective modifications are determined *inside* the module but are applied to its *outside* design context. On that account, one particular issue has to be kept in mind: the internal coordinate system of a module instance is not necessarily equal to the coordinate system of its context (unless the instance's point of origin is $(0, 0)$ and the instance is not rotated). For that reason, some detailed thoughts need to be spent on the adoption process, and –first of all– on the absorption operation therein. As already mentioned in the beginning of Section 7.2.2, the absorption operation requires the type, the parameter values, the location, and the orientation of each external component. The former two items can be neglected for the following discussion, because only the location and the orientation depend on the coordinate system.

The *location* L of a design component is subsequently expressed as a pair of coordinates $L = (x, y)$, where x denotes the horizontal coordinate and y denotes the vertical coordinate of the location. Regarding the *orientation* O of a design component, it must be acknowledged that the component may not only be rotated by a certain angle but that it can also be flipped around a certain axis. As illustrated in Figure 7.11, this results in a total of eight possible orientations each of which can be formally denoted via $O_{\bar{h}} = (\alpha, \bar{h})$. The *angle* α of a component's orientation is to be interpreted counterclockwise and can assume one of four possible values: $\alpha \in \{0°, 90°, 180°, 270°\}$. Flipping can occur horizontally (around a vertical axis) or vertically (around a horizontal axis). As Figure 7.11 shows, the value for $\bar{h}$ indicates *horizontal flipping*, with $\bar{h} \in \{0, 1\}$. Value $\bar{h} = 1$ means that the design component is horizontally flipped, whereas $\bar{h} = 0$ signifies that the component is not flipped. Vertical flipping does not provide additional orientations because any orientation that involves a vertical flip, can also be obtained with an appropriate rotation and a horizontal flip. The equivalence between the *horizontal notation* $O_{\bar{h}} = (\alpha, \bar{h})$ for orientations based on horizontal flipping, and the *vertical notation* $O_v = (\alpha, v)$ for orientations based on *vertical flipping* v, is given in Table 7.3.

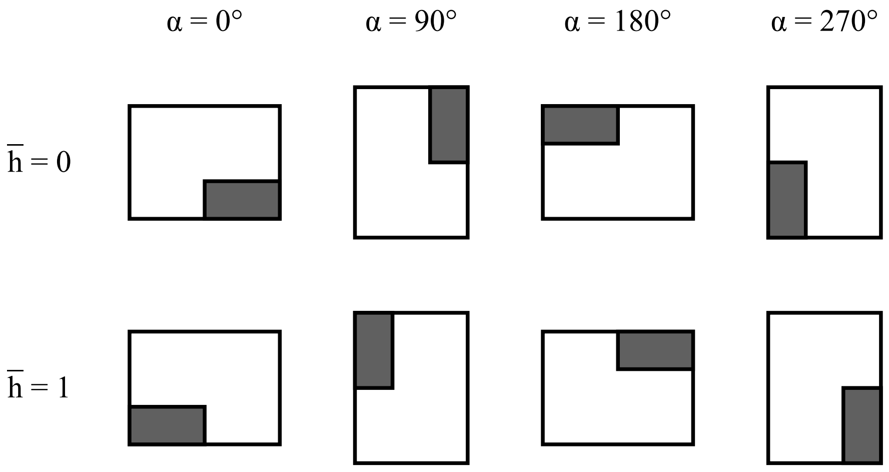


Figure 7.11: Eight orientations of a design component, depending on angle α and horizontal flipping $\bar{h}$.

Table 7.3: Equivalence between horizontal notation and vertical notation for a component's orientation.

Horizontal Notation $O_{\bar{h}} = (\alpha, \bar{h})$	**Vertical Notation** $O_v = (\alpha, v)$
$(0°, 1)$	$(180°, 1)$
$(90°, 1)$	$(270°, 1)$
$(180°, 1)$	$(0°, 1)$
$(270°, 1)$	$(90°, 1)$

In the initialization phase of a SWARM run, the absorption operation of a module instance works straightforward because the module can be instantiated in default orientation, i.e., with $O_{\bar{h}} = (0°, 0)$. For every component that is to be absorbed, the following location coordinates x_i and y_i, as well as the orientation specifiers α_i and $\bar{h}_i$ must be passed to the module instance for internal context duplication:

$$x_i = x_A - x_P, \tag{7.3a}$$
$$y_i = y_A - y_P, \tag{7.3b}$$
$$\alpha_i = \alpha_A, \tag{7.3c}$$
$$\bar{h}_i = \bar{h}_A, \tag{7.3d}$$

where x_A and y_A are the coordinates of the component to Absorb, while x_P and y_P are the coordinates of the Procedural module instance, with α_A and $\bar{h}_A$ representing the orientation specifiers of the component to Absorb. During the self-organization phase of a SWARM run, the orientation of a module instance can become $O_{\bar{h}} \neq (0°, 0)$, which must be taken into account by the absorption operation. This is achieved by transforming the location coordinates and the orientation specifiers converse to the orientation of the module instance, as will now be explained in greater detail.

Regarding the location coordinates, these need to be rotated by $-\alpha_P$ in order to compensate the angle α_P in the orientation of the module instance. From planar trigonometry, the rotation of a point (x, y) around an angle α can be calculated via:

$$x' = x \cdot \cos(\alpha) - y \cdot \sin(\alpha), \tag{7.4a}$$
$$y' = x \cdot \sin(\alpha) + y \cdot \cos(\alpha). \tag{7.4b}$$

For the absorption operation, this rotation must proceed around the origin of the module instance. This means, that $x = x_A - x_P$ and $y = y_A - y_P$. To take flipping into account, the location coordinates must first be flipped in reverse before the rotation is applied. In this thesis, the possible orientations of a design component (see Figure 7.11) are expressed via the notation $(\alpha, \bar{h})$, which means that the component is *first* rotated by α and *then* horizontally flipped in case $\bar{h} = 1$. Concerning the absorption operation, this is why the reverse flipping of the location coordinates must be performed *prior* to the reverse rotation.[3] This can be conveniently achieved with the term $1 - 2\bar{h}_P$, which evaluates either to 1 (in case $\bar{h}_P = 0$) or to -1 (in case $\bar{h}_P = 1$) and can thus be used to negate (or not negate) the horizontal coordinate $x_A - x_P$. The full formulas for the calculation of x_i and y_i are shown in equation 7.6.

Regarding the orientation specifiers $(\alpha_A, \bar{h}_A)$ of the component to absorb, these must –like the location coordinates– also be "reversed" to equalize the orientation of the module instance. For each possible component orientation $(\alpha_A, \bar{h}_A)$ and for all possible module orientations $(\alpha_P, \bar{h}_P)$, Table 7.4 lists the reversed component orientation $(\alpha_i, \bar{h}_i)$ that must be passed to the module instance for internal context duplication.

Table 7.4: Reversed orientation of a component that is to be absorbed by a rotated governing module.

	reversed component orientation $(\alpha_i, \bar{h}_i)$ **for all possible module orientations** $(\alpha_P, \bar{h}_P)$							
$(\alpha_A, \bar{h}_A)$	$(0°, 0)$	$(90°, 0)$	$(180°, 0)$	$(270°, 0)$	$(0°, 1)$	$(90°, 1)$	$(180°, 1)$	$(270°, 1)$
$(0°, 0)$	$(0°, 0)$	$(270°, 0)$	$(180°, 0)$	$(90°, 0)$	$(0°, 1)$	$(90°, 1)$	$(180°, 1)$	$(270°, 1)$
$(90°, 0)$	$(90°, 0)$	$(0°, 0)$	$(270°, 0)$	$(180°, 0)$	$(90°, 1)$	$(180°, 1)$	$(270°, 1)$	$(0°, 1)$
$(180°, 0)$	$(180°, 0)$	$(90°, 0)$	$(0°, 0)$	$(270°, 0)$	$(180°, 1)$	$(270°, 1)$	$(0°, 1)$	$(90°, 1)$
$(270°, 0)$	$(270°, 0)$	$(180°, 0)$	$(90°, 0)$	$(0°, 0)$	$(270°, 1)$	$(0°, 1)$	$(90°, 1)$	$(180°, 1)$
$(0°, 1)$	$(0°, 1)$	$(90°, 1)$	$(180°, 1)$	$(270°, 1)$	$(0°, 0)$	$(270°, 0)$	$(180°, 0)$	$(90°, 0)$
$(90°, 1)$	$(90°, 1)$	$(180°, 1)$	$(270°, 1)$	$(0°, 1)$	$(90°, 0)$	$(0°, 0)$	$(270°, 0)$	$(180°, 0)$
$(180°, 1)$	$(180°, 1)$	$(270°, 1)$	$(0°, 1)$	$(90°, 1)$	$(180°, 0)$	$(90°, 0)$	$(0°, 0)$	$(270°, 0)$
$(270°, 1)$	$(270°, 1)$	$(0°, 1)$	$(90°, 1)$	$(180°, 1)$	$(270°, 0)$	$(180°, 0)$	$(90°, 0)$	$(0°, 0)$

As can be seen in that table, the component to absorb has to be rotated into a flipped orientation if and only if either the component is in a flipped orientation or if the module instance is in a flipped orientation. This relation is an exclusive or (XOR), which in Boolean logic means that $\bar{h}_i = \bar{h}_A \oplus \bar{h}_P$. In arithmetic terms, this can be expressed as:

$$\bar{h}_i = 1 - |\bar{h}_A + \bar{h}_P - 1|. \tag{7.5}$$

Equivalent to the reverse rotation of the component location (x_A, y_A) (see above), the angle α_A of the component has to be rotated by $-\alpha_P$. Intuitively, this is expected to be achievable by simply calculating $\alpha_i = \alpha_A - \alpha_P$. But, as Table 7.4 shows (see top-left and bottom-right quadrants), this is only true in case $\bar{h}_i = 0$. If $\bar{h}_i = 1$ (see top-right and bottom-left quadrants, marked gray) then the reversed angle must be calculated as $\alpha_i = \alpha_A + \alpha_P$. The reason for this subtlety is illustrated in Figure 7.12: a positive rotation of a component that is in a regular (i.e., not flipped) orientation, entails an *increase* of the component's angle (a), but rotating a component in flipped orientation causes a *decrease* of the angle (b). The correct sign can be included by putting a factor before α_P. That factor is calculated as $2\bar{h}_i - 1$, which evaluates to $1 - 2 \cdot |\bar{h}_A + \bar{h}_P - 1|$.

Putting it all together, the advanced expressions for calculating the location coordinates and orientation specifiers from equation 7.3 can now be written in closed form as follows:

$$x_i = (1 - 2\bar{h}_P) \cdot (x_A - x_P) \cdot \cos(-\alpha_P) - (y_A - y_P) \cdot \sin(-\alpha_P), \tag{7.6a}$$

$$y_i = (1 - 2\bar{h}_P) \cdot (x_A - x_P) \cdot \sin(-\alpha_P) + (y_A - y_P) \cdot \cos(-\alpha_P), \tag{7.6b}$$

$$\alpha_i = \alpha_A + (1 - 2 \cdot |\bar{h}_A + \bar{h}_P - 1|) \cdot \alpha_P, \tag{7.6c}$$

$$\bar{h}_i = 1 - |\bar{h}_A + \bar{h}_P - 1|. \tag{7.6d}$$

[3]The order "first rotate, then flip" in the notation $O_{\bar{h}} = (\alpha, \bar{h})$ is just a question of definition. It could just as well be defined vice-versa, for both horizontal flipping and vertical flipping. However, it is important that this order is respected by the absorption operation.

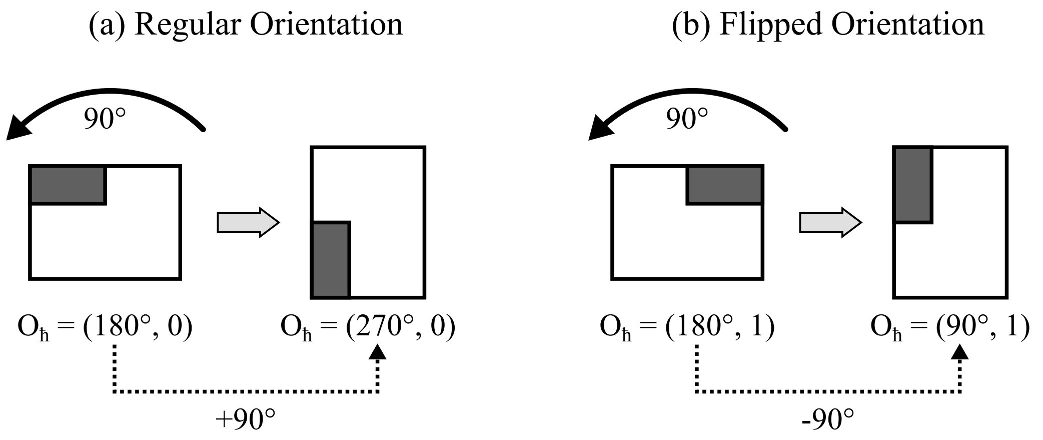


Figure 7.12: Angle change of a rotated component in (a) regular orientation and (b) flipped orientation.

With these formulas, the expressions from equation 7.3 become obsolete, because the trivial case that the module instance is in default orientation $(0°, 0)$ is of course also covered by the calculations in equation 7.6: setting $\alpha_P = 0°$ and $\bar{h}_P = 0$ in equation 7.6 leads to the expressions given in equation 7.3. Thus, the formulas of equation 7.6 are valid for a module's adoption process in the initialization phase of a SWARM run, and also when the module rotates and deforms itself during the self-organization phase. Pertaining to the initial remark of this Section 7.2.3.4, the coordinate system issues that have been discussed above with regard to *reading* a module's design context, must equivalently be taken into account when the module is *modifying* its design context.

7.2.4 Layout Variability

With the ability to deform itself into a different layout variant, a governing module can cover various aspect ratios without altering its nominal electrical behavior. For that purpose, a governing module must –in addition to the layout operations discussed in Section 7.2.2 and Section 7.2.3– first of all be able to *tell* the respective degrees of freedom for the circuit it implements. These degrees of freedom define the *variability* $\mathcal{V}$ of the module, i.e., the set of all feasible layout variants that the module can assume. Depending on the degrees of freedom, a distinction can be drawn between *discrete variability*, where the set of assumable layout variants is limited, and *full variability*, where the module is able to change its dimensions in a continuous range.

SWARM supports both kinds of variability. However, if a module features full variability, possible deformations cannot be set out in advance but are determined ad hoc during the module interaction. For that reason, the topic of full variability will be deferred to Section 7.3. Thus, the subsequent discussion concentrates on discrete variability, which is the prevalent kind of variability on the level of *simple modules*.

7.2.4.1 Intrinsic Variability

Every design component has an *intrinsic variability* $\acute{\mathcal{V}}$. For example, in the case of a parameterized MOS transistor (referred to as type `MT`), the degree of freedom for potential deformations is its *number of fingers* (see Figure 2.1), here denoted as `f`. Formally, the possible values for `f` are given by $\mathbb{D}_{\mathtt{f}}$, which represents the *domain* of that parameter. In the particular case of `f`, which is defined as an integer parameter, this domain is in general a subset of the set $\mathbb{N}^+$ of positive natural numbers: $\mathbb{D}_{\mathtt{f}} \subset \mathbb{N}^+$. For a particular instance *MT* of the transistor, the domain is $\mathbb{D}_{MT,\,\mathtt{f}} = \{1, 2, \ldots, \mathfrak{f}_{\max}\}$, where the parameter value $\mathfrak{f}_{\max}$ for the maximum number of fingers depends on the transistor's total channel width.

Simply put, $\mathbb{D}_{MT,\,\mathtt{f}}$ defines the intrinsic variability $\acute{\mathcal{V}}_{MT}$ of the transistor instance. More formally, the intrinsic variability can be expressed as a function $g_{\mathtt{MT}}$ of $\mathbb{D}_{MT,\,\mathtt{f}}$ where $g_{\mathtt{MT}}$ represents the behavior[4] of

[4]The behavior $g_{\mathtt{MT}}$ corresponds to the behavior g mentioned in Section 3.1.2.1, which is an "introversive" generator behavior and should not be mistaken for the "extroversive" interaction behavior that will be subject to Section 7.3.

the transistor generator:

$$\acute{\mathcal{V}}_{MT} = g_{\mathtt{MT}}\big(\mathbb{D}_{MT,\,\mathtt{f}}\big). \tag{7.7}$$

As an alternative notation, the intrinsic variability is henceforth written as a mapping from the domain of a parameter to the set of corresponding layout variants. In the case of the MOS transistor instance, the mapping is thus written as:

$$\acute{\mathcal{V}}_{MT} \Leftarrow \mathbb{D}_{MT,\,\mathtt{f}}. \tag{7.8}$$

The use of this notation is encouraged due to its compactness. This becomes even more evident when multiple parameters are involved, as the following Section 7.2.4.2 is about to show.

7.2.4.2 Cumulative Variability

If transistors are adopted by a governing module, the intrinsic variability of the transistors contributes to the variability of the module. An example is the Differential Pair module in Figure 7.7, which deforms itself from a 1-finger variant into a 2-finger variant by incrementing the number of fingers in each of its transistors. However, a module may further provide its own intrinsic variability. Regarding the deformation in Figure 7.9, the Current Mirror module association deforms itself into a different variant without modifying the layout of its adopted transistors, but simply by rotating them. So, the orientation of these transistors, a parameter here denoted as $\mathtt{o}$ is one degree of freedom of the Current Mirror $\mathtt{CM}$ with the domain $\mathbb{D}_{CM,\,\mathtt{o}} = \{across, upright\}$. Another degree of freedom is given by the possibility to break the positioning from the depicted single-row variant into a dual-row variant. Hence, the domain of this positioning parameter $\mathtt{p}$ is $\mathbb{D}_{CM,\,\mathtt{p}} = \{single, dual\}$. With these degrees of freedom, the intrinsic variability $\acute{\mathcal{V}}_{CM}$ of a Current Mirror instance *CM* is defined by the Cartesian product $\mathbb{D}_{CM,\,\mathtt{o}} \times \mathbb{D}_{CM,\,\mathtt{p}}$.

Apart from this, the Current Mirror can also change the number of fingers in its adopted transistors. Thus, the total variability of the module instance, referred to as its *cumulative variability* $\widetilde{\mathcal{V}}$, is given by $\mathbb{D}_{CM,\,\mathtt{o}} \times \mathbb{D}_{CM,\,\mathtt{p}} \times \mathbb{D}_{MT,\,\mathtt{f}}$. In general terms, the *cumulative variability* $\widetilde{\mathcal{V}}_P$ of a Parameterized design component P is the product of its own *intrinsic variability* $\acute{\mathcal{V}}_P$ and the *cumulative variability* $\widetilde{\mathcal{V}}_A$ of its Adopted children:

$$\widetilde{\mathcal{V}}_P = \acute{\mathcal{V}}_P \times \widetilde{\mathcal{V}}_A. \tag{7.9}$$

Regarding the Current Mirror module above, please note that $\widetilde{\mathcal{V}}_A = \acute{\mathcal{V}}_A$ because the cumulative variability of a transistor is identical to its intrinsic variability: $\widetilde{\mathcal{V}}_{MT} = \acute{\mathcal{V}}_{MT}$. This equation holds true for all primitive devices (since they do not have a subhierarchy) and allows to calculate a module's cumulative variability from the bottom up. On that basis, inserting $\widetilde{\mathcal{V}}_P$ for $\widetilde{\mathcal{V}}_A$ on the next higher module level is the key to determine the cumulative variability of a hierarchical module association. For example, the two Current Mirrors in a Symmetric Current Mirror Pair module $\mathtt{SCMP}$ (as in Section 7.2.3) can either be positioned in their default orientation (see Figure 7.10 (b)) or be rotated (as done in Figure 7.10 (c)). Thus, comparable to the transistors in a Current Mirror module, this orientation $\mathtt{o}$ of the Current Mirrors in a Symmetric Current Mirror Pair is a degree of freedom with $\mathbb{D}_{SCMP,\,\mathtt{o}} = \{across, upright\}$, which defines the intrinsic variability $\acute{\mathcal{V}}_{SCMP}$. Then, the cumulative variability $\widetilde{\mathcal{V}}_{SCMP}$ of an instance *SCMP* evaluates to

$$\widetilde{\mathcal{V}}_{SCMP} \tag{7.10}$$

$$= \overbrace{\acute{\mathcal{V}}_{SCMP} \times \quad \widetilde{\mathcal{V}}_{CM}} \tag{7.11}$$

$$= \acute{\mathcal{V}}_{SCMP} \times \overbrace{\acute{\mathcal{V}}_{CM} \times \widetilde{\mathcal{V}}_{MT}} \tag{7.12}$$

$$= \acute{\mathcal{V}}_{SCMP} \times \acute{\mathcal{V}}_{CM} \times \overbrace{\acute{\mathcal{V}}_{MT}} \tag{7.13}$$

and can be obtained from

$$\widetilde{\mathcal{V}}_{SCMP} = \overbrace{g_{\mathtt{SCMP}}\big(\mathbb{D}_{SCMP,\,\mathtt{o}}\big)}^{\acute{\mathcal{V}}_{SCMP}} \times \overbrace{g_{\mathtt{CM}}\big(\mathbb{D}_{CM,\,\mathtt{o}}\big) \times g_{\mathtt{CM}}\big(\mathbb{D}_{CM,\,\mathtt{p}}\big)}^{\acute{\mathcal{V}}_{CM}} \times \overbrace{g_{\mathtt{MT}}\big(\mathbb{D}_{MT,\,\mathtt{f}}\big)}^{\acute{\mathcal{V}}_{MT}} \tag{7.14}$$

which can be expressed more compactly using the notation introduced in equation 7.8:

$$\widetilde{\mathcal{V}}_{SCMP} \Leftarrow \mathbb{D}_{SCMP,\,\mathtt{o}} \times \mathbb{D}_{CM,\,\mathtt{o}} \times \mathbb{D}_{CM,\,\mathtt{p}} \times \mathbb{D}_{MT,\,\mathtt{f}}. \tag{7.15}$$

Based on this calculation principle, all governing modules and module associations are able to determine the total variability that they may exploit to perform deformations. This is a crucial qualification for Section 7.3, because the more exhaustive the variability of a module is, the more choices of action the module will have during a self-organization run.

7.2.4.3 Variability of Primitive Devices

Talking about the self-organization in SWARM, one remark should be made about primitive devices. Primitive devices are *native* devices that come with the semiconductor technology as part of the PDK. Thus, they are not responsive and are therefore not able to perform actions by themselves, nor are they meant to be capable of answering questions about their variability. To remedy these deficiencies, a *variator module* can be employed. A variator module is a meta-module with the sole purpose to adopt and manage a single, primitive device instance. To perform deformations, it is the variator module's job to determine the variability of that instance.

As should become apparent from the discussion above, the variability of a primitive device is (in contrast to that of the modules introduced so far) usually instance-specific.[5] In the case of a MOS transistor, as already mentioned in Section 7.2.4.1, the variability is defined by $\acute{\mathcal{V}}_{MT} \Leftarrow \mathbb{D}_{MT,\,\mathtt{f}}$ with

$$\mathbb{D}_{MT,\,\mathtt{f}} = \{1, 2, \ldots, \mathfrak{f}_{\max}\} = \{1, 2, \ldots, \lfloor \tfrac{\mathfrak{w}_{\text{ch}}}{\mathfrak{w}_{\min}} \rfloor\}, \tag{7.16}$$

where $\mathfrak{w}_{\text{ch}}$ is the total channel width of the transistor instance and $\mathfrak{w}_{\min}$ is the minimum channel width (and thus also the minimum finger width) for that type of transistor.

7.2.4.4 Variability of Simple Modules

To explicate the variability of common, analog basic circuits that belong to the level of *simple modules*, it is apposite to expose which of these circuits have been implemented in SWARM, and what components these circuits are made up of in this implementation.

Showing device identifiers as done by the Current Mirror in Figure 7.8 is a largely circuit-independent thing to do. Similarly, the style of device positioning is quite comparable among several circuits, and even the wiring in these circuits shares a lot of common ground. For that reason, it is feasible to address these three tasks by implementing three generic governing modules, as seen in the bottom rows of Table 7.5. The Info module `I` is so generic, that it can be employed as-is for every basic circuit which allows for custom device interdigitation. The Positioning module `P` and the Wiring module `W` provide parameters for switching between slightly different positioning and wiring behaviors in order to cover several circuit types as well. Thus, the Wiring module `W` and the Wiring modules $\mathtt{W}_{\text{CM}}$, $\mathtt{W}_{\text{CS}}$, and $\mathtt{W}_{\text{WS}}$ (which target different types of Current Mirror circuits) are implemented as one single procedural generator, providing a parameter to choose the respective topology. The situation is the same with the Positioning module `P` and its offshoots $\mathtt{P}_{\text{BK}}$ (which implements a so-called current mirror Bank), $\mathtt{P}_{\text{CS}}$ (which represents a Cascode module), and $\mathtt{P}_{\text{SP}}$ (which realizes a Symmetric Pair module). The Differential Pair module $\mathtt{PW}_{\text{DP}}$ is a panfunctional module since it performs both positioning and wiring.

Table 7.6 lists the analog basic circuits that have been implemented in SWARM utilizing the governing modules presented in Table 7.5. Also included is the MOS Transistor primitive device, as well as two kinds of circuits that have not been discussed so far: the Cascode Current Mirror `CCM` and the Wide-swing Current Mirror `WCM`. Layout examples of these circuits are given by the module instances in Figure 7.13 and Figure 7.14 respectively.

For every kind of circuit listed in Table 7.6, column **Components** shows which types of components (governing modules and primitive devices) the circuit has been realized with. Also given for each circuit

[5]However, it may also be the case with a module that it has to concede an instance-specific curtailment in its variability, e.g., to satisfy a certain design constraint.

Table 7.5: Generic governing modules and their topological offshoots, as implemented in SWARM.

Module	**Purpose**	**Comment**	**Based on**
$\mathtt{W}_{\mathrm{WS}}$	wiring	for Wide-swing Current Mirror	`W`
$\mathtt{W}_{\mathrm{CS}}$	wiring	for Cascode Current Mirror	`W`
$\mathtt{W}_{\mathrm{CM}}$	wiring	for Current Mirror	`W`
$\mathtt{P}_{\mathrm{SP}}$	positioning	Symmetric Pair module	`P`
$\mathtt{P}_{\mathrm{CS}}$	positioning	Cascode module	`P`
$\mathtt{P}_{\mathrm{BK}}$	positioning	Bank module	`P`
$\mathtt{PW}_{\mathrm{DP}}$	positioning and wiring	for Differential Pair	–
`I`	show interdigitation	generic Info module	–
`W`	wiring	generic Wiring module	–
`P`	positioning	generic Positioning module	–

Table 7.6: Analog basic circuits covered by SWARM, based on its implemented governing modules.

Circuit	**Abbr.**	**Components**	**Ductility**	**Examples**
Symmetric Current Mirror Pair	`SCMP`	$\mathtt{CM} + \mathtt{P}_{\mathrm{SP}}$	$8 \cdot \mathfrak{f}_{max}$	Figure 7.10
Wide-swing Current Mirror	`WCM`	$\mathtt{MT} + \mathtt{P}_{\mathrm{BK}} + \mathtt{P}_{\mathrm{CS}} + \mathtt{W}_{\mathrm{WS}}$	$2 \cdot \mathfrak{f}_{max}$	Figure 7.14
Cascode Current Mirror	`CCM`	$\mathtt{MT} + \mathtt{P}_{\mathrm{BK}} + \mathtt{P}_{\mathrm{CS}} + \mathtt{W}_{\mathrm{CS}}$	$2 \cdot \mathfrak{f}_{max}$	Figure 7.13
Current Mirror	`CM`	$\mathtt{MT} + \mathtt{P}_{\mathrm{BK}} + \mathtt{W}_{\mathrm{CM}}$	$4 \cdot \mathfrak{f}_{max}$	Figure 7.9
Differential Pair	`DP`	$\mathtt{MT} + \mathtt{PW}_{\mathrm{DP}}$	$2 \cdot \mathfrak{f}_{max}$	Figure 7.7
MOS Transistor	`MT`	–	$\mathfrak{f}_{max}$	Figure 2.1

is the total number of its possible deformation variants, denoted as its *ductility* d. For a circuit realized as a module (or module association) P, the module's ductility d_P is simply the mathematical cardinality of the module's cumulative variability:

$$d_P = |\widetilde{\mathcal{V}}_P|. \tag{7.17}$$

If $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ is a set denoting the types of components that a circuit is realized with, and if (for every P in $\mathcal{P}$) the set $\mathcal{I}_P = \{I_{P,1}, I_{P,2}, \ldots, I_{P,n}\}$ denotes the input parameters that can be varied as degrees of freedom to perform deformations, then the ductility d can be formally calculated as the product

$$d = \prod_{i=1}^{k} \prod_{j=1}^{n} |\mathbb{D}_{P_i, I_{P_i,j}}|. \tag{7.18}$$

However, one should take note that in some cases the ductility may actually be smaller than what equation 7.18 suggests. This situation occurs either (1) if the variability of a module is reduced for layout reasons, or (2) when there is a dependence between at least two degrees of freedom. Both aspects can be observed in the `CCM` circuit and the `WCM` circuit (as is reflected by column **Ductility** in Table 7.6):

- Aspect (1) is showcased by the practice to place all transistors in each of these circuits in an upright orientation only. From a layout perspective, this is quite convenient because it allows to connect the transistor gates in a straightforward way just by drawing stripes of poly across the transistors. The evident downside is –of course– the reduction in variability that stems from this confinement.
- Concerning aspect (2), two such dependencies can be encountered in both the `CCM` circuit and the `WCM` circuit. The first dependence pertains to the intrinsic variability of the Bank module $\mathtt{P}_{\mathrm{BK}}$ and the Cascode module $\mathtt{P}_{\mathrm{CS}}$: if, and only if, the Bank module breaks its positioning from a single-row layout into a dual-row variant, then the Cascode module must do so as well (such a deformation is depicted in Figure 7.13). So, the combined variability in these degrees of freedom $\mathtt{p}_{BK}$ and $\mathtt{p}_{CS}$ is not the Cartesian product $\mathbb{D}_{BK,\mathtt{p}} \times \mathbb{D}_{CS,\mathtt{p}}$ but only the subset $\{(single, single), (dual, dual)\}$. The second dependence can be found in the cumulative variability of $\mathtt{P}_{\mathrm{BK}}$ and $\mathtt{P}_{\mathrm{CS}}$: the number

of fingers is supposed to be equal among the transistors of both modules. The deformation in Figure 7.14 illustrates how both modules change from a 2-finger variant into a 3-finger variant. Setting the number of fingers to different values is not only disadvantageous in regard of matching, but also leads to a significant amount of dead space, since the total channel width of the Bank transistors is usually identical to that of the Cascode transistors. Therefore, keeping the number of fingers identical represents a constraint that is implicitly considered by SWARM's modules (which comes up to the motivation of this thesis).

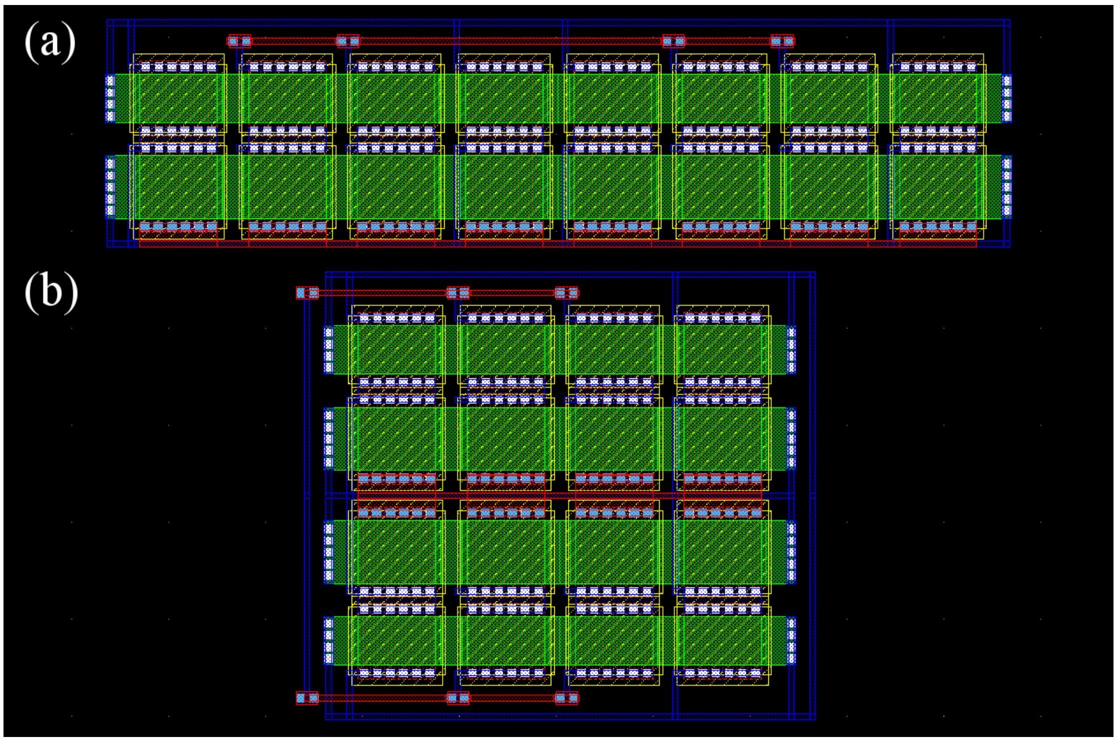


Figure 7.13: Cascode Current Mirror deformation from (a) single-row variant into (b) dual-row variant.

For each of the circuits in Table 7.6, column **Examples** as a summary again refers to those figures in this thesis which illustrate the respective module layout and the module's layout variability. Regarding layout variability, the thoughts in this work are supposed to be universally valid, independent of the semiconductor technology at hand. In contrast, the layout details of the implemented modules are not strictly part of the SWARM methodology itself, but always depend on the chosen semiconductor technology, the intended application, and other factors. Thus, the concrete module implementation in practice always lies under the authority of the respective design team (i.e., the **Design Expert** in Figure 7.2). But although the implementations in this thesis can be regarded as being only exemplary, they demonstrate how governing modules and module associations allow to automate the layout creation of basic circuits on the level of *simple modules*. At higher levels, additional and more groundbreaking automation concepts need to be embraced. One such automation concept is the module interaction that will now be described in Section 7.3 and represents the second core concept of SWARM.

7.3 Module Interaction

To form the desired layout block, the governing modules and module associations from Section 7.2 are to be arranged in a constellation that fits within a user-defined zone and explicitly satisfies all design constraints that are not yet implicitly covered by the modules themselves. At this level, irregular (non-matrix) constellations and arbitrary aspect ratios of the zone outline need to be served. Based on the considerations of Section 7.2.4, the modules may provide sufficient variability to achieve that goal, but the enormous amount of possible variations and constellations raises a combinatorial challenge. Instead

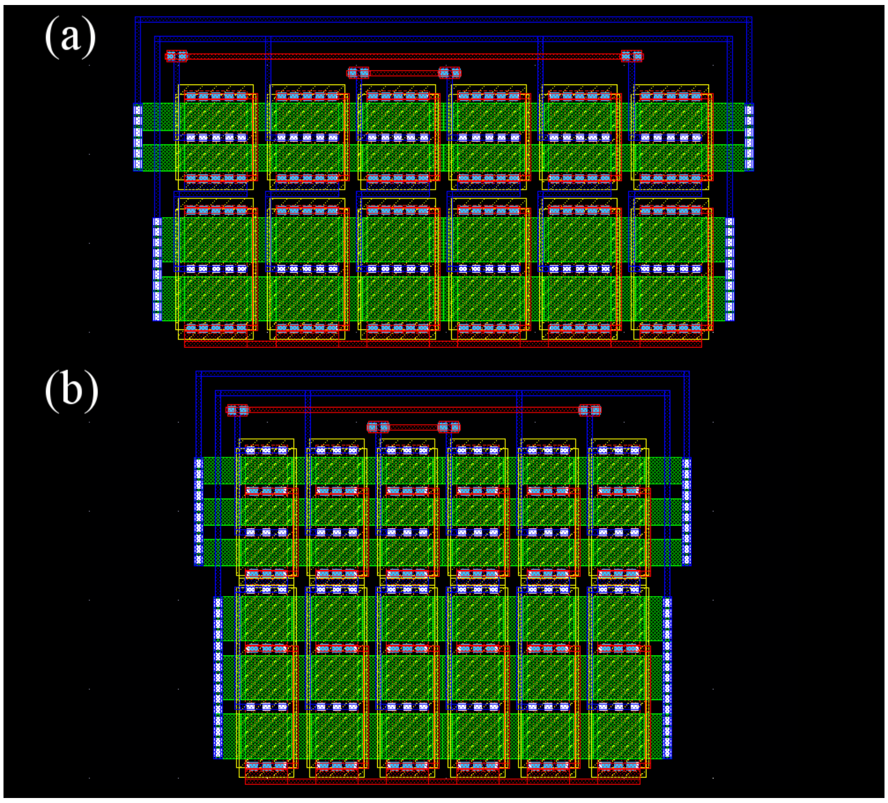


Figure 7.14: Wide-swing Current Mirror deformation from (a) 2-finger variant into (b) 3-finger variant.

of exploring this immense solution space in a top-down manner, the modules are –as outlined in Section 7.1– impelled into a flow of *module interaction* to let them find a suitable arrangement on their own.

Following the idea of decentralization, the modules in SWARM interact upon a maxim of self-determination: in a selfish pursuit of its own personal well-being, each participating module repeatedly chooses its individual course itself, always based on some simple utilitaristic rules and just a local assessment of its current situation. For that purpose, the modules' layout-generating abilites, which can be considered their *introversive behavior* (see Section 7.2.4.1), are extended by an *extroversive behavior*, which dictates a module's reaction to changes of its environment. The extroversive behavior is not necessarily identical among all modules (which are now, in the context of interaction, again referred to as *participants*), but it always abides by a common *action scheme* comprised of the following four *measures*:

(1) assessing the participant's *condition* (see Section 7.3.1),
(2) perceiving its *free peripheral space* (see Section 7.3.2),
(3) exploring and evaluating possible *actions* (see Section 7.3.3),
(4) executing the *preferred* action or staying idle (see Section 7.3.4).

These four measures are exemplarily illustrated in Figure 7.15. Considering a constellation of six participants as in (1), assume that it is participant P's turn to take an action. Following the action scheme, P begins by assessing its condition. Since P detects interference with another participant, P strives for an action that improves its condition. To do so, P perceives the vacant area around it, because most of the possible actions are based on this so-called free peripheral space. As shown in (2), SWARM determines the free peripheral space by extending each of P's four edges it its respective direction until another participant or the zone outline is encountered. Then, as indicated in (3), P explores and evaluates all actions available in the current situation. Some of these actions only affect the participant itself, while the other actions also involve other participants. Next, the actions are compared such that the one which improves P's condition the most can be chosen and executed. In (4), the executed action is even synergistic (see Section 6.4.6): P trades places with another participant and *both* get rid of their interference.

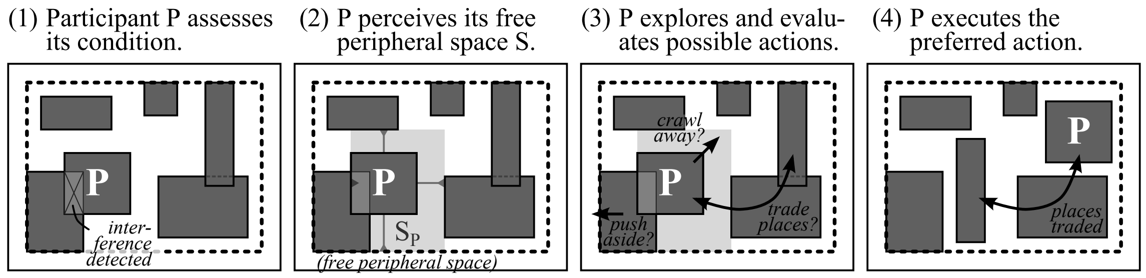

(1) Participant P assesses its condition.
(2) P perceives its free peripheral space S.
(3) P explores and evaluates possible actions.
(4) P executes the preferred action.
P
inter-ference detected
P
S_P
(free peripheral space)
crawl away?
P
push aside?
trade places?
P
places traded

Figure 7.15: A participant's actions follow a common action scheme consisting of four measures.

In the remainder of this section, every single measure of the action scheme will now be covered in a subsection of its own (see Section 7.3.1, Section 7.3.2, Section 7.3.3, and Section 7.3.4, respectively).

7.3.1 Assessment of the Participant's Condition

In terms of game theory (see Section 6.5.2), SWARM can be considered an infinitely-repeated, non-cooperative, discrete, asymmetric, non-constant-sum, sequential, perfect-information game in extensive form, with an unknown number of stage games. In each stage game (here: a *round* of interaction), every participant acts in a self-interested way to improve its personal situation. This *utility-theoretic* attitude is a characteristic trait of noncooperative game theory, but unlike typical *utility functions* which map a player's preferences to some real numbers (thus quantifying the player's favored "states of the world" [281]), SWARM implements a more sophisticated decision-making model built around a participant's *condition*.

The condition of a participant P decides whether there is a need to take action or not. Action is provoked when the condition sustains negative influence, which can be exerted by five different *influencing factors*. These influencing factors are largely based on the fact that all participants are geometric objects with a rectangular bounding box and thus have an area, in contrast to dot-like entities found in some other systems.

The two major influencing factors are denoted as *interference* (repulsion of participants due to overlaps, as already indicated in Figure 7.15) and *turmoil* (attraction of participants due to connectivity), which *can* –depending on their magnitude– provoke an action. In contrast, a participant is *forced* to take an action if it suffers at least one of the other three influencing factors: *protrusion* (overhang beyond the zone outline), *wounds* (regions on P which repeatedly overlapped with other participants in previous rounds of interaction), or *noncompliance* (violation of constraints).[6]

For each kind of influencing factor, every participant has a particular *desire* to reduce and eliminate the factor's negative influence on its condition. This will be subsequently discussed in detail for each individual kind of influencing factor.

7.3.1.1 Interference

Interference is when a participant overlaps one or multiple other participants, as illustrated in Figure 7.16. The interference (written Υ) of a participant P is a sum of *troubles* τ between P and n overlapping participants $P_1, P_2, \ldots, P_n$:

$$\Upsilon_P = \sum_{i=1}^{n} \tau_i. \tag{7.19}$$

Therein, each individual trouble τ_i is defined as

$$\tau_i = \omega_i + \vartheta_i \tag{7.20}$$

where both ω_i and ϑ_i are scalar values: ω_i is called the *overlap* of P and P_i, while ϑ_i is the *aversion* of P towards P_i.

[6]However, there are situations, where the participant may temporarily tolerate even wounds and noncompliance, for example when performing a so-called *Re-entering* action (see Section 7.3.3.1).

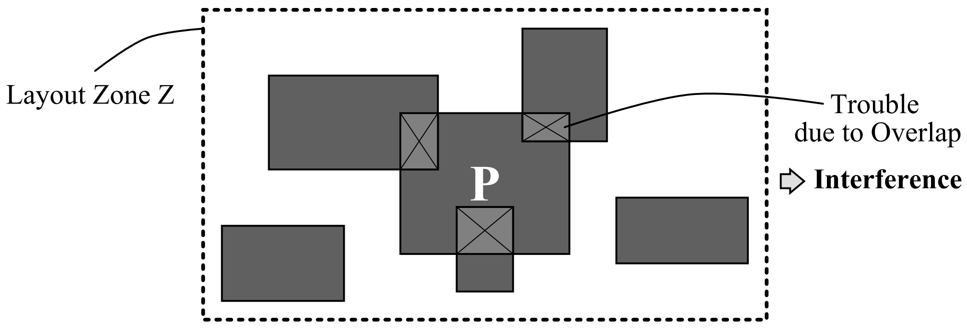


Figure 7.16: Illustration of interference for a participant P.

The overlap ω_i of P and P_i is their intersection area, multiplied with the area of P_i. The first multiplicand is denoted as the *intensity* of the overlap, while the second multiplicand is called *tenacity*. Using the geometrical operators from page 243, the calculation of the overlap can be expressed as

$$\omega_i = \underbrace{\lgroup ⬚P \sqcap ⬚P_i \rgroup}_{\text{intensity}} \cdot \underbrace{\lgroup ⬚P_i \rgroup}_{\text{tenacity}} \ . \tag{7.21}$$

One remark should be made about the ⬚ operator, which determines the bounding box of a geometrical object. Since a participant P is not a rudimentary geometrical object (i.e., a mere *shape* such as a polygon) but a hierarchical design component, the bounding box of P must be determined by examining all physically relevant shapes throughout the entire subhierarchy of P.

Definition 7.1. *Consider a participant P, given as an instance of a procedural generator. Let P contain a set of geometrical shapes $\mathcal{G}$ and a set of subinstances $\mathcal{P}$. For the semiconductor technology at hand, let Λ be the set of layout layers that are physically relevant (e.g., excluding logical layers, auxiliary layers and text layers). Then, the set $\mathcal{G}'$ of physically relevant shapes inside P is a subset of $\mathcal{G}$ containing every geometrical shape G whose layout layer ℓ_G is a physically relevant layer:*

$$\mathcal{G}' = \{G \mid G \in \mathcal{G} \wedge \ell_G \in \Lambda\}. \tag{7.22}$$

To determine the bounding box of a hierarchical design component, the ⬚ operator can now be defined in a recursive fashion[7] *as follows:*

$$\begin{aligned} ⬚P = \Bigg(& \bigg(\min\Big(\min_{\forall G \in \mathcal{G}'} \big(\vdash G\big), \min_{\forall P' \in \mathcal{P}} \big(\vdash (⬚P')\big)\Big), \min\Big(\min_{\forall G \in \mathcal{G}'} \big(\perp G\big), \min_{\forall P' \in \mathcal{P}} \big(\perp (⬚P')\big)\Big)\bigg), \\ & \bigg(\max\Big(\max_{\forall G \in \mathcal{G}'} \big(\dashv G\big), \max_{\forall P' \in \mathcal{P}} \big(\dashv (⬚P')\big)\Big), \max\Big(\max_{\forall G \in \mathcal{G}'} \big(\top G\big), \max_{\forall P' \in \mathcal{P}} \big(\top (⬚P')\big)\Big)\bigg) \Bigg). \end{aligned} \tag{7.23}$$

This expression returns the bounding box of P in rectangle notation $((\check{x}, \check{y}), (\hat{x}, \hat{y}))$, *where $(\check{x}, \check{y})$ represents the south-western vertex of the bounding box and $(\hat{x}, \hat{y})$ represents the north-eastern vertex of the bounding box.*

With the inclusion of the tenacity $\lgroup ⬚P_i \rgroup$ into equation 7.21, the repulsion against P is correlated with the size of the overlapping participant P_i. This idea comes to the fore in the calculation of the *prospective*[8] interference that P can achieve by performing an action: if P is compelled to move into a new location where it also overlaps with another participant P', P thus prefers a smaller participant over a larger one (because the prospective interference is smaller). Since P' then in turn has to perform an action to get rid of the interference, being rather small than large is advantageous for the progress of self-organization because finding a new location is easier for smaller participants. In other words,

[7] The formula is recursive since $⬚P$ also appears on the right-hand side of the expression as $⬚P'$. This recursive formulation is valid because the deepest component in each branch of P's subhierarchy has no subinstances, i.e., $\mathcal{P} = \emptyset$ for every leaf component of the subhierarchy tree.

[8] See Section 7.3.4.

including $[\square P_i]$ into equation 7.21 achieves that the necessity to move away from situations of interference is inclined to propagate from larger participants to smaller participants. Therefore, the actions to perform become less and less disruptive which benefits the convergence of the overall interaction flow.

Participant's Desire 1 (Interference):
If a participant does not overlap with other participants, then the participant is said to be clear. *During the interaction, every participant strives to act in a way such that it becomes clear. If $\mathcal{P}$ denotes the set of all participants, then the desire of a participant P regarding interference is to achieve that*

$$\forall P' \in \mathcal{P} - P,\ \square P \sqcap \square P' = \varnothing \tag{7.24}$$

which means that

$$\Upsilon_P = 0. \tag{7.25}$$

In the beginning of the interaction, there is no aversion among the participants. So, the initial aversion is zero:

$$\vartheta_{i,0} = 0. \tag{7.26}$$

The same is true for the initial number of *clashes* γ between the participants. Hence:

$$\gamma_{i,0} = 0. \tag{7.27}$$

An overlap ω_i of P and P_i increases P's aversion towards P_i such that the aversion changes from the current value $\vartheta_{i,j}$ to the new value $\vartheta_{i,j+1}$ according to the formula

$$\vartheta_{i,j+1} = (\vartheta_{i,j} + \omega_i) \cdot (\gamma_{i,j} + 1). \tag{7.28}$$

Here, $\gamma_{i,j}$ is the previous number of clashes between P and P_i. After the calculation of $\vartheta_{i,j+1}$, the number of clashes is incremented to the value

$$\gamma_{i,j+1} = \gamma_{i,j} + 1 \tag{7.29}$$

due to the overlap. If there is no overlap between participants P and P_i within one round of interaction, the aversion drops to the value

$$\vartheta_{i,j+1} = \vartheta_{i,j} \cdot \varphi \tag{7.30}$$

where φ is a *conciliation quota* chosen within the interval $[0, 1]$. If $\varphi = 1$, there is no conciliation. In this case, the aversion is never reduced but can only become larger, which means that

$$\vartheta_{i,j+1} \geq \vartheta_{i,j}. \tag{7.31}$$

As soon as the interacting participants yield a *settlement* that is *viable*, the aversion and the number of clashes are both reset to zero for each pair of participants.

7.3.1.2 Turmoil

Turmoil is used to take distances between participants into account during the interaction. As shown in Figure 7.17, the distance between two participants is modeled as a straight *connection* between their centerpoints. Formally, every connection C is a quadruple (L_1, L_2, e, s) where L_1 and L_2 are the two endpoints of C, while e is called the *emphasis* and s represents the *strength* of the connection. Both e and s are scalar values that remain constant during a SWARM run.

The emphasis $e \in \mathbb{R}$ is 1 by default and can –depending on the particular requirements of the problem at hand– be individually set by the user within the interval $(0, 1]$ to downgrade the importance of a connection in relation to the other connections (there being a distinction between SWARM's idea of emphasis and the algorithmic notion of a *weight*, as will be discussed in Section 7.5.1). The strength $s \in \mathbb{N}^+$ of a connection between two participants P_1 and P_2 is automatically calculated as the sum

$$s = \eta_1 + \eta_2 - 1 \tag{7.32}$$

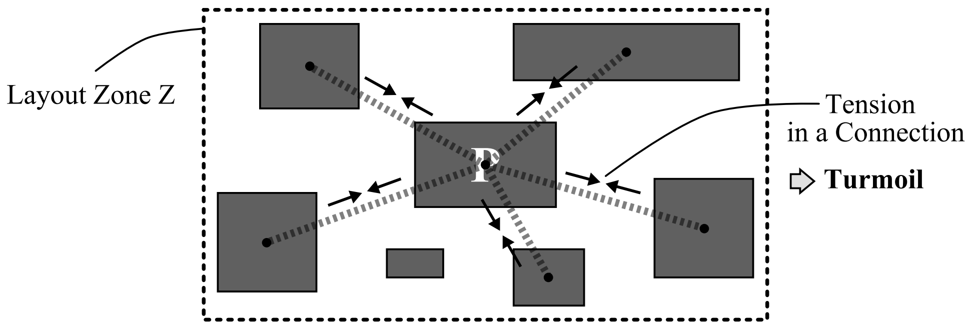


Figure 7.17: Illustration of turmoil for a participant P.

where η_1 and η_2 represent the total *number of connections* of P_1 and P_2 respectively (the subtrahend -1 effectuates that the common connection of P_1 and P_2 is not counted twice). Similar to the idea of tenacity in equation 7.21, the consideration of strength has the purpose of streamlining the self-organization. As illustrated in Figure 7.18, a participant is rather drawn to another one having many connections than to one having fewer connections, which reduces the number of participants that need to follow this motion due to connectivity.

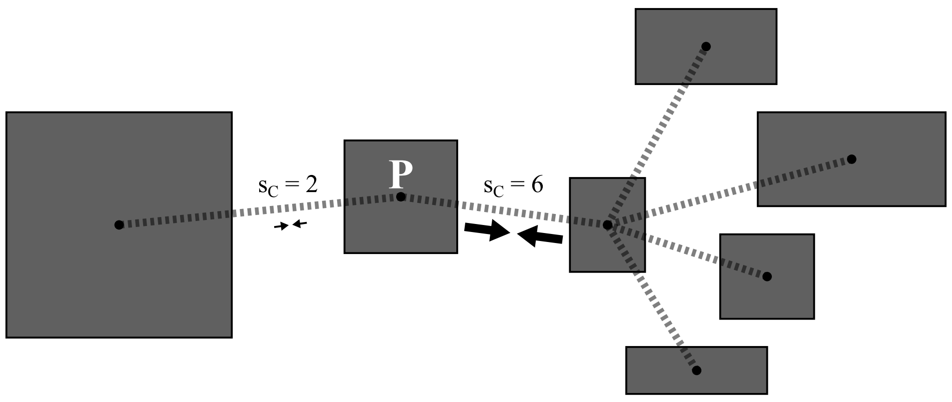


Figure 7.18: Effect of the strength s_C of a connection C on a participant's action.

The condition of a participant sustains negative influence, if its distance to a connected participant exceeds a certain threshold. For any connection, this so-called *relaxation threshold* ϱ is defined as follows.

Definition 7.2. *Given a set $\mathcal{P}$ of participants inside the layout zone Z, consider a connection C between two participants P_1 and P_2. Let Q_1 be a square with an area equal to the area of P_1, and let r_1 be the radius of the smallest possible circle around Q_1. Furthermore, let λ be a* leeway coefficient *calculated as*

$$\lambda = \sqrt{\frac{[Z]}{[\boxplus\mathcal{P}]}} = \sqrt{\frac{[Z]}{\sum\limits_{P\in\mathcal{P}} [\square P]}} \tag{7.33}$$

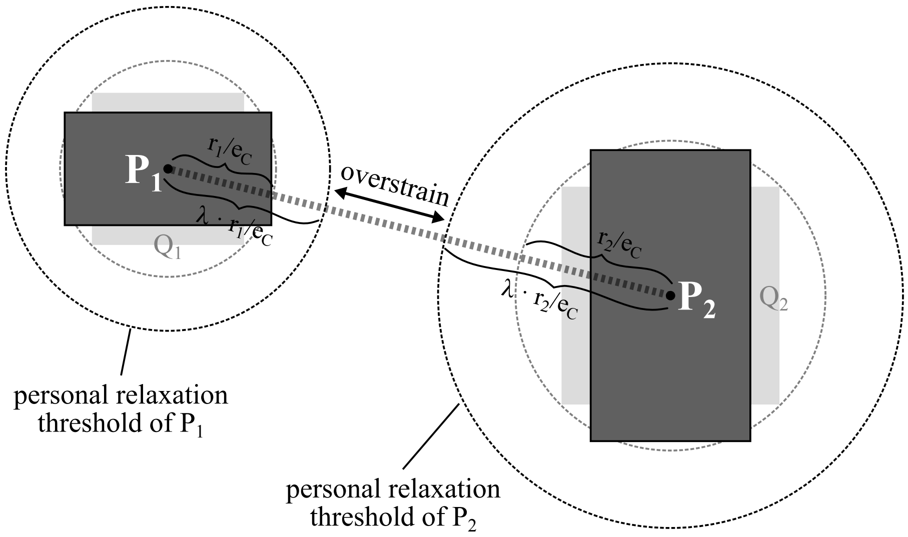
where $[Z]$ is the area of Z and $[\boxplus\mathcal{P}]$ is the sum of the participants' individual areas. Then, the value

$$\varrho_{C,1} = \frac{\lambda \cdot r_1}{e_C} \tag{7.34}$$

where e_C represents the emphasis of C, is denoted as the "personal" relaxation threshold *of P_1 for connection C. With $\varrho_{C,2}$ as the personal relaxation threshold of P_2, the sum*

$$\varrho_C = \varrho_{C,1} + \varrho_{C,2} \tag{7.35}$$

is defined as the "total" relaxation threshold *of connection C.*



r1/eC
P1
overstrain
λ · r1/eC
Q1
r2/eC
λ · r2/eC
P2
Q2
personal relaxation threshold of P1
personal relaxation threshold of P2

Figure 7.19: Personal relaxation thresholds and overstrain between two connected participants.

To illustrate the idea of relaxation threshold, Figure 7.19 shows the connection C of two participants P_1 and P_2, each of which is circumfered by its personal relaxation threshold $\varrho_{C,1}$ and $\varrho_{C,2}$ (for $\lambda = 1.5$ and $e_C = 1$). The difference $l_C - (\varrho_{C,1} + \varrho_{C,2})$ between the length l_C of the connection and the total relaxation threshold is denoted as *overstrain* in the figure.

The radius r of the smallest possible circle around a square Q is half the diagonal of Q and can be determined via

$$r = \frac{1}{2}\sqrt{2[Q]}. \tag{7.36}$$

According to the definition above, the area of Q_1 is equal to the area of P_1. This is also true for the area of Q_2 and P_2:

$$[Q_1] = [\square P_1],\ [Q_2] = [\square P_2]. \tag{7.37}$$

Using equation 7.36 and equation 7.37, the relaxation threshold can thus be calculated in closed form according to the following formula:

$$\begin{aligned}
\varrho_C &= \frac{\lambda \cdot r_1}{e_C} + \frac{\lambda \cdot r_2}{e_C} && (7.38)\\
&= \frac{\lambda}{e_C} \cdot (r_1 + r_2) && (7.39)\\
&= \frac{\lambda}{e_C} \cdot \left(\frac{1}{2}\sqrt{2[\square P_1]} + \frac{1}{2}\sqrt{2[\square P_2]}\right) && (7.40)\\
&= \frac{\lambda}{\sqrt{2} \cdot e_C} \cdot \left(\sqrt{[\square P_1]} + \sqrt{[\square P_2]}\right). && (7.41)
\end{aligned}$$

The length l_C of a connection C is simply the Euclidean distance between its endpoints L_1 and L_2. With $L_1 = (x_1, y_1)$ and $L_2 = (x_2, y_2)$, the length is given by Pythagoras' theorem:

$$l_C = \overline{L_1 L_2} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \tag{7.42}$$

Participant's Desire 2 (Turmoil):
If the length of a connection C is below its relaxation threshold ϱ_C, the connection is said to be relaxed *(otherwise* unrelaxed*). During the interaction, every participant strives to act in a way such that all of its connections become relaxed, in which case the participant itself is said to be relaxed. If $\mathcal{C}_P$ denotes the set of a participant P's connections, then the participant's desire regarding turmoil is to achieve that*

$$\forall C \in \mathcal{C}_P,\ l_C \leq \varrho_C. \tag{7.43}$$

Using $\zeta_P \in \mathbb{N}^0$ to denote the number of P's unrelaxed connections, the desire of P to become relaxed can be shortly written as

$$\zeta_P = 0. \tag{7.44}$$

The leeway coefficient λ prevents the participants from clumping together when there is still much space left in the beginning of a SWARM run. With every zone tightening, λ becomes smaller and approaches 1 since Z is successively downsized towards a minimal outline Z' whose area equals $[\text{⁘}\mathcal{P}]$:

$$\lim_{Z \to Z'} \lambda(Z) = \lim_{Z \to Z'} \sqrt{\frac{[Z]}{[\text{⁘}\mathcal{P}]}} = 1\ . \tag{7.45}$$

Assuming that the area of a participant is roughly the same among all layout variants the participant can deform into, the relaxation threshold of a connection C only depends on λ since e_C is a constant value. Thus, the relaxation threshold ϱ_C can also be written as a function of Z, which successively converges towards

$$\lim_{Z \to Z'} \varrho_C(Z) = \underbrace{\lim_{Z \to Z'} \lambda(Z)}_{=1} \cdot \frac{1}{e_C} \cdot (r_1 + r_2) = \frac{r_1 + r_2}{e_C} \tag{7.46}$$

throughout the course of a SWARM run. In this fashion, the participants are carefully drawn into their final arrangement without depriving them of the leeway they need to organize themselves. If the emphasis of a connection C is set to the default value $e_C = 1$, the relaxation threshold during the last tightening-settlement cycle is $\varrho_C = r_1 + r_2$, so the two participants P_1 and P_2 are expected to move into immediate vicinity of each other.

The above considerations deal with the modeling of a single connection C. But since a participant can be connected to multiple other participants, multiple connections must be taken into account during the interaction. For that purpose, the turmoil Θ of a participant P is calculated by adding up the so-called *tension* θ in each of P's n connections $C_1, C_2, \ldots, C_n$:

$$\Theta_P = \sum_{i=1}^{n} \theta_i. \tag{7.47}$$

The concept of tension elucidates the notion of the term *turmoil*: a participant virtually is "in a turmoil" for being "torn" apart by its omnidirected connections, all of which influence the actions of the participant like tensely strained rubber straps. The respective tension θ in a connection C is given by

$$\theta(l_C) = \begin{cases} l_C \cdot s_C e_C & \Leftrightarrow\ l_C \le \varrho_C e_C \\ \left((l_C + \frac{1}{2} - \varrho_C e_C)^2 - \frac{1}{4} + \varrho_C e_C\right) \cdot s_C e_C & \Leftrightarrow\ l_C > \varrho_C e_C \end{cases} \tag{7.48}$$

where the only nonconstant values are the connection's relaxation threshold ϱ_C (which is changed with each tightening of the layout zone) and the connection's length l_C (which is meant to be reduced via moves of the participants). For

$$l_C \le \underbrace{\varrho_C e_C}_{=\lambda \cdot (r_1 + r_2)} \tag{7.49}$$

the relation of tension to length is linear, otherwise it is quadratic to penalize longer distances more severely und thus vivify the interaction. As illustrated by the graph in Figure 7.20, the constituent formulas of equation 7.48 are chosen such that the entire function is continuous and differentiable. Therefore, using $\theta_\dagger(l_C)$ and $\theta_\ddagger(l_C)$ to denote the constituent formulas for $l_C \le \varrho_C e_C$ and $l_C > \varrho_C e_C$ respectively, the tension $\theta_\dagger(\varrho_C e_C)$ equals the tension $\theta_\ddagger(\varrho_C e_C)$ as proven by:

$$\begin{aligned} \overbrace{\theta_\dagger(\varrho_C e_C)}^{\theta(l_C)\text{ for }l_C \le \varrho_C e_C} &= \overbrace{\theta_\ddagger(\varrho_C e_C)}^{\theta(l_C)\text{ for }l_C > \varrho_C e_C} \\ \varrho_C e_C \cdot s_C e_C &= \left((\varrho_C e_C + \tfrac{1}{2} - \varrho_C e_C)^2 - \tfrac{1}{4} + \varrho_C e_C\right) \cdot s_C e_C \\ \varrho_C\, s_C\, e_C^2 &= \left((\tfrac{1}{2})^2 - \tfrac{1}{4} + \varrho_C e_C\right) \cdot s_C e_C \\ \varrho_C\, s_C\, e_C^2 &= \varrho_C e_C \cdot s_C e_C \\ \varrho_C\, s_C\, e_C^2 &= \varrho_C\, s_C\, e_C^2. \quad \square \end{aligned}$$

Equivalently, the slope of $\theta_\dagger(l_C)$ is identical to the slope of $\theta_\ddagger(l_C)$ for $l_C = \varrho_C e_C$. Using $\theta'_\dagger(l_C) = s_C e_C$ and $\theta'_\ddagger(l_C) = (2l_C + 1 - 2\varrho_C e_C) \cdot s_C e_C$ to notate the derivatives of $\theta_\dagger(l_C)$ and $\theta_\ddagger(l_C)$ respectively, where $\theta'_\ddagger(l_C)$ has been determined after expanding

$$\begin{aligned}
\theta_\ddagger(l_C) &= \left((l_C + \tfrac{1}{2} - \varrho_C e_C)^2 - \tfrac{1}{4} + \varrho_C e_C\right) \cdot s_C e_C \\
&= \left(l_C^2 + 2l_C(\tfrac{1}{2} - \varrho_C e_C) + (\tfrac{1}{2} - \varrho_C e_C)^2 - \tfrac{1}{4} + \varrho_C e_C\right) \cdot s_C e_C \\
&= (l_C^2 + l_C - 2\varrho_C e_C l_C + (\tfrac{1}{2})^2 - \varrho_C e_C + \varrho_C^2 e_C^2 - \tfrac{1}{4} + \varrho_C e_C) \cdot s_C e_C \\
&= (l_C^2 + l_C - 2\varrho_C e_C l_C + \varrho_C^2 e_C^2) \cdot s_C e_C,
\end{aligned}$$

the proof can be given as:

$$\begin{aligned}
\overbrace{\theta'_\dagger(\varrho_C e_C)}^{\theta'(l_C)\text{ for }l_C \le \varrho_C e_C} \quad &= \quad \overbrace{\theta'_\ddagger(\varrho_C e_C)}^{\theta'(l_C)\text{ for }l_C > \varrho_C e_C} \\
s_C e_C &= (2\varrho_C e_C + 1 - 2\varrho_C e_C) \cdot s_C e_C \\
s_C e_C &= s_C e_C. \quad \square
\end{aligned}$$

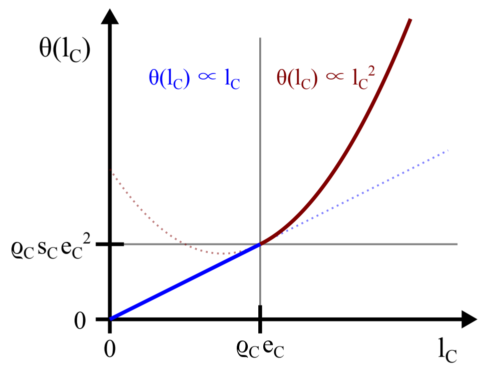


Figure 7.20: Tension in a connection between two participants, depending on the connection's length.

7.3.1.3 Protrusion

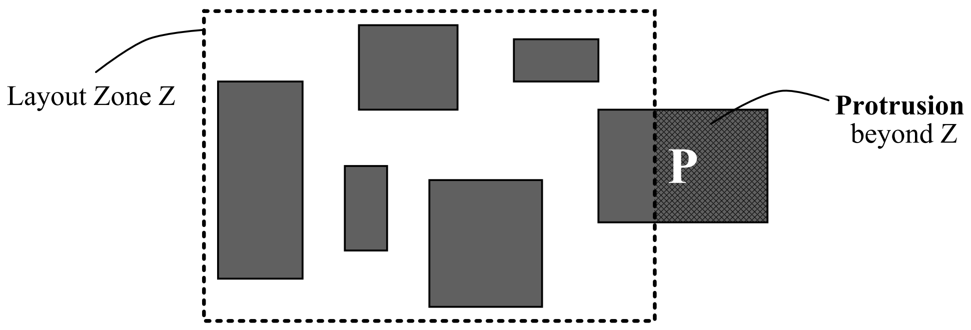


Figure 7.21: Illustration of protrusion for a participant P.

Protrusion is the case when a participant does not completely lie within the current outline of the layout zone, as depicted in Figure 7.21. In that case, the participant is forced to take an action. Geometrically, the protrusion Ψ of a participant P can be written as

$$\Psi_P = \text{⬚}P \sqcap \overline{Z} \tag{7.50}$$

where $\overline{Z}$ (the complement of Z) denotes the territory beyond the layout zone.

Depending on the grade of protrusion, a participant is denoted as *lost* (entirely outside Z), *prone* (partially outside Z), or *safe* (entirely inside Z). Each of these three cases is illustrated in Figure 7.22 and bijectively maps to a distinct geometrical condition as follows:

$$P \text{ is } \begin{cases} \textit{lost} & \Leftrightarrow \quad \Psi_P = \square P \\ \textit{prone} & \Leftrightarrow \quad \Psi_P \sqsubset \square P \\ \textit{safe} & \Leftrightarrow \quad \Psi_P = \varnothing. \end{cases} \tag{7.51}$$

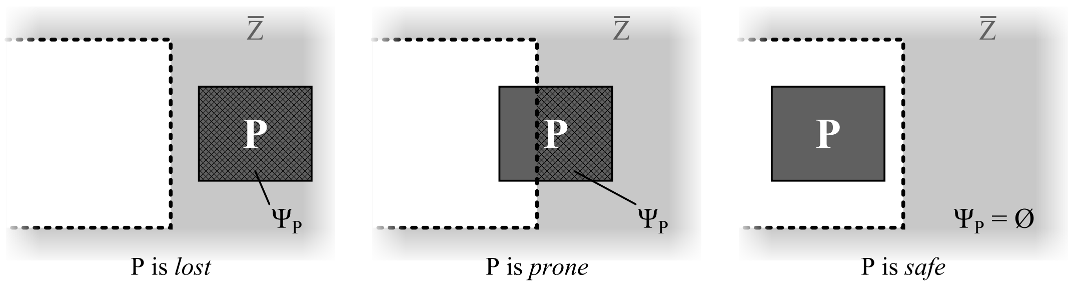


Figure 7.22: The three grades of protrusion: a participant can be either lost, prone, or safe.

Participant's Desire 3 (Protrusion):
As mentioned above, a participant that does not at all protrude the layout zone is said to be safe. *During the interaction, every participant strives to act in a way such that it becomes safe. So, the desire of a participant P regarding protrusion is to achieve that*

$$\Psi_P = \varnothing. \tag{7.52}$$

If a participant would become *prone* after performing one of its potential actions, it is good to know how far exactly the participant would protrude the layout zone horizontally and vertically. As will be discussed in the end of Section 7.3.3.1, this *protrusion extent* (ψ_x, ψ_y) can be used to correct the potential action such that it leads the participant into a safe location. To determine the protrusion extent of a prone participant, eight different cases of protrusion are to be distinguished, as shown in Figure 7.23. In that regard, it is important to note that the layout zone may be rectilinear, not just rectangular.

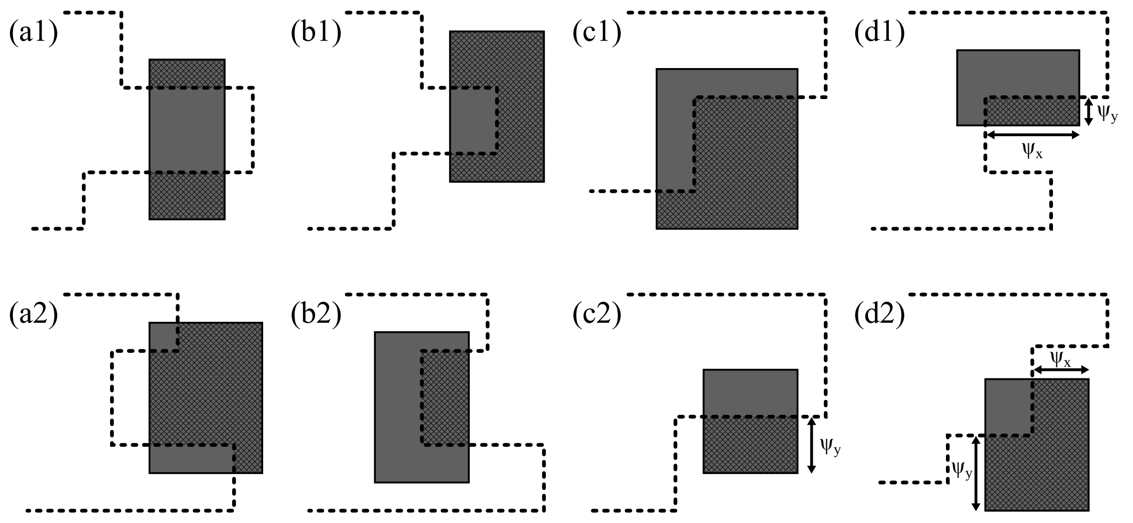


Figure 7.23: The different cases of protrusion for a prone participant and a rectilinear layout zone.

Referring to the eight images in Figure 7.23, there are five cases in which it is not possible to determine an unambiguous protrusion extent, depending on the number and the form of those parts of a participant P that lie inside and outside the layout zone Z:

(a1) There are more than one disjunct parts of P that lie outside of Z.
(a2) There are more than one disjunct parts of P that lie inside of Z.
(b1) The part of P inside Z is rectangular, but the outside part is a concave polygon with more than one reflex interior angle (i.e., an interior angle greater than 180°).
(b2) The part of P outside Z is rectangular, but the inside part is a concave polygon with more than one reflex interior angle.
(c1) The part of P outside Z and the part inside Z are both not rectangular.

In these five cases, no *action correction* is performed due to the lack of an unequivocal protrusion extent. Furthermore, this is justified by the conjecture that the likelihood of encountering one of these cases is rather low. If Z is rectangular, cases (a1) and (b1) are merely theoretical, while the occurrence of cases (a2), (b2) or (c1) is even impossible. In the remaining three cases, the calculation of the protrusion extent is well-defined and trivial:

(c2) The part of P outside Z and the part inside Z are both rectangular. In this case, the protrusion extent is given by the dimensions of the part outside Z.
(d1) The part of P outside Z is rectangular and the inside part is a concave polygon with only one reflex interior angle. As in (c2), the protrusion extent is given by the dimensions of the part outside Z.
(d2) The part of P inside Z is rectangular and the outside part is a concave polygon with only one reflex interior angle. Then, the protrusion extent is given by the bounding box dimensions of P minus the dimensions of the part inside Z.

With the aim of performing an action correction, an important matter must be respected if the protrusion occurs at an edge (not at a vertex) of the layout zone: either the horizontal or the vertical protrusion extent must be zero since an *axis-oriented* action correction is desired in this case. More specifically, if P protrudes Z upwards or downwards –as in the example of (c2)– this requires that $\psi_x = 0$ (to obtain a purely vertical action correction). If P protrudes Z leftwards or rightwards, then $\psi_y = 0$ (horizontal action correction). This can be achieved by calculating the protrusion extent in the same way as described above regarding case (d2).

Whether a geometrical shape G is a rectangle, can be determined by evaluating if the condition $\square G = G$ holds true. When it is known that every angle in G is a right angle, an alternative option is to check if the number of vertices is $|G| = 4$. Whether a rectilinear, not self-intersecting polygon G has exactly one reflex interior angle, can be determined by evaluating if $|G| = 6$. Thus, the calculation of the protrusion extent for the cases (c2), (d1), and (d2) can be done according to the following formulas

$$\psi_x = \begin{cases} \dashv(\square P \sqcap \overline{Z}) - \vdash(\square P \sqcap \overline{Z}) & \Leftrightarrow \quad |\square P \sqcap Z| = 6 \wedge |\square P \sqcap \overline{Z}| = 4 \\ \dashv(\square P) - \vdash(\square P) - \dashv(\square P \sqcap Z) + \vdash(\square P \sqcap Z) & \Leftrightarrow \quad |\square P \sqcap Z| = 4 \wedge |\square P \sqcap \overline{Z}| \leq 6 \end{cases} \tag{7.53a}$$

$$\psi_y = \begin{cases} \top(\square P \sqcap \overline{Z}) - \bot(\square P \sqcap \overline{Z}) & \Leftrightarrow \quad |\square P \sqcap Z| = 6 \wedge |\square P \sqcap \overline{Z}| = 4 \\ \top(\square P) - \bot(\square P) - \top(\square P \sqcap Z) + \bot(\square P \sqcap Z) & \Leftrightarrow \quad |\square P \sqcap Z| = 4 \wedge |\square P \sqcap \overline{Z}| \leq 6 \end{cases} \tag{7.53b}$$

where $|\square P \sqcap Z| = 4 \wedge |\square P \sqcap \overline{Z}| = 4$ represents case (c2), $|\square P \sqcap Z| = 6 \wedge |\square P \sqcap \overline{Z}| = 4$ represents case (d1), and $|\square P \sqcap Z| = 4 \wedge |\square P \sqcap \overline{Z}| = 6$ represents case (d2).

7.3.1.4 Wounds

When a participant P assesses its condition, every overlap ω between P and another participant inflicts wounds on P, as exemplified in Figure 7.24. Formally, a wound W is a pair (ρ, ς) which has a specific rectangular *region* ρ on P and a discrete *severity* ς represented as a natural number $\varsigma \in \mathbb{N}^0$.

An overlap between P and another participant P' inflicts a new wound W on P which has a severity $\varsigma_W = 1$ and covers the overlap region $\rho_W = \square P \sqcap \square P'$. If that region also overlies an existing wound W^* on P, then the wound is *aggravated* throughout the part of W^* overlain by ρ_W. That is to say, the severity of the wounded part $\rho_W \sqcap \rho_{W^*}$ is incremented by $\Delta\varsigma$, wherein the value for $\Delta\varsigma$ is either 2 (if W^* was inflicted by P') or 1 (if W^* was inflicted by a participant other than P'). Those parts of a wound,

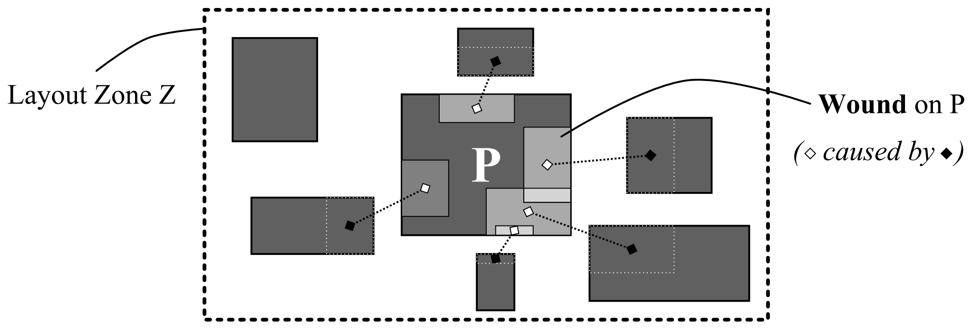


Figure 7.24: Illustration of wounds for a participant P.

that do not become aggravated due to an overlap, can *ameliorate*. This is achieved by decrementing the severity of those parts by 1. If the severity drops below zero, then that part of the wound vanishes and is said to be *healed*.

One particular comment should be made about the implementation of the wounds concept. Aggravating the overlain part ρ^* of an existing wound can be done by geometrically separating that part from the wound and then incrementing its severity to the new value ς^*. However, this leads to the necessity of dealing with arbitrarily intricate polygonal contours of wounded parts. A convenient alternative is simply to inflict a new wound with the increased severity ς^* on top of the existing wound, covering the region ρ^*. With this practice, geometrical operations on wounds involve only rectangular regions, as Figure 7.25 illustrates in an example.

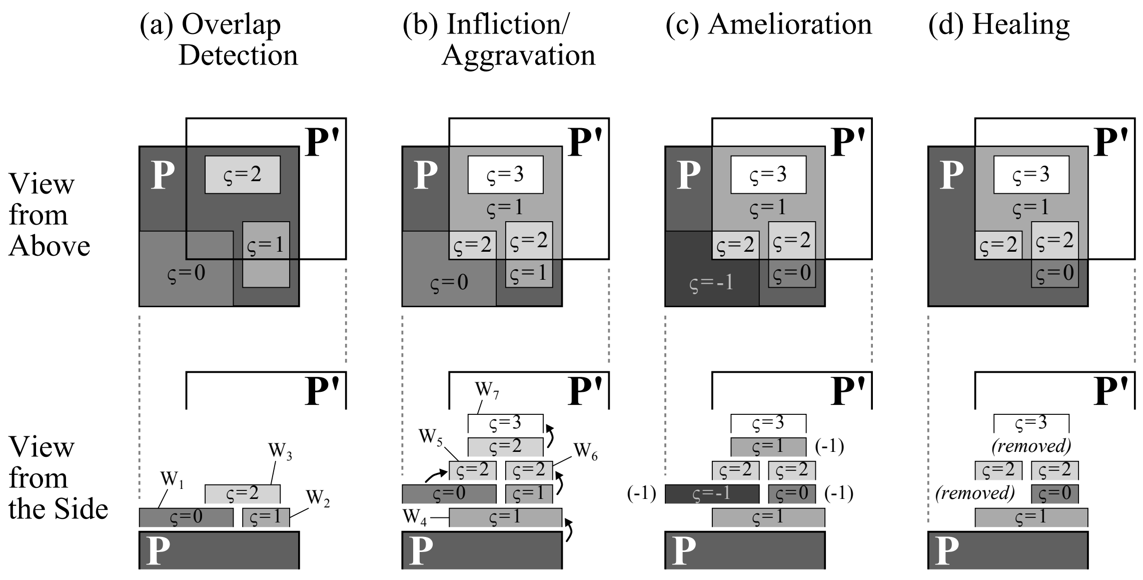


Figure 7.25: Exemplary depiction of how a participant P gets wounded by another participant P'.

For this example, Figure 7.25 shows a view from above onto the layout, and a cross section through the “stack” of wounds. Both depictions are given for the four different stadiums of the wound infliction which can be described as follows:

(a) *Overlap Detection:*
Given is a participant P with three wounds: a wound W_1 with a severity of $\varsigma = 0$, a wound W_2 with $\varsigma = 1$, and a wound W_3 with $\varsigma = 2$. As illustrated in the image, P is overlapped by another participant P' which was already responsible for having inflicted wound W_1 in a previous round of interaction.

(b) *Infliction/Aggravation:*
The overlap inflicts a new wound W_4 on P with a severity of $\varsigma = 1$, covering the overlap region $\square P \sqcap \square P'$. Since the overlap also intersects W_1 and W_2 in parts and encompasses W_3 in full, these

wounds are aggravated. That means, the overlap region $\square P' \sqcap \rho_{W_1}$ is aggravated by causing another wound W_5 with $\varsigma = 2$ (i.e., $\Delta\varsigma = 2$, since W_1 was inflicted by P'), while the regions $\square P' \sqcap \rho_{W_2}$ and $\square P' \sqcap \rho_{W_3}$ are aggravated from $\varsigma = 1$ to $\varsigma = 2$ and from $\varsigma = 2$ to $\varsigma = 3$, which is achieved via new wounds W_6 and W_7 respectively (i.e., $\Delta\varsigma = 1$, since W_2 and W_3 were not inflicted by P').

(c) *Amelioration:*
Now that the new wounds (W_4, W_5, W_6, and W_7) have been added to P, the old wounds (W_1, W_2, and W_3) are ameliorated. This is done via decrementing the respective severity by $\Delta\varsigma = -1$. For W_3, the severity thus becomes $\varsigma = 1$, and for W_2, the severity reaches $\varsigma = 0$. For W_1, the severity drops to the (actually excluded) value $\varsigma = -1$, which is thus subject to healing in the next stadium.

(d) *Healing:*
Since the severity of W_1 has gone below zero, the wound has fully healed and is therefore removed from the stack of wounds. Hence, it is again true that the severity of all remaining wounds is $\varsigma \in \mathbb{N}^0$. Since the new wound W_7 covers the old wound W_3 not only in parts but in full, W_3 is in fact inane and can therefore also be removed.

When the aggravation of a wound exceeds a *critical severity* ς_c, the wound is said to be critical and the participant is forced to perform an action. In that case, the participant begins a strategy of *recuperation* and chooses its subsequent actions such that the wound is not aggravated any further before being fully healed. By the same token, a participant is not allowed to aggravate another participant's wound if that wound is currently subject to recuperation.

Participant's Desire 4 (Wounds):
If a participant's wound is overlapped by another participant despite being subject to recuperation, or if a participant overlaps another participant's wound which is currently subject to recuperation, then the participant is said to be in an unhealthy *location, otherwise in a* healthy *location. During the interaction, every participant strives to act in a way such that it does not get into an unhealthy location. If $\mathcal{P}$ denotes the set of all participants and $\mathcal{W}_P^{\dagger}$ represents a participant P's set of wounds that are currently subject to recuperation, then the desire of a participant P regarding wounds is to achieve that*

$$\forall P' \in \mathcal{P} - P\,,\ (\nexists W \in \mathcal{W}_P^{\dagger} : \square P' \sqcap \rho_W \neq \varnothing) \wedge (\nexists W' \in \mathcal{W}_{P'}^{\dagger} : \square P \sqcap \rho_{W'} \neq \varnothing). \tag{7.54}$$

Using

$$\mathcal{W}_{P_{\ltimes}}^{\ddagger} = \{W \in \mathcal{W}_P^{\dagger} \mid \exists P' \in \mathcal{P} - P : \square P' \sqcap \rho_W \neq \varnothing\} \tag{7.55}$$

to denote the set of P's recuperating wounds currently overlapped by other participants, and using

$$\mathcal{W}_{P_{\rtimes}}^{\ddagger} = \{W' \mid \exists P' \in \mathcal{P} - P : W' \in \mathcal{W}_{P'}^{\dagger} \wedge \square P \sqcap \rho_{W'} \neq \varnothing\} \tag{7.56}$$

to denote the set of other participants' recuperating wounds currently overlapped by P, then P is in a healthy location if the union $\mathcal{W}_{P_{\bowtie}}^{\ddagger}$ of both sets is empty:

$$\mathcal{W}_{P_{\bowtie}}^{\ddagger} = \mathcal{W}_{P_{\ltimes}}^{\ddagger} \cup \mathcal{W}_{P_{\rtimes}}^{\ddagger} = \emptyset. \tag{7.57}$$

Wounds are a pivotal element of SWARM, akin to the idea of aversion in Section 7.3.1.1 but with different motives in two regards. First, aversion has a long-term effect in that it hinders a perpetual interference of two participants, whereas a critical wound has an immediate impact. Second, as aversion correlates with the area of an overlap, wounds are more effective for preventing marginal interferences. A careful balance in the modeling of both aversions and wounds is one key to a fluent progress of self-organization in SWARM.

7.3.1.5 Noncompliance

Noncompliance indicates that a participant, in its current location, violates at least one explicitly formulated design constraint. An example is given in Figure 7.26, where participant P violates an Alignment

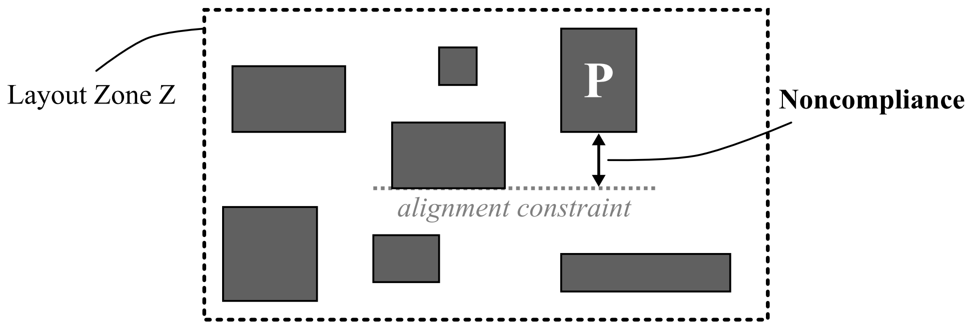


Figure 7.26: Illustration of noncompliance for a participant P.

constraint. Noncompliance specifically addresses only *hard constraints* (strict confinements), since *soft constraints* (optimization goals) are incorporated into SWARM via other mechanisms. Mentioning two examples for the latter, wirelength minimization is achieved with the consideration of turmoil (Section 7.3.1.2), while minimization of the total area is pursued through the successive tightening of the layout zone (Section 7.4).

Formally, a hard constraint imposes a certain restriction on a set $\mathcal{M}$ of design objects denoted as the *constraint members*. For a concrete, hard constraint H of constraint type t_H, there must be a type-dependent *verification function* $v_{t_H}(\mathcal{M})$ whose codomain is the Boolean domain $\mathbb{B} = \{0, 1\}$. A return value of 1 indicates that the constraint H is currently satisfied for its constraint members $\mathcal{M}_H$, whereas a return value of 0 indicates that the constraint is currently not satisfied.

For example, in the case of an Alignment constraint such as in Figure 7.26, where the constraint members $\mathcal{M}$ are supposed to be aligned by their bottom edge, the verification function $v_{\text{Alignment}}$ can be expressed as

$$v_{\text{Alignment}}(\mathcal{M}) = \begin{cases} 1 & \Leftrightarrow \quad \exists y \in \mathbb{R} : \forall P \in \mathcal{M}, \perp(\text{⬚}P) = y \\ 0 & \Leftrightarrow \quad \text{otherwise.} \end{cases} \tag{7.58}$$

If $\mathcal{H}_P$ denotes the set of all hard constraints that have been imposed on a participant P, i.e.,

$$\mathcal{H}_P = \{H \mid H \in \mathcal{H} \wedge P \in \mathcal{M}_H\}, \tag{7.59}$$

with $\mathcal{H}$ representing the set of all hard constraints in the current design, then participant P is in a state of noncompliance if

$$\exists H \in \mathcal{H}_P : v_{t_H}(\mathcal{M}_H) = 0. \tag{7.60}$$

Participant's Desire 5 (Noncompliance):
If a participant does not violate any explicitly formulated hard design constraint, then the participant is referred to as compliant. *During the interaction, every participant strives to act in a way such that it becomes compliant. So, the desire of a participant P regarding noncompliance is to achieve that*

$$\forall H \in \mathcal{H}_P,\ v_{t_H}(\mathcal{M}_H) = 1. \tag{7.61}$$

Using $\mathcal{H}_P^{\ddagger} = \{H \in \mathcal{H}_P \mid v_{t_H}(\mathcal{M}_H) = 0\}$ *to denote the set of hard constraints imposed on participant P that are currently not satisfied, P is compliant if that set is empty:*

$$\mathcal{H}_P^{\ddagger} = \emptyset. \tag{7.62}$$

As has been stated in Chapter 2, it is the essential aim of this thesis to provide the means for a comprehensive consideration of design constraints. To that effect, the conception of *noncompliance* represents only one of several instruments. While it should again be remarked, that each participant is supposed to take care of its own particular design constraints implicitly, there are also multiple distinct flavors of explicit constraint consideration in SWARM:

- As already mentioned above, wirelength minimization is achieved by the conception of turmoil, while minimization of the total area is addressed through the successive tightening of the layout zone. Wirelength minimization and area minimization represent optimization goals, also referred to as soft constraints.
- Noncompliance, as an influencing factor, can be utilized to target hard design constraints. In that regard, noncompliance is feasible for constraints that are supposed to be satisfiable via SWARM's native catalog of actions (which will be described in Section 7.3.3).
- Certain design constraints cannot be adequately covered by SWARM's native catalog of actions. However, for such constraints a participant may provide and pursue its own dedicated kind of action. An example is the action called *Imitation*, by which a participant mimics another participant's transformations, as will be discussed in Section 7.3.3.2.
- Design constraints pertaining to the distance between two participants, can be included in the model of *tension* (see Section 7.3.1.2). Apart from the *emphasis* value (which acts as a "soft" quantifier), a hard restriction –such as a maximum distance– can be imposed by overriding a connection's *relaxation threshold* with the desired value.
- Some design constraints just imply, that a participant may not assume certain layout variants. That is to say, the participant is not allowed to exploit its entire *variability* during the interaction. Instead, the permitted variability is reduced to a subset of the participant's total variability. For example, a Current Mirror may be prevented from deforming itself into a single-row variant for reasons of matching.

In summary, if a participant is not affected by any of the above influencing factors (interference, turmoil, protrusion, wounds, noncompliance), then the participant is said to be *contented*. Otherwise, the participant strives for action because it is *discontented*. When contented, no action is required, but nonetheless the participant attempts to perform a movement in this case if possible: to center itself within its *free peripheral space* (which will now be covered in Section 7.3.2). If that is not possible, the participant lingers where it is. Although the centering does not improve the participant's condition, it is regarded as a betterment of its situation, imagining that the participant "feels best" when it can equalize the distances to its nearby neighbors.

7.3.2 Perception of the Free Peripheral Space

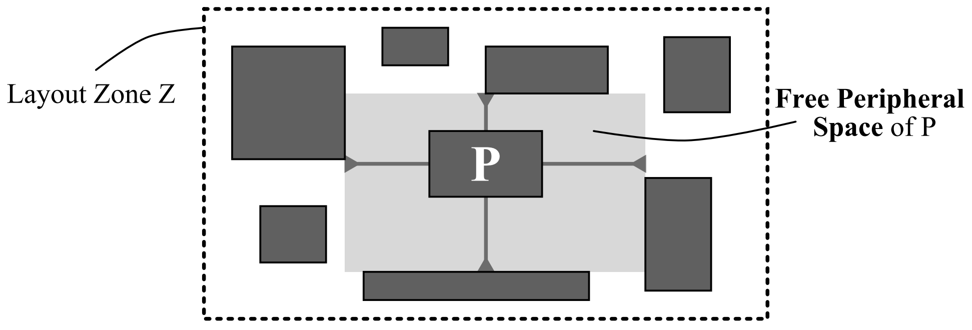


Figure 7.27: The free peripheral space of a participant P is the vacant area around it.

The basis for a participant's exploration of possible actions is the vacant area around it (and, for actions involving other participants, also the vacant areas around *them*). While there are various possible conceptions of how this *free peripheral space* could be specified, SWARM provides the following unambiguous definition, in accordance with the exemplary depiction of Figure 7.27.

Definition 7.3. *Given the layout zone Z as a rectilinear polygon, let P be a participant located (at least partially) inside Z. Let B be the rectangular bounding box around the part of P that is completely inside Z. For every edge E of B, let the* corridor K_E *be a rectangle beginning at E and sprawling away from P to infinity with a width equal to the length of E. Let $\mathcal{K}$ be a set containing the four corridors of P and let $\mathcal{U}$ be a set of obstacles containing the complement of Z as well as the bounding boxes of all participants* *except P. Then, the free peripheral space S_P of P is uniquely defined as the largest possible rectangle around P not containing any intersection $\mathcal{K} \sqcap \mathcal{U}$.*

7.3.2.1 Geometrical Recipe for Perceiving the Free Peripheral Space

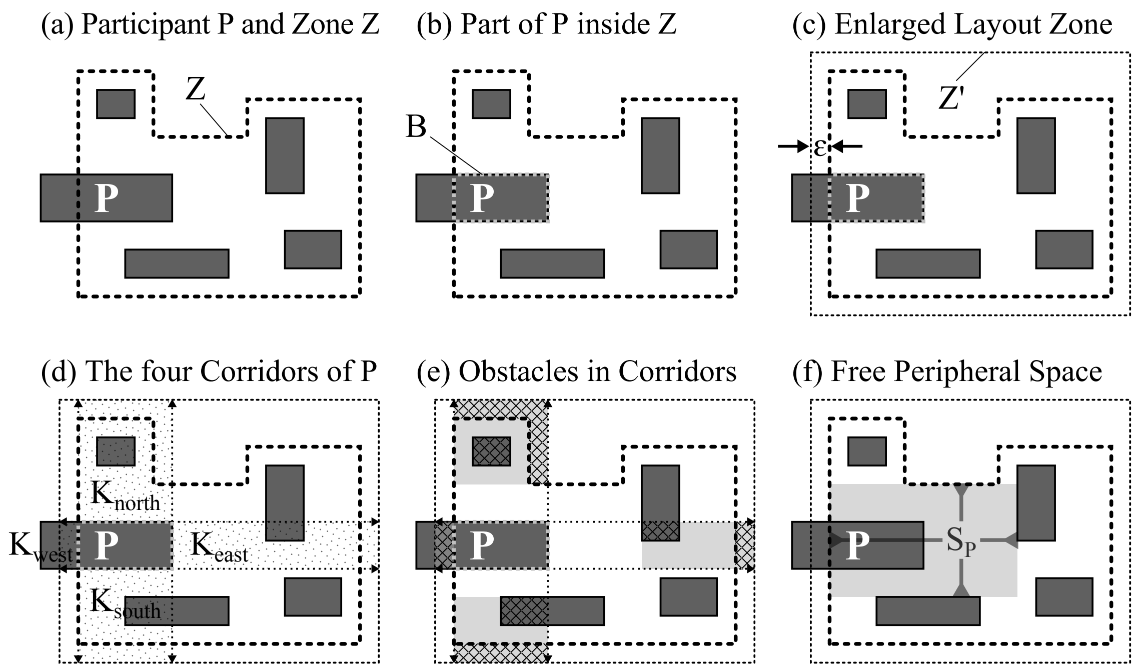

(a) Participant P and Zone Z
Z
P
(b) Part of P inside Z
B
P
(c) Enlarged Layout Zone
Z'
ε
P
(d) The four Corridors of P
K_north
K_west
P
K_east
K_south
(e) Obstacles in Corridors
P
(f) Free Peripheral Space
P
S_P

Figure 7.28: A participant's geometrical recipe for perceiving its free peripheral space.

Geometrically, a participant can perceive its free peripheral space as described below and illustrated in Figure 7.28. For a participant P and a layout zone Z –such as in Figure 7.28 (a)– the bounding box around the part of P inside Z –see (b)– is given by

$$B = ⬚(Z \sqcap ⬚P) \tag{7.63}$$

and the four edges of B can be determined as lines $\big((x_1, y_1), (x_2, y_2)\big)$ via

$$E_{\text{north}} = \big((\vdash B, \top B), (\dashv B, \top B)\big), \tag{7.64a}$$
$$E_{\text{east}} = \big((\dashv B, \bot B), (\dashv B, \top B)\big), \tag{7.64b}$$
$$E_{\text{south}} = \big((\vdash B, \bot B), (\dashv B, \bot B)\big), \tag{7.64c}$$
$$E_{\text{west}} = \big((\vdash B, \bot B), (\vdash B, \top B)\big). \tag{7.64d}$$

Notwithstanding the theoretical definition above, the corridors of these four edges cannot sprawl to infinity in practice. For calculating the free peripheral space, the corridors just need to go *beyond* the zone outline Z, no matter how far. For that purpose, as shown in image (c), it is suitable to define an auxiliary rectangular zone outline

$$Z' = \circledcirc_{\varepsilon}(⬚Z) \tag{7.65}$$

which is obtained by getting the bounding box of Z and enlarging it by an arbitrarily small positive number $\varepsilon > 0$. Then, the four corridors of $\mathcal{K}$ –depicted in (d)– can be defined as

$$K_{\text{north}} = \big((\vdash B, \top B), (\dashv B, \top Z')\big), \tag{7.66a}$$
$$K_{\text{east}} = \big((\dashv B, \bot B), (\dashv Z', \top B)\big), \tag{7.66b}$$
$$K_{\text{south}} = \big((\vdash B, \bot Z'), (\dashv B, \bot B)\big), \tag{7.66c}$$
$$K_{\text{west}} = \big((\vdash Z', \bot B), (\vdash B, \top B)\big). \tag{7.66d}$$

If $\mathcal{P}$ represents the set of all participants, then the set of obstacles $\mathcal{U}$ containing the complement of Z and the bounding boxes of all participants expect P is given by

$$\mathcal{U} \;=\; \overline{Z} \sqcup \square\!\square(\mathcal{P} - P) \tag{7.67}$$

and allows to determine the obstacles' intersections with each individual corridor via

$$\mathcal{U}_{\text{north}} = K_{\text{north}} \sqcap \mathcal{U}, \tag{7.68a}$$
$$\mathcal{U}_{\text{east}} = K_{\text{east}} \sqcap \mathcal{U}, \tag{7.68b}$$
$$\mathcal{U}_{\text{south}} = K_{\text{south}} \sqcap \mathcal{U}, \tag{7.68c}$$
$$\mathcal{U}_{\text{west}} = K_{\text{west}} \sqcap \mathcal{U}, \tag{7.68d}$$

as indicated by the criss-cross hatching in image (e). With the bounding boxes around these intersections –lightly grayed in (e)–, the four bounding box edges that face towards P demarcate the free peripheral space S_P, which can thus be perceived by evaluating the expression

$$S_P = \Big(\big(\dashv(\square\mathcal{U}_{west}), \top(\square\mathcal{U}_{south})\big), \big(\vdash(\square\mathcal{U}_{east}), \perp(\square\mathcal{U}_{north})\big)\Big) \tag{7.69}$$

and therefore yields the free peripheral space S_P illustrated in image (f) of Figure 7.28.

7.3.2.2 Pervasion (Obstacles in the Free Peripheral Space)

Although the given definition of free peripheral space is quite convenient, it may occur that the free peripheral space is not really "free" for the reason of being pervaded by an obstacle. This situation is denoted as *pervasion* and can be divided into two different cases:

- The first case is illustrated in Figure 7.29. Image (a) shows a participant P being overlapped by another participant P'. This means that P is in a state of *interference* (see Section 7.3.1.1), and therefore the overlapping part of P' intersects the free peripheral space of P. The situation is similar if the obstacle is not another participant but the complement of the layout zone. As can be seen in image (b), participant P may be in a state of *protrusion* (see Section 7.3.1.3) where the part of P inside Z is not rectangular. In this situation, the bounding box B around that part –and thus, the free peripheral space as well– also protrudes beyond the given layout zone. So, in both Figure 7.29 (a) and Figure 7.29 (b) the free peripheral space is not truly vacant due to the impaired *condition* of the participant.
- The second case is depicted in Figure 7.30, which points out the fact that the free peripheral space can contain up to four so-called *blind spots*. These blind spots are comprised of $S_P \lrcorner \mathcal{K}$ (i.e., those parts of the free peripheral space outside the four corridors K_{north}, K_{east}, K_{south}, K_{west}). Obstacles in a blind spot, such as participant P' in image (a), are accidentally overlooked when P perceives its free peripheral space. As above, a similar situation arises when the obstacle is the complement of the layout zone. Unless the zone outline is strictly rectangular, the free peripheral space can inadvertently exceed the bounds of the layout zone as in the example of image (b). So, in both Figure 7.30 (a) and Figure 7.30 (b) the free peripheral space is not entirely vacant because of its blind spots.

Luckily, neither of these cases is problematic, since eventual situations of interference or protrusion will be detected before a participant performs an action. Furthermore, it should be noted that blind spots become less and less troublesome throughout the course of a SWARM run: due to the successive tightening of Z, the arrangement of participants becomes increasingly compact and thus the area of blind spots approaches zero. This observation is visualized in Figure 7.31. In the initial constellation (a), there is a high degree of pervasion: almost the entire layout zone is littered with blind spots, such that every single participant is overlooked by at least one of its neighbors. As shown by the intermediate constellation of (b), where only five out of the nine participants pervade another participant's free peripheral space, the degree of pervasion becomes smaller throughout the self-organization flow. In the final constellation (c),

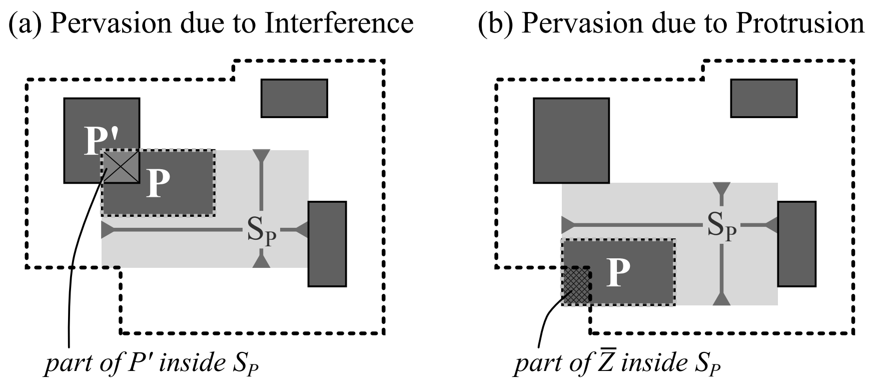

(a) Pervasion due to Interference
(b) Pervasion due to Protrusion
P'
P
S_P
part of P' inside S_P
P
S_P
part of Z̄ inside S_P

Figure 7.29: Pervasion of a participant's free peripheral space due to its bad condition.

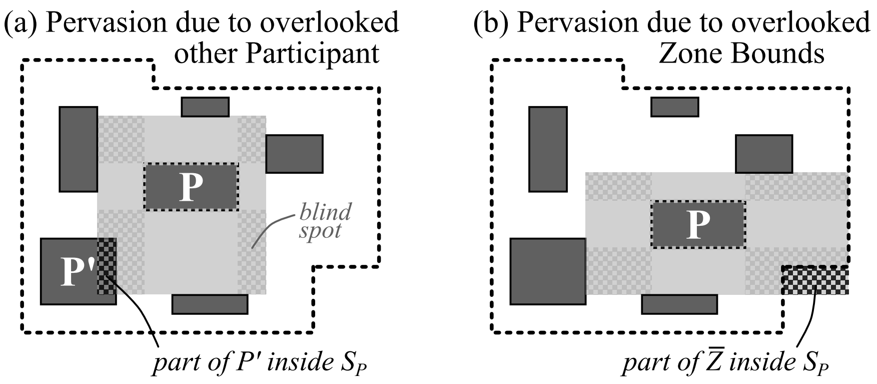

(a) Pervasion due to overlooked other Participant
(b) Pervasion due to overlooked Zone Bounds
P
blind spot
P'
part of P' inside S_P
P
part of Z̄ inside S_P

Figure 7.30: Pervasion of a participant's free peripheral space due to blind spots.

the arrangement is so compact that no pervasion at all is encountered in the relatively small areas of the remaining blind spots.

In addition to the above two cases, one can identify a third case, in which the idea of free peripheral space becomes completely moot. That is, if a participant has become *lost* (again see Section 7.3.1.3), S_P entirely lies beyond the given layout zone. In this case, a dedicated *Re-entering* action, which does not rely on the participant's free peripheral space, is used to hurl the participant back into the layout zone as will now be described in Section 7.3.3.

7.3.3 Exploration and Evaluation of Possible Actions

Every action that a participant can perform is basically a set of transformations for all participants $\mathcal{P}_\iota$ *involved* in the action. As introduced in Section 7.2.3.3, each transformation T is a triple $T = (M, R, D)$ that may comprise a movement M, a rotation R, and a deformation D:

- The movement M is a two-dimensional vector $M = (\Delta x, \Delta y)$ that displaces the participant without altering its aspect ratio. To obtain another aspect ratio, a rotation or a deformation need to be included in the transformation.
- The rotation R either rotates the participant around its center point or preserves its orientation. If the participant can be treated as a "black box" during the self-organization phase of the SWARM run, then it suffices to consider rotations $R \in \{0°, 90°\}$ which either keep or invert the participant's aspect ratio.
- The deformation $D \in \{V_1, V_2, \ldots, V_d\}$ refers to one of the d layout variants that the participant can assume (as given by its cumulative variability $\widetilde{\mathcal{V}}$). The higher the ductility d is, the more variants are available for potential deformations. A deformation always changes the participant's aspect ratio to that of the assumed layout variant but does not dislocate its center point.

Depending on the prospective aspect ratio, the participant's next measure is to spot an assortment of possible locations it might eventually decide to head for. Serving this purpose, different kinds of actions are

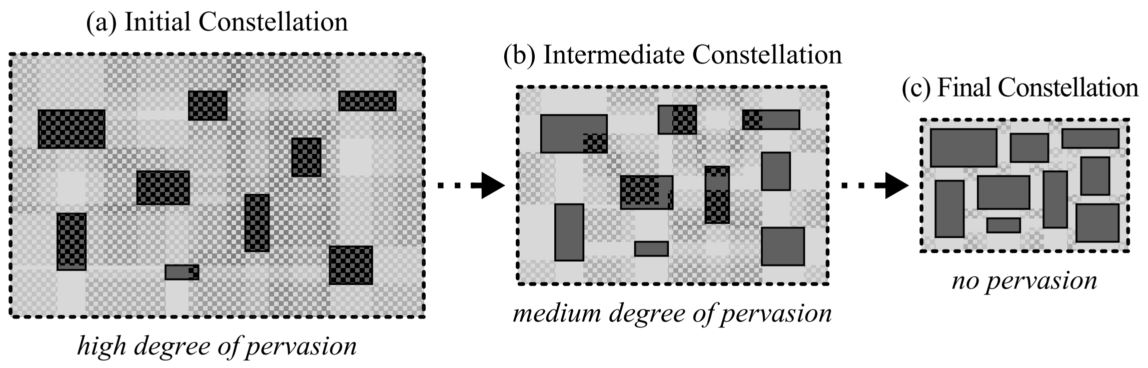

(a) Initial Constellation
(b) Intermediate Constellation
(c) Final Constellation
high degree of pervasion
medium degree of pervasion
no pervasion

Figure 7.31: Successive decline of free peripheral space pervasion caused by blind spots.

at hand, including SWARM's catalog of native actions (Section 7.3.3.1), constraint-specific custom actions (Section 7.3.3.2), and a special kind of functionality to account for full variability (Section 7.3.3.3).

Every action that the participant explores, also has to be evaluated. First and foremost, this is necessary in order to see whether an action is *valid* at all.

Definition 7.4. *An action is* valid *if it leads the participant into a location that is devoid of* protrusion*,* wounds *and* noncompliance*. Otherwise the action is* invalid*. If the participant currently is in an invalid location, then an invalid action that somehow improves the participant's condition (as done by the Re-entering action described below) is called* tolerable*. If an action is exacted by a constraint (such as the Imitation action in Section 7.3.3.2), the action is* mandatory*. An action is defined as* acceptable *if it is valid, tolerable, or mandatory.*

Actions that are not acceptable can be immediately discarded. Among the acceptable actions, the respective evaluation then allows to compare the actions' prospective *interference* and *turmoil*. This decides, which action will be finally performed, if an action is performed at all (Section 7.3.4).

7.3.3.1 Native Actions

Although particular design requirements may necessitate dedicated actions (such as the already mentioned *Imitation* that will be discussed in Section 7.3.3.2), SWARM implements a fundamental catalog of nine different actions considered to be adequate as a kind of natural, "instinctive" demeanor. Regarding a participant P and a layout zone Z, each of these actions is subsequently discussed in a paragraph of its own, with each paragraph providing

- a description of the respective action,
- the instructions to put the action into practice,
- additional comments about the action in general, and
- an illustration that depicts the action in a demonstrative example.

As already mentioned, some actions only involve participant P whereas other actions may also involve further participants. In either case, participant P (i.e., the participant who decides about the action to be performed) is denoted as the *leading participant*.

It should again be emphasized that each of the following paragraphs concentrates on *how* an action is explored, but not *when* that action is explored nor when (if at all) it gets indeed performed within SWARM's common action scheme. These open questions will be answered in Section 7.3.4, covering the preference of actions as well as the overall flow of action execution.

Re-entering

Description: *Re-entering* is performed when participant P is *lost*. This action has the sole aim of catapulting P back into the given layout zone Z at the nearest possible location. *Re-entering* thus remedies protrusion, while disregarding all other *influencing factors* covered in Section 7.3.1.

Instructions:

(1) If participant P is lost, then for each edge E of the rectilinear layout zone Z, determine whether E is a horizontal edge or a vertical edge.

(2) If E is a horizontal edge, determine and memorize a purely vertical move by which P aligns its southern (if P is below E) or northern (if P is above E) edge with E. If E is a vertical edge, determine and memorize a purely horizontal move by which P aligns its western (if P is to the left of E) or eastern (if P is to the right of E) edge with E.

(3) If a memorized move would lead P into a state of protrusion, correct the move such that it leads P to an allegedly *safe* location. For a vertical move, this implies an offset to the right (if P is to the left of E) or to the left (if P is to the right of E). For a horizontal move, this implies an offset upwards (if P is below E) or downwards (if P is above E). Formally, one can say:
$\forall E \in Z, M_E = (\Delta x, \Delta y)$ with

$$\Delta x = \begin{cases} \max(\vdash E - \vdash(\square P), 0) + \min(\dashv E - \dashv(\square P), 0) & \Leftrightarrow \quad E \text{ is horizontal} \\ \max(x_E - \vdash(\square P), 0) + \min(x_E - \dashv(\square P), 0) & \Leftrightarrow \quad E \text{ is vertical} \end{cases}$$

$$\Delta y = \begin{cases} \max(y_E - \perp(\square P), 0) + \min(y_E - \top(\square P), 0) & \Leftrightarrow \quad E \text{ is horizontal} \\ \max(\perp E - \perp(\square P), 0) + \min(\top E - \top(\square P), 0) & \Leftrightarrow \quad E \text{ is vertical.} \end{cases}$$

(4) Sort the memorized moves by the Euclidean distance of the movements and let P perform the move with the smallest distance.

Comments: As will be discussed in Section 7.4, the tightening of the layout zone is supposed to be realized in a way such that no participant gets lost through a tightening. However, this is not that easy to accomplish in case the layout zone is a nonrectangular polygon. Keeping the implementation simple, and conceding that a participant can thus get lost, *Re-entering* is a convenient action to remedy such situations.
Of course, *Re-entering* actions can also be explored for participants that are only *prone* but not entirely lost. In some cases, such a move may turn out to be a better alternative than an *Evasion* action (described next). For that reason, *Re-entering* actions can also be encountered in SWARM runs where the layout zone is rectangular.

Illustration:

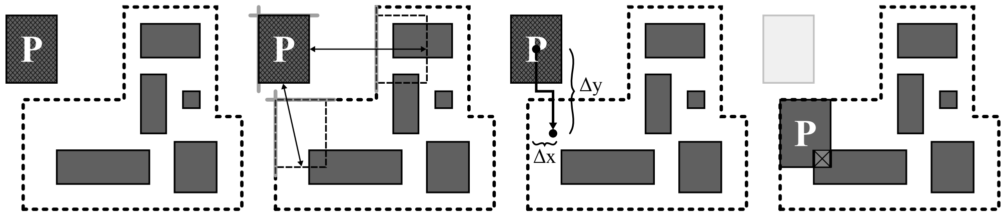


Figure 7.32: Exemplary visualization of a *Re-entering* move.

Evasion

Description: *Evasion* is only applicable when P is prone. With this action, P moves back into the layout area Z, evading other participants by going sideways.

Instructions:

(1) If P is prone, then perceive its free peripheral space S_P (based on the rectangular part B of P that lies inside Z, as described in Section 7.3.2).

(2) Get the width w_P and the height h_P of P. These quantities are relative to the coordinate system and can be formally calculated as follows:

$$w_P = \dashv(\square P) - \vdash(\square P),$$
$$h_P = \top(\square P) - \bot(\square P).$$

(3) Determine the three locations inside S_P by which P aligns its bounding box with the nearest edge or with one of the two nearest vertices of S_P. In case S_P lies to the right of P (as in the illustration below), the set of locations for potential moves is

$$\begin{aligned}\mathcal{L}_{\text{new}} = \Big\{ &\big(\vdash S_P + \tfrac{1}{2}w_P, \top S_P - \tfrac{1}{2}h_P\big), && \textit{// north-west of } S_P\\ &\big(\vdash S_P + \tfrac{1}{2}w_P, \tfrac{1}{2}(\bot S_P + \top S_P)\big), && \textit{// center-west of } S_P\\ &\big(\vdash S_P + \tfrac{1}{2}w_P, \bot S_P + \tfrac{1}{2}h_P\big) \Big\}. && \textit{// south-west of } S_P\end{aligned}$$

For the cases in which S_P lies above, below, or to the left of P, the set of locations is to be determined analogously.

(4) Let P move such that its center point lies at one of the new locations $\mathcal{L}_{\text{new}}$. As is the case with most of the actions that follow, it is up to the participant's decision-making which one of the explored actions will eventually be performed (see Section 7.3.4). Thus, the subsequent action descriptions are only meant to expound the principal idea behind each action, regardless of the question which action will finally be chosen.

Comments: If P protrudes Z at one of its convex vertices (not at one of its edges as shown in the example below), i.e., if $w_B < w_P \wedge h_B < h_P$, then S_P does neither definitely lie above, nor below, nor to the left, nor to the right of P. In that case, the participant should try to align its bounding box with the nearest vertex and with one of the two nearest edges of S_P.
If Z is not a rectangular layout zone, it can happen that P protrudes Z at a concave vertex. This represents a case of pervasion due to protrusion (as depicted in Figure 7.29). To address such situations, the *Evasion* action would have to be enhanced, but for convenience it is also possible to skip the *Evasion* attempt in favor of one of the other actions which may just as well help P to get back into the layout zone.

Illustration:

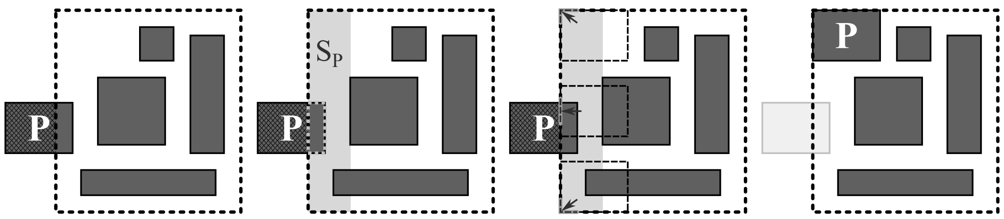


Figure 7.33: Exemplary visualization of a *Evasion* move.

Centering

Description: *Centering* is an elementary action where P aligns its center point with the center point of its free peripheral space. As such, *Centering* is not necessarily meant to solve conflict situations, but rather to balance the distances between participants when they are *contented* (also see the remark at the end of Section 7.3.1).

Instructions:

(1) Perceive P's free peripheral space S_P (as explained in Section 7.3.2).
(2) Determine the center point of S_P to go for it as the new location L_{new}:

$$L_{\text{new}} = \big(\tfrac{1}{2}(\vdash S_P + \dashv S_P), \tfrac{1}{2}(\perp S_P + \top S_P)\big).$$

(3) Let P move such that its center point lies at the new location L_{new}.

Comments: A mandatory concern here and in general is the definition of a *minimal movement distance* m to prevent infinitesimal actions. This is not only for reasons of performance, but to prevent a group of contented participants from *Centering* themselves ad infinitum. As the current implementation of SWARM shows, it is feasible to correlate m with the amount of free space in Z such that it changes dynamically during the self-organization.

Illustration:

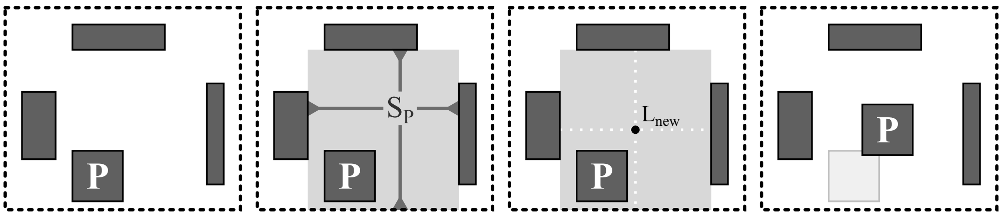


Figure 7.34: Exemplary visualization of a *Centering* move.

Lingering

Description: *Lingering* occurs if P is contented but cannot perform a *Centering* (e.g., due to prospective interference with an obstacle in a *blind spot*). In that case, P deliberately does nothing but to stay where it is, waiting for the next round of interaction.

Instructions:

(1) The situation is expected to be such that P is contented and attempts to perform a *Centering*.
(2) Determine if P would become discontented through the *Centering*.
(3) If P would become discontented, let P stay at its current location.

Comments: It should again be emphasized that *Lingering* is an action where the participant remains idle intentionally, as opposed to the case where a participant cannot perform an action although it would like to do so for the reason of being discontented.

Illustration:

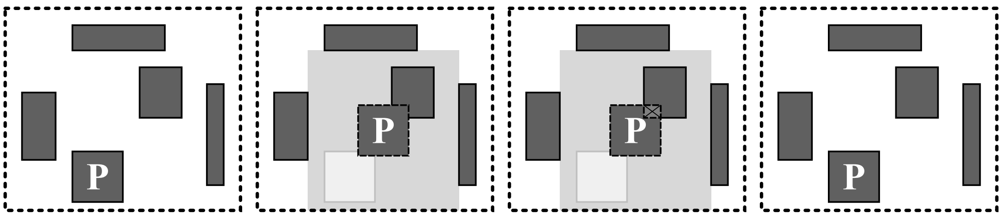


Figure 7.35: Exemplary visualization of a *Lingering* move.

Budging

Description: *Budging* makes P spot additional vacant room by perceiving the four free peripheral spaces from the viewpoints of its four corners. Then, P tries to slip into an appropriate location within that "secondary" free peripheral space.

Instructions:

(1) For each vertex of P's bounding box, perceive the secondary free peripheral space from the viewpoint of the vertex (with infinitely narrow *corridors* emanating from the vertex).

(2) Find a suitable location L_{new} inside S_P^{NE} (free peripheral space from the viewpoint of the north-eastern vertex), S_P^{SE} (of the south-eastern vertex), S_P^{SW} (of the south-western vertex), or S_P^{NW} (of the north-western vertex). One basal location is the center point of the free peripheral space (see comments below).

(3) Let P move such that its center point lies at the new location L_{new}.

Comments: Of course, vertices that lie inside another participant's bounding box, can be discarded right off the bat. For the other vertices, several locations inside the respective free peripheral space (in addition to its center point) might be probed as suitable targets. The specification, which of these targets are to be explored, is denoted as an *exploration plan* (also see the formal definition in Section 7.3.4.2).

For example, an exhaustive exploration plan would be letting P try to align its center point, its northern edge, north-eastern vertex, eastern edge, south-eastern vertex, southern edge, south-western vertex, western edge, and north-western vertex with the corresponding element of the free peripheral space. However, in the current implementation of SWARM, only the center point of the participant's free peripheral space is taken into consideration as the potential new location for P.

This is prompted by the desire to find an adequate trade-off for a participant's efforts on the local level. On the one hand, it is important to explore a rich fund of diverse actions that the participant may then choose from. On the other hand, a participant should refrain from investing too much labor into exploring a single kind of action, because the impairments of a participant's condition (e.g., interference or protrusion) are ultimately remediated by the totality of all participants' actions across several tightening-settlement cycles.

Balancing the three different core concepts (to bring the variability of the *responsive modules*, the possibilities of the *module interaction*, and the tightening conduct of the *interaction control* organ into a well-adjusted flow of self-organization) is a particular necessity but also a powerful setscrew of the SWARM methodology.

Illustration:

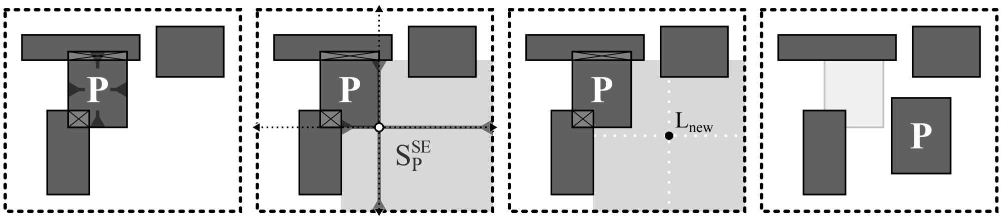


Figure 7.36: Exemplary visualization of a *Budging* move.

Swapping

Description: *Swapping* lets P trade places with another participant P'. That is, P jumps into the free peripheral space of P' and P' in turn jumps into the free peripheral space of P.

Instructions:

(1) Perceive the free peripheral space S_P of P and, considering another participant P', perceive the free peripheral space $S_{P'}$ of P'.

(2) Determine if a *Swapping* with P' should be explored. In that case, the *Swapping* is said to be *promising* (see comments below).

(3) If a *Swapping* appears to be promising, determine the center point of $S_{P'}$ as the new location L_{new} for P, and determine the center point of S_P as the new location L'_{new} for P':

$$L_{\text{new}} = \left(\tfrac{1}{2}(\vdash S_{P'} + \dashv S_{P'}), \tfrac{1}{2}(\bot S_{P'} + \top S_{P'})\right),$$
$$L'_{\text{new}} = \left(\tfrac{1}{2}(\vdash S_P + \dashv S_P), \tfrac{1}{2}(\bot S_P + \top S_P)\right).$$

(4) Let P move with its center point to the new location L_{new} and pull P' with its center point to the new location L'_{new}.

Comments:

In general, it is worthwhile to explore *Swappings* with all other participants. But as indicated above, it might be feasible to elide a presumably futile *Swapping* to save computation time – according to a specific heuristic. As an example for such a heuristic, one may try to make a prediction by comparing the size of P and S_P with the size of P' and $S_{P'}$, and then only explore that *Swapping* if the deviation regarding size does not exceed a certain proportion.

An action worth being explored according to such a –or a similar– heuristic is qualified as promising. Here, it must be clearly understood, that the adjective *promising* approves the exploration of an action, not the execution of that action (which is authorized by the predicate *acceptable*, as specified in Definition 7.4). The postulated saving in computation time results from the fact, that comparing two sizes is computationally less expensive than having to assess the prospective conditions of the two participants (if the action is indeed explored and therefore also has to be evaluated – considering all influencing factors).

Unfortunately, it is not trivial to find an appropriate rule of thumb that reliably tells whether a *Swapping* would probably be useful or not. Experiments show that surprisingly often P opts for a *Swapping* that would otherwise have been rejected by a heuristic such as comparing sizes (or aspect ratios, or both, for that matter). On these grounds, the current implementation of SWARM by default explores all possible *Swappings* with all other participants. So, the issue of finding an adequate heuristic is worth being kept in mind and may be examined in future research, but is not broached any further within the scope of this thesis.

Another remark should be made about the fact that a *Swapping* need not necessarily imply that the two participants *center* themselves within the free peripheral space of their counterpart. Instead, the participants might pursue a more exhaustive exploration plan as also discussed in the comments of the *Budging* action above.

Illustration:

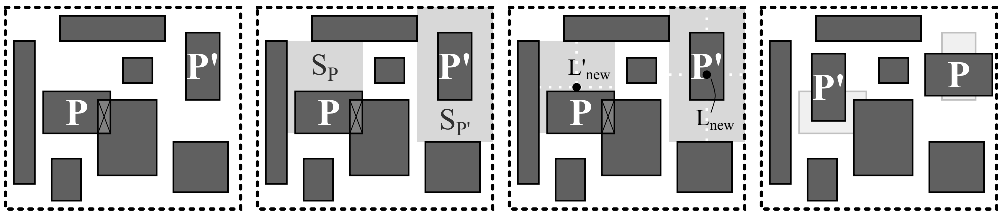


Figure 7.37: Exemplary visualization of a *Swapping* move.

Pairing

Description: *Pairing* means that P jumps next to another participant P', thereby pushing P' aside such that both participants then share the free peripheral space of P'.

Instructions:

(1) Considering another participant P', determine if a horizontal *Pairing*, a vertical *Pairing*, or both (or none) are promising and should therefore be explored.

(2) For a horizontal *Pairing*, get the width w_P of P and the width $w_{P'}$ of P'. For a vertical *Pairing*, get the height h_P of P and the height $h_{P'}$ of P'.

(3) For a horizontal *Pairing*, determine a horizontal movement where P' is pushed to the right, and a horizontal movement where P' is pushed to the left (such that P and P' will be centered around the vertical symmetry axis of P' after the *Pairing*). The respective Δx is

$$\begin{aligned} \Delta x &= -\tfrac{1}{2}w_{P'} + \tfrac{1}{2}(w_P + w_{P'}) = \tfrac{1}{2}w_P \\ \text{and} \quad \Delta x &= \tfrac{1}{2}w_{P'} - \tfrac{1}{2}(w_P + w_{P'}) = -\tfrac{1}{2}w_P \end{aligned}$$

such that the potential movements $M_{P'}$ for P' can be shortly written as

$$M_{P'} = (\pm\tfrac{1}{2}w_P, 0).$$

With a vertical *Pairing*, the potential movements for an upwards-push and a downwards-push can be equivalently given as

$$M_{P'} = (0, \pm\tfrac{1}{2}h_P).$$

(4) Calculate the new location L_{new} for P accordingly, which can be done in a way that is analogous to the calculation of $M_{P'}$ above. For a horizontal *Pairing*, the respective locations accompanying a rightwards-push and a leftwards-push can thus be compactly expressed as

$$L_{\text{new}} = \Big(\tfrac{1}{2}\big(\vdash(\square P') + \dashv(\square P')\big) \mp \tfrac{1}{2}w_{P'}, \tfrac{1}{2}\big(\bot(\square P') + \top(\square P')\big)\Big)$$

while the respective locations for a vertical *Pairing* can be calculated via

$$L_{\text{new}} = \Big(\tfrac{1}{2}\big(\vdash(\square P') + \dashv(\square P')\big), \tfrac{1}{2}\big(\bot(\square P') + \top(\square P')\big) \mp \tfrac{1}{2}h_{P'}\Big).$$

(5) Push P' by $M_{P'}$ and let P move with its center point to the new location L_{new}.

Comments: In the basic form of *Pairing*, as described above and as currently implemented in SWARM, participant P aligns its vertical (horizontal) center with the vertical (horizontal) center of P' when performing a horizontal (vertical) *Pairing*. Additionally, it would also be possible to follow a more comprehensive exploration plan and let P explore further options of alignment, where P aligns its northern or southern edge with the corresponding edge of P' (for a horizontal *Pairing*), and likewise its western or eastern edge (for a vertical *Pairing*).
On the other hand, as indicated above, the participant might dismiss the exploration of a *Pairing* if it does not look promising at first sight (based on a heuristic such as a comparison of the participants' aspect ratios, for example). This idea has also been discussed in the comments of the *Swapping* action, with the mentioned caveats unfortunately also applying here to a similar extent. Therefore, the participants in the current implementation of SWARM by default decide that all possible *Pairings* are promising (and thus worthy of exploration). Still, the idea of not exploring certain actions based on some heuristic to save computation time might be investigated in the future.

Illustration:

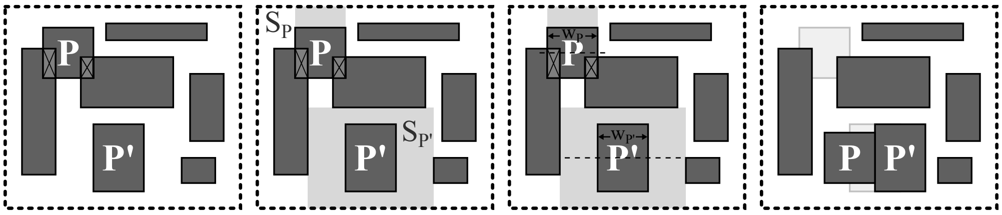


Figure 7.38: Exemplary visualization of a *Pairing* move.

Hustling

Description: *Hustling* is an action by which P remains where it is, but pushes away all other participants that currently overlap P. These other participants are pushed as far as necessary such that P gets rid of their interference and becomes *clear* again. So far, *Hustling* is the only action that may involve more than two participants.

Instructions:

(1) For every partipant P' that overlaps P, measure the width $w'_\sqcap$ and the height $h'_\sqcap$ of the overlap region $\square P \sqcap \square P'$.

(2) For each of these participants, check whether the horizontal extent of the overlap is larger than the vertical extent. In the first case, determine a vertical movement to get rid of P'. In the second case, determine a horizontal movement for that purpose. If $\mathcal{P}'$ denotes the set of participants that overlap P, one can formally say: $\forall P' \in \mathcal{P}'$ the respective movement $M_{P'}$ is

$$M_{P'} = \begin{cases} (0, h'_\sqcap) \text{ or } (0, -h'_\sqcap) & \Leftrightarrow \quad w'_\sqcap > h'_\sqcap \\ (w'_\sqcap, 0) \text{ or } (-w'_\sqcap, 0) & \Leftrightarrow \quad \text{otherwise} \end{cases}$$

where the polarity of the pushing depends on whether P' overlaps P from the northern ($\Rightarrow h'_\sqcap$), southern ($\Rightarrow -h'_\sqcap$), eastern ($\Rightarrow w'_\sqcap$), or western ($\Rightarrow -w'_\sqcap$) side of P.

(3) Push every overlapping participant P' away via the respective $M_{P'}$.

Comments: Similar to some of the other actions discussed so far (i.e., *Budging*, *Swapping*, *Pairing*), a more exhaustive exploration plan might be pursued here. For example, P might try both the horizontal and the vertical movement above, regardless of the overlap's dimensions. In addition, P may even attempt to exert a movement $M_{P'} = (w'_\sqcap, h'_\sqcap)$ such that P' is pushed away both horizontally and vertically. In its current implementation however, SWARM sticks to the instructions above, exploring either a vertical movement or a horizontal movement for every overlapping participant P', but not both. As with the other actions mentioned at the beginning of this comment, this is done to keep the participant's exploration effort rather low for this particular kind of action.

There are situations, where P does not become completely *clear* by performing a *Hustling*, for example when P is *properly enclosed* by another participant P' (i.e., $P \sqsubset P'$). Generally speaking, this is the case if P pushes P' away in a direction where P' juts beyond P on *both* sides of P. However, such a situation is not that problematic since it does not jeopardize the entire *Hustling* action: despite the remaining overlap, the *Hustling* may still be better than all other actions explored by P.

Illustration:

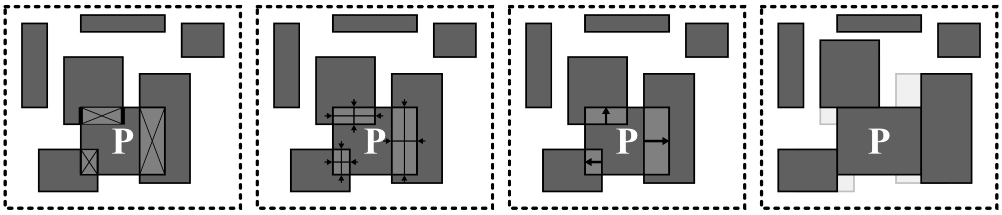


Figure 7.39: Exemplary visualization of a *Hustling* move.

Yielding

Description: *Yielding* is only done if P suffers from interference but cannot find an appropriate action. In this case, P determines the polygon Y_P (the so-called *yielding region*, which circumscribes the part of P inside layout zone Z excluding the overlap regions) and aligns its center with the geometric centroid (i.e., the barycenter) of Y_P.

Instructions:

(1) If P is in a state of interference, determine the polygon Y_P. If $\mathcal{P}'$ denotes the set of participants that overlap P, Y_P can be formally obtained with

$$Y_P = ⬚P \urcorner (\overline{Z} \sqcup ⬚\mathcal{P}').$$

(2) Find the geometric centroid of the yielding region Y_P, which is to be used as the new location L_{new} for P. Calculating the signed area of Y_P via

$$[Y_P]^{\pm} = \frac{1}{2} \sum_{i=1}^{n} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

where $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ are the vertices along the contour of Y_P (with $x_{n+1} = x_1$ and $y_{n+1} = y_1$), the centroid of Y_P is given by the coordinates

$$x_c = \frac{1}{6\,[Y_P]^{\pm}} \sum_{i=1}^{n} (x_i + x_{i+1})\,(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i),$$
$$y_c = \frac{1}{6\,[Y_P]^{\pm}} \sum_{i=1}^{n} (y_i + y_{i+1})\,(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i).$$

(3) Let P move with its center point to the new location $L_{\text{new}} = (x_c, y_c)$.

Comments: The name of this action suggests that the participant abandons itself to some sort of makeshift maneuver. Although this can not be entirely disavowed (since indeed no helpful action could be found so far by the participant), *Yielding* has a rightful place among the catalog of native actions and should not be frivolously repudiated, because a *Yielding* often has a reviving effect on the self-organization flow when stuck in a period of stagnation.
One reason for this is that a yielding participant ignores all influencing factors apart from protrusion (like in a *Re-entering*). Another reason is that *Yielding* typically leads to interference with all neighboring participants that P is currently oppressed by (as in the example below). In many cases, this animates the neighbors to move away a bit instead of continuing to shove P around in endless repetitive circles.

Illustration:

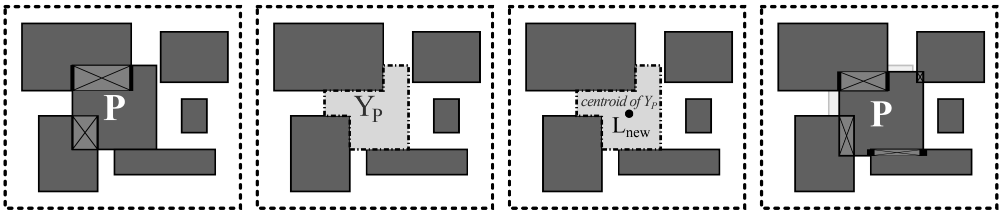


Figure 7.40: Exemplary visualization of a *Yielding* move.

Although each of the above illustrations depicts mere movement-based transformations, every kind of action can also involve a rotation and/or a deformation. During a participant's action exploration, this is basically taken into account by *first* determining the would-be aspect ratio and *then* spotting the new location for the participant. This proceeding is mandatory for those kinds of action, where the new location depends on the participant's aspect ratio, as in the *Evasion* example given by Figure 7.41 (a). On the other hand, there are also actions where the new location is independent from the aspect ratio, like with the *Centering* example shown in Figure 7.41 (b). For these actions, it is –of course– wise not to recalculate the new location again and again for all possible orientations and layout variants that the participant can assume.

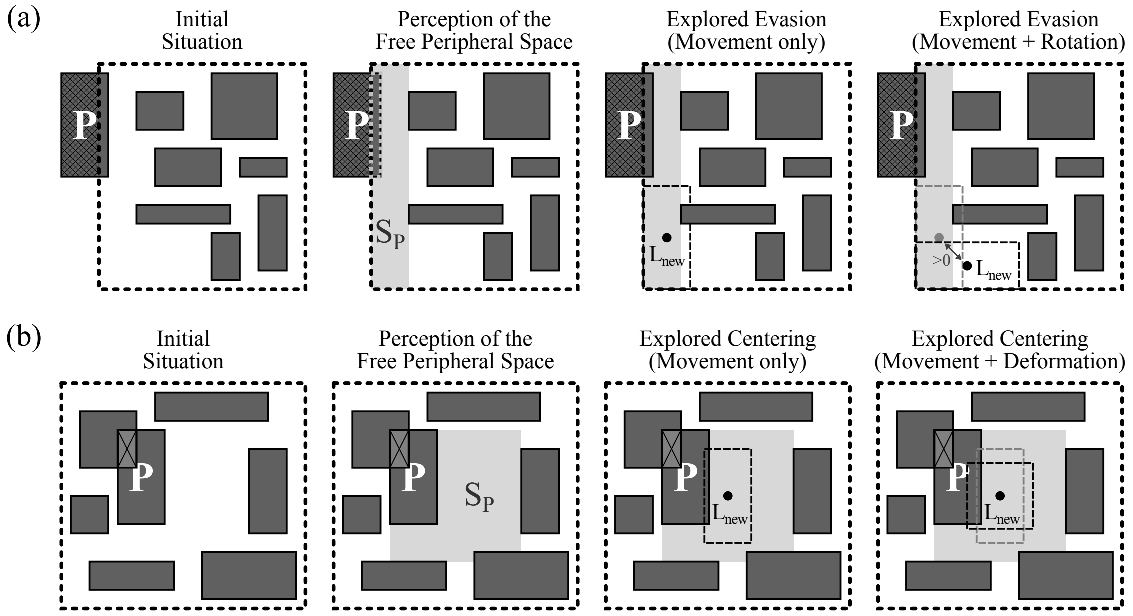

(a)
Initial Situation
Perception of the Free Peripheral Space
Explored Evasion (Movement only)
Explored Evasion (Movement + Rotation)
P
S_P
L_new
>0
(b)
Initial Situation
Perception of the Free Peripheral Space
Explored Centering (Movement only)
Explored Centering (Movement + Deformation)
P
S_P
L_new

Figure 7.41: The new location of a participant can be (a) dependent or (b) independent of its aspect ratio.

As described above, certain actions (e.g., *Evasion*) make the participant align itself with the layout zone Z, such that the participant is definitely *safe* afterwards. But with most of the other actions, the participant takes the risk of getting into a state of protrusion. While it is unlikely to get completely *lost* (which can only happen in the course of a *Pairing*), the participant must often face the chance of becoming *prone*. In such a case, the action need not be turned down, but can be corrected via the *protrusion extent* ψ introduced in Section 7.3.1.3. This is done by simply subtracting the horizontal and the vertical part of the protrusion extent from the respective element of the movement vector such that

$$M = (\Delta x - \psi_x, \Delta y - \psi_y). \tag{7.70}$$

However, one must pay attention here: if ψ_x and ψ_y are not signed (since equation 7.53 returns absolute values), the action correction above only works if the participant protrudes Z in upward or rightward direction. Otherwise, the respective protrusion extent has to be *added* to Δx and Δy, not subtracted. An example of an action correction is presented in Figure 7.42. Initially, P and P' both suffer from interference with other participants. After perceiving the free peripheral spaces S_P and $S_{P'}$, P explores a potential *Swapping* with P'. Originally, the action would lead both participants into a state of protrusion, but when adjusting the movements by $-\psi_y$ and ψ_x respectively, the prospective situation is *safe* for P and P'. Although the action correction again entails some overlaps with other participants, the *Swapping* might still be performed because the overall interference is reduced thereby.

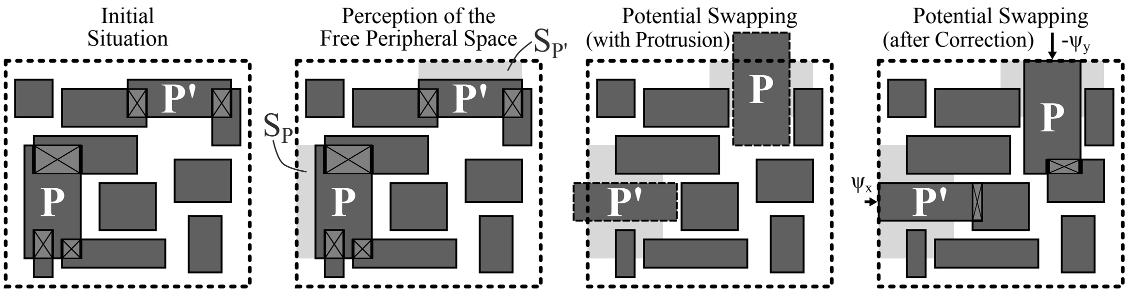

Initial Situation
Perception of the Free Peripheral Space
Potential Swapping (with Protrusion)
Potential Swapping (after Correction)
P'
P
S_P
S_P'
-ψ_y
ψ_x

Figure 7.42: Action correction example: P's move is corrected downwards, the move of P' rightwards.

To sum it up, Table 7.7 provides an overview that lists SWARM's catalog of native actions. For each action, the table indicates how many participants would be moved (i.e., transformed) thereby, under which condition of the *leading participant* the action is explored, under which prospective condition of the *involved* participants the action is deemed to be *acceptable*, and whether this predicate stems from the fact that the action is *valid* or *tolerable*. The third possibility according to Definition 7.4 –the action being *mandatory*– so far only applies to custom actions, which will now be discussed in Section 7.3.3.2.

Table 7.7: Overview of SWARM's catalog of native actions.

Action	Number of moved Participants	Explored if leading Participant P is	Acceptable if all involved Participants will be	Action is
Re-entering	1	lost	safe	tolerable
Evasion	1	prone	safe & healthy & compliant	valid
Centering	1	not lost	safe & healthy & compliant	valid
Lingering	0	contented	contented	valid
Budging	1	safe	safe & healthy & compliant	valid
Swapping	2	safe	safe & healthy & compliant	valid
Pairing	2	safe	safe & healthy & compliant	valid
Hustling	$1 \ldots n$	safe & not clear	safe & healthy & compliant	valid
Yielding	1	not lost & not clear	safe	tolerable

7.3.3.2 Custom Actions

Depending on the layout problem at hand, it may occur that SWARM's catalog of native actions is not adequate when a participant needs to satisfy some particularly rigorous design constraints. In such cases, it can be advisable to equip the participant with its own particular behavior by implementing some custom action that the participant is to pursue. This idea will now be elucidated by means of the *Imitation* action that was already brought up in Section 7.3.1.5.

For matching reasons, it may be demanded that a participant behaves symmetrically to another participant, such that (1) the two participants have equal dimensions, that (2) they are aligned in one direction, and that –in the orthogonal direction– (3) they are kept at the same distance on either side of a certain axis. There is no need to point out that the odds of meeting all these demands via SWARM's native actions are quite bad. An alternative is to let the two participants be managed by a superordinate governing module such as the Symmetric Pair known from Figure 7.10. However, that module is meant to keep its two submodules closely together with a certain fixed space in between. This approach would be infeasible if there was another module between the two participants, the more so when that module changes its aspect ratio throughout the self-organization. This is where the *Imitation* action comes to the rescue.

Imitation

Description: *Imitation* is a custom action by which a participant P mimics another participant's transformations. The *Imitation* action can thus be employed to satisfy a Symmetry constraint (see Figure 3.3 in Section 3.1.1.2). Since an *Imitation* action is irrevocable, it does not pay respect to the *influencing factors* described in Section 7.3.1.

Instructions:

(1) If P is to behave in horizontal or vertical symmetry to another participant P', get the location $L' = (x_{P'}, y_{P'})$, which is supposed to be the center point of P':

$$L' = \Big(\frac{1}{2}\big(\vdash(\square P') + \dashv(\square P')\big), \frac{1}{2}\big(\bot(\square P') + \top(\square P')\big)\Big).$$

(2) Calculate the new location L_{new} for participant P. If x_{sym} and y_{sym} denote the horizontal and vertical coordinate of the specified iso-oriented symmetry axis respectively, the new location can be determined via

$$L_{\text{new}} = \begin{cases} (2\,x_{\text{sym}} - x_{P'}, y_{P'}) & \Leftrightarrow \quad \text{horizontal symmetry} \\ (x_{P'}, 2\,y_{\text{sym}} - y_{P'}) & \Leftrightarrow \quad \text{vertical symmetry.} \end{cases}$$

(3) Get the dimensions $(w_{P'}, h_{P'})$ of P' with the width $w_{P'} = \dashv(\square P') - \vdash(\square P')$ and the height $h_{P'} = \top(\square P') - \bot(\square P')$.

(4) Make P assume another layout variant that conforms to the dimensions $(w_{P'}, h_{P'})$ and let P move with its center point to the new location L_{new}.

Comments: In what way P is able to assume another layout variant that comes up to the dimensions of P', depends on the *variability* of P (see Section 7.2.4). If P has only *discrete variability*, but is –as a layout module– constructed identically to P', then P can simply take over the parameters of P'. If P and P' are constructed differently, some additional savvy might be necessary to help P choose the layout variant that at least comes closest to that of P'.

The instructions above assume that P has *full variability* (also see the next Section 7.3.3.3) and can take on dimensions in a continuous range. In this case, the purpose of the *Imitation* action is to match P with P' not (or not only) for the sake of P or P', but to achieve overall symmetry in the layout constellation as a whole – in this manner enforced *explicitly*.

As indicated above (and illustrated below), the *Imitation* action allows to leave some mutable room for other participants scrimmaging between P and P' – a feature that could not be adequately effectuated by using a superordinate governing module instead (in particular with respect to the consideration that this mutable room dynamically adjusts itself to the in-between participants during the self-organization). Vice versa, the *Imitation* action does not facilitate interdigitation, which in turn is an easy job for a governing module (like the Quad shown in Figure 7.6, for example). Following a given interdigitation pattern, such a module can take care of corresponding placement constraints *implicitly*.

So, the ability to team suchlike modules (incorporating nonformalized expert knowledge) with custom actions like *Imitation* (embodying formalized expert knowledge) is a powerful competence of SWARM regarding the title of this thesis.

Illustration:

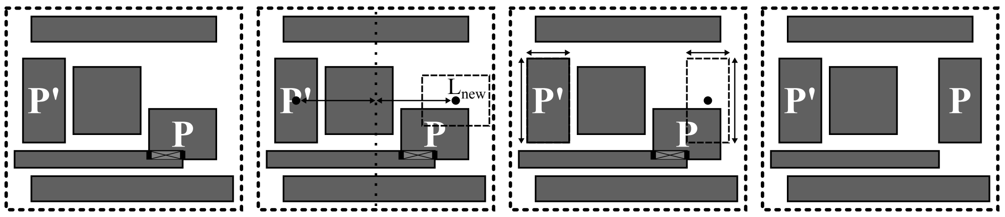


Figure 7.43: Exemplary visualization of a *Imitation* move.

So far, no coercive remarks have been made about the order in which the individual participants should act. But for custom actions, this order might be of exceptional relevance with regard to the efficiency of the self-organization. This is particularly evident in the *Imitation* action, where it is most reasonable to let participant P act as soon as P' has taken an action. Elsewise, all transformations performed by other participants in response to the action of P' might be more or less in vain, because they need to be remedied after P performs its irrevocable, ensuing *Imitation* move.

As the name indicates, custom actions are special-purpose actions and are thus not as generic as SWARM's catalog of native actions. Hence, it is not too much to ask that a participant provide its own custom actions as required. Still, a custom action –including the *Imitation* action above– may prove useful on such a regular basis that the action is deemed to be more universal than other custom actions. For future developments, it might therefore be worth a thought to include such actions into SWARM as extensions to the catalog of native actions. One action-related functionality, that has already been implemented in SWARM apart from the native actions covered in Section 7.3.3.1 addresses design components with full variability (as will now be described in Section 7.3.3.3).

7.3.3.3 Full Variability

If a participant covers multiple discrete layout variants, SWARM considers these via deformations during the action exploration, as has been discussed at the end of Section 7.3.3.1. The proceeding is different with participants that support full variability and are here denoted as *elastic*. An elastic participant P has a width w_P, a height h_P, and an area $[\square P] = w_P \cdot h_P$. It starts out with some initial dimensions w_P^{init} and h_P^{init}, but during the interaction P can change w_P and h_P into the new values w_P^{new} and h_P^{new} within the respective range

$$\check{w}_P \le w_P^{\text{new}} \le \underbrace{\frac{w_P\, h_P}{\check{h}_P}}_{=\hat{w}_P} \quad \text{and} \quad \check{h}_P \le h_P^{\text{new}} \le \underbrace{\frac{w_P\, h_P}{\check{w}_P}}_{=\hat{h}_P} \tag{7.71}$$

(where $\check{w}_P$ and $\check{h}_P$ allow to individually specify a minimal width and a minimal height for P, which also defines the maximum width $\hat{w}_P$ and the maximum height $\hat{h}_P$), such that the area of P remains unaltered:[9]

$$w_P^{\text{new}} \cdot h_P^{\text{new}} = w_P \cdot h_P. \tag{7.72}$$

Most of SWARM's native actions are geared towards some free peripheral space (e.g., for the *Centering* of a participant P, it is that of P; for the *Swapping* with another participant P', it is that of P'). If P is elastic, then P must be able to fit its aspect ratio to the dimensions (w_S, h_S) of such a free peripheral space S, thereby literally "pouring" into the vacant region. Thus, P's new layout variant is not one of a discrete set of layout variants $\{V_1, V_2, \ldots, V_d\}$, but rather a function of the free peripheral space that is currently available. With this function, five elastic deformation behaviors are explored per action, such that participant P either

[9] In reality, the area of such a layout component is usually not exactly constant for different aspect ratios, but only approximately – this is due to certain layout structures on the fringe. The smallest possible variant then is that of a square layout because the circumference-to-area quotient (and thus the portion of the fringe structures) reaches a minimum in that case.

(a) keeps its current aspect ratio,
(b) reassumes its initial aspect ratio,
(c) changes into a variant with square aspect ratio,
(d) fits its width to that of the free peripheral space (and adjusts its height),
(e) or fits its height to that of the free peripheral space (and adjusts its width).

These behaviors and the respective change of P's current width w_P and current height h_P into the new values w_P^{new} and h_P^{new} are listed in Table 7.8 and illustrated in Figure 7.44 by means of a *Centering* action for an exemplary, given situation.

Table 7.8: Listing of the five deformation behaviors explored by an elastic participant.

New Variant	**New Width**	**New Height**
(a) Current variant	$w_P^{\text{new}} = w_P$	$h_P^{\text{new}} = h_P$
(b) Initial variant	$w_P^{\text{new}} = w_P^{\text{init}}$	$h_P^{\text{new}} = h_P^{\text{init}}$
(c) Square variant	$w_P^{\text{new}} = \sqrt{w_P\, h_P}$	$h_P^{\text{new}} = \sqrt{w_P\, h_P}$
(d) Width-fitted	$w_P^{\text{new}} = \min(\max(w_S, \breve{w}_P), w_P h_P / \check{h}_P)$	$h_P^{\text{new}} = w_P\, h_P \,/\, w_P^{\text{new}}$
(e) Height-fitted	$w_P^{\text{new}} = w_P\, h_P \,/\, h_P^{\text{new}}$	$h_P^{\text{new}} = \min(\max(h_S, \check{h}_P), w_P h_P / \breve{w}_P)$

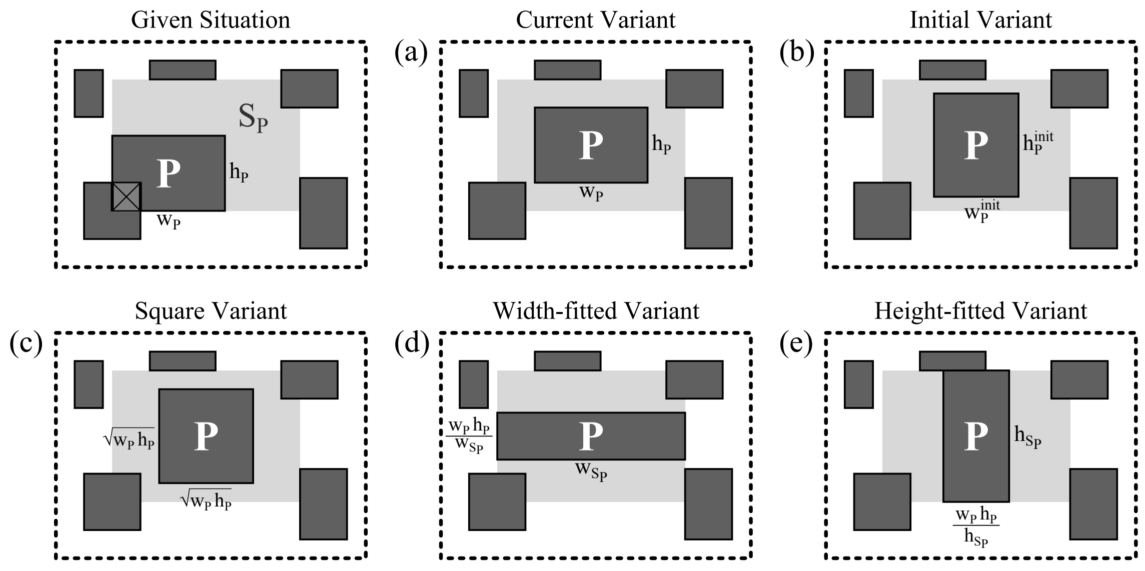


Figure 7.44: Illustration of the five deformation behaviors explored by an elastic participant.

The five deformation behaviors do not constitute an action of its own, but are automatically fathomed during a participant's exploration of SWARM's native actions in case the participant is elastic. Thereupon, the "static" traversal of discrete layout variants (including different orientations) during the action exploration is simply substituted by probing the five alternatives of "dynamic" deformation. As with SWARM's native actions, every elastic deformation has to be evaluated to check whether it represents a valid action or not. Otherwise, it could –for example– happen, that the participant P moves into an *unhealthy* location, overlapping a critical wound of another participant which penetrates a blind spot of P's free peripheral space.

7.3.4 Execution of the Preferred Action

After having explored and evaluated an assortment of potential actions, the fourth and final measure of SWARM's action scheme makes a participant choose and execute the most preferred one of these actions. To understand this decision-making, the following Section 7.3.4.1 illuminates by what judgment one action is preferred over another. In some cases, no such comparison has to be made because a satisfactory action can be performed right away, thus eliminating the need to investigate further alternatives. Such cases can best be comprehended from regarding the action execution in its entirety, which is therefore done in Section 7.3.4.2 and rounds out the current Section 7.3 in a synoptical fashion.

7.3.4.1 Action Preference

Comparing two actions to determine which one is preferable presupposes that both actions are *valid*. As already mentioned at the outset of Section 7.3.3, the preference then depends on two criteria: *interference* and *turmoil*. If no turmoil is involved (i.e., the connectivity is disregarded), then the comparison metric described in paragraph (1) suffices, otherwise the more elaborate comparison metric of paragraph (2) is employed. For the sake of convenience, an action is preliminarily considered as a single transformation T of only *one* participant P – this can be assumed without loss of generality. Afterwards, paragraph (3) discusses how the metrics can be generalized to address actions that involve multiple participants.

(1) Comparison Metric for Interference Only

Some layout problems elude the necessity to take distances between participants into consideration during the module interaction. For instance, this is the case if the locations of the participants in relation to each other have been predefined in advance via a *placement template* (in SWARM enacted by constraints, as in the examples of Section 8.3). Then, no turmoil needs to be accounted for, which simplifies the action comparison significantly: if two actions are given as a transformation T_a and a transformation T_b, with $\Upsilon_P^{\text{new}}(T_a)$ and $\Upsilon_P^{\text{new}}(T_b)$ denoting the prospective interference that the acting participant P will sustain when performing the corresponding action, then action T_a is preferred over action T_b if

$$\Upsilon_P^{\text{new}}(T_a) < \Upsilon_P^{\text{new}}(T_b). \tag{7.73}$$

In the case that $\Upsilon_P^{\text{new}}(T_a) > \Upsilon_P^{\text{new}}(T_b)$, then action T_b is preferred over action T_a. If the prospective interference of the two actions are equal, then the participant chooses the action with the lesser Euclidean distance of the involved translational movement $M_a = (\Delta x_a, \Delta y_a)$ or $M_b = (\Delta x_b, \Delta y_b)$. Therefore, if

$$\sqrt{(\Delta x_a)^2 + (\Delta y_a)^2} < \sqrt{(\Delta x_b)^2 + (\Delta y_b)^2} \tag{7.74}$$

then action T_a is preferred over T_b. If the distances of the movements are equal, the action that has been explored first, will be preferred over the other action. In that regard, it is worthwhile to think about the order in which the actions –and in particular the layout variants that the participant can assume– are to be explored. This order should not be arbitrary, since it gives some valuable control over the decision-making such that –for example– squarish layout variants are preferred over layout variants with more extreme aspect ratios (or vice versa, depending on the problem at hand), e.g., to achieve a better matching. However, a detailed investigation of this topic is beyond the scope of this thesis.

If the participant finds an action T with $\Upsilon_P^{\text{new}}(T) = 0$ (such that there will be no interference at all), then this action can be immediately performed without exploring further alternatives. If $\Upsilon_P^{\text{new}} > 0$ for *all* explored actions, then the comparison condition 7.73 makes P execute the action with the least prospective interference – but only if

$$\Upsilon_P^{\text{new}}(T) - \Upsilon_P < 0 \tag{7.75}$$

which means that the prospective interference $\Upsilon_P^{\text{new}}(T)$ must be less than P's present interference Υ_P.[10] Such an action is called *beneficial*. If $\Upsilon_P^{\text{new}}(T) = 0$, the action is denoted as *adjuvant*. In all these cases, no constraint (e.g., template-induced) is violated since only valid actions are considered, as said above.

If no beneficial action can be found, but P is discontented, then P performs a *Yielding*, even if it is not beneficial. As already mentioned in Section 7.3.3.1, *Yielding* leaves P in an unsatisfactory condition but has the effect that, if P is jammed in between multiple other participants $\mathcal{P}'$, then P will deliberately interfere with all of them. Thus every $P' \in \mathcal{P}'$ is provoked to move away such that P can again try to find a beneficial action on its next turn.

[10]It is not necessary to take the absolute value $|\Upsilon|$ of interference here (nor in equation 7.73), since it is always the case that $\Upsilon \geq 0$, which means that interference can never be negative.

(2) Comparison Metric for Interference and Turmoil

To take distances between participants into account as well, condition 7.73 has to be enhanced such that it also includes a participant's turmoil. The following definition explains how this enhancement leads to condition 7.77 and condition 7.78. In that regard, it should be recapitulated that the distance between two participants is modeled as a straight connection C, and that the connection is said to be *relaxed* if the length of C is below its *relaxation threshold* ϱ_C (as described in Section 7.3.1.2).

Definition 7.5. *For a participant P, the value ζ_P is defined as the number of its* unrelaxed connections *(also see Section 7.3.1.2). An action T changes ζ_P by the amount $\Delta\zeta_P(T)$, which is denoted as the so-called* relaxation delta. *If $\Delta\zeta_P(T) < 0$, then T represents a* relaxing action. *Two actions T_a and T_b are said to be* equally relaxing *if both are relaxing or if they have the same relaxation delta. This definition can be written in form of the following logical expression:*

$$\Delta\zeta_P(T_a) < 0 \wedge \Delta\zeta_P(T_b) < 0 \vee \Delta\zeta_P(T_a) = \Delta\zeta_P(T_b). \tag{7.76}$$

The idea of relaxation delta is illustrated in the example of Figure 7.45. Participant P, having a total of five connections ($C_1, C_2, \ldots, C_5$), is about to act. Initially (a), connections C_1, C_2 and C_3 are not relaxed (i.e., unrelaxed), while C_4 and C_5 are relaxed, which means that $\zeta_P = 3$. With the *Centering* shown in image (b), connections C_1 and C_2 become relaxed, while C_3 remains unrelaxed, C_4 remains relaxed, and C_5 becomes unrelaxed. Thus, the new number of unrelaxed connections is $\zeta_P = 2$, such that the relaxation delta is $\Delta\zeta_P = -1$. Hence, the action performed here represents a relaxing action.

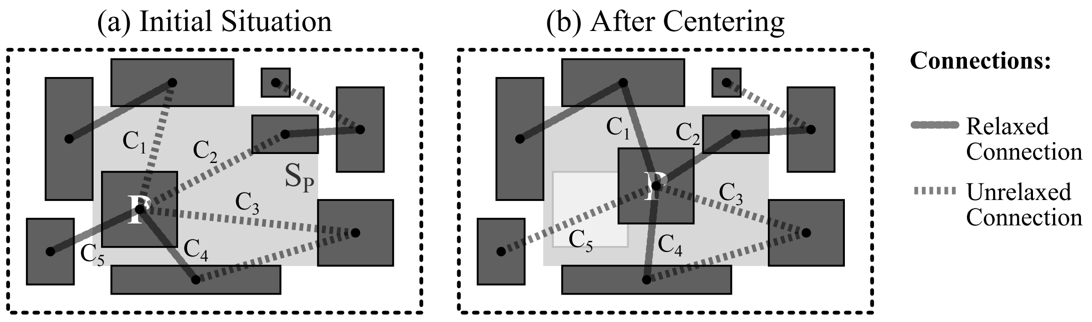


Figure 7.45: A relaxing action decreases a participant's number of unrelaxed connections.

With these conceptions in the sphere of relaxation, the comparison condition 7.73 is now enhanced, distinguishing two cases. If two actions are *not* equally relaxing, they are simply compared by their relaxation deltas. Otherwise, as will be detailed farther below, the prospective interference *and* the prospective turmoil are taken into account to rate an action. This is done by adding up the *relative change* of interference and turmoil, written as $\delta\Upsilon_P$ and $\delta\Theta_P$ respectively. Thus, if two actions T_a and T_b are *not* equally relaxing, they are compared via condition

$$\Delta\zeta_P(T_a) < \Delta\zeta_P(T_b) \tag{7.77}$$

while otherwise –i.e., if T_a and T_b *are* equally relaxing– the following condition

$$\delta\Upsilon_P(T_a) + \delta\Theta_P(T_a) < \delta\Upsilon_P(T_b) + \delta\Theta_P(T_b) \tag{7.78}$$

is applied. Therein, the relative change of interference $\delta\Upsilon_P(T)$ achieved via action T is defined as

$$\delta\Upsilon_P(T) = \frac{\Upsilon_P^{\text{new}}(T) - \Upsilon_P}{\Upsilon_P^{\text{ref}}} \tag{7.79}$$

where Υ_P again is P's present interference while $\Upsilon_P^{\text{new}}(T)$ denotes the prospective interference that P achieves with T. The reference value Υ_P^{ref} is given as

$$\Upsilon_P^{\text{ref}} = \begin{cases} \Upsilon_P & \Leftrightarrow \quad \Upsilon_P > 0 \\ [\square P]^2 & \Leftrightarrow \quad \text{otherwise.} \end{cases} \tag{7.80}$$

Here, the value $[\text{⬚}P]^2$ is used as a worst-case baseline for interference, reflecting the situation where P is completely overlapped by another participant. This *trouble* is approximated by putting $[\text{⬚}P]$ (the area of P) to the square, such that it represents the intensity of the trouble on the one hand, and the trouble's tenacity on the other hand (also see equation 7.21 in Section 7.3.1.1). Since it is always the case that $[\text{⬚}P] > 0$, the calculation of the reference value Υ_P^{ref} does never risk to run into a *division by zero* problem. Furthermore, $[\text{⬚}P] > 0$ (or rather $[\text{⬚}P]^2 > 0$) means that the relative change of interference $\delta\Upsilon_P(T)$ is beneficial if and only if

$$\delta\Upsilon_P(T) < 0. \tag{7.81}$$

Considering distances between the participants, $\delta\Theta_P(T)$ defines the relative change of turmoil for a participant P (equivalent to equation 7.79 above) as follows:

$$\delta\Theta_P(T) = \frac{\Theta_P^{\text{new}}(T) - \Theta_P}{\Theta_P^{\text{ref}}} \tag{7.82}$$

with Θ_P and $\Theta_P^{\text{new}}(T)$ denoting P's present and prospective turmoil. With $\hat{\Theta}_P$ being the greatest turmoil that participant P has encountered so far, the reference value in equation 7.82 is defined as

$$\Theta_P^{\text{ref}} = \begin{cases} \Theta_P & \Leftrightarrow \quad \Theta_P > 0 \\ \hat{\Theta}_P & \Leftrightarrow \quad \text{otherwise.} \end{cases} \tag{7.83}$$

Having equation 7.83 cover the circumstance that the present turmoil could be zero is analogous to the case distinction made in equation 7.80. However, during the module interaction the chance of running into a constellation where the turmoil is indeed $\Theta_P = 0$, can be assumed to be minuscule. As discussed in Section 7.3.1.2, a participant's turmoil is calculated as the sum of tensions in its connections. For a connection C with its length l_C, its emphasis e_C, its strength s_C, and its relaxation threshold ϱ_C, it is always true that $l_C \geq 0$, $e_C > 0$, $s_C > 0$ and $\varrho_C > 0$. From equation 7.48, this means that the tension θ in a connection can never be negative (as confirmed by Figure 7.20), such that also the turmoil of a participant is always $\Theta_P \geq 0$. For a participant P with connections $\mathcal{C}_P$, the turmoil Θ_P can only become zero if the tension in –and thus the length of– every connection in $\mathcal{C}_P$ is zero:

$$\Theta_P = 0 \iff \forall C \in \mathcal{C}_P, l_C = 0. \tag{7.84}$$

This incident can be neglected since it is extremely unlikely to occur. But in the beginning of a SWARM run, one would definitely encounter that $\Theta_P = 0$ (which inevitably means that $\hat{\Theta}_P = 0$ as well because $\hat{\Theta}_P = \Theta_P$ in that moment) if all participants were centered at the very same location. So, in order to avoid a *division by zero* problem here, the initial constellation of the participants $\mathcal{P}$ must be chosen such that in formal terms (using $C_{P,P'}$ to denote that C connects participant P with participant P') the following statement is true:

$$\forall P \in \mathcal{P},\ \exists C_{P,P'} \in \mathcal{C}_P : L_P \neq L_{P'} \tag{7.85}$$

which means that for every participant P, at least one participant P' among its connected participants must be stationed at a location $L_{P'}$ that is different from the location L_P of P. This is easily achieved by choosing an initial constellation without any interference (i.e., $\forall P \in \mathcal{P}, \Upsilon_P = 0$). Since it can be expected that $\Theta_P^{\text{ref}} > 0$, the relative change of turmoil $\delta\Theta_P(T)$ is beneficial if and only if

$$\delta\Theta_P(T) < 0. \tag{7.86}$$

Hence, condition 7.81 and condition 7.86 justify why the sum of $\delta\Upsilon_P(T)$ and $\delta\Theta_P(T)$ is used for the action comparison in condition 7.78, which achieves that the action with the best compromise between prospective interference and prospective turmoil will finally be picked. Again, that action will only be executed if it is beneficial, which implies that

$$\delta\Upsilon_P(T) + \delta\Theta_P(T) < 0. \tag{7.87}$$

This condition indicates, that an acion is considered to be beneficial even if it concedes an increase in either (a) interference or (b) turmoil, as long as that increase is more than counterbalanced by a decrease in (a) turmoil or (b) interference, respectively.

(3) Actions Involving Multiple Participants

As to the *without loss of generality* remark at the beginning of this Section 7.3.4.1, the entire comparison metric described above can also be applied to actions that involve transformations of more than one participant. This is basically done by accumulating the respective quantities (i.e., the interference Υ, the turmoil Θ, and the relaxation delta $\Delta\zeta$).

To generalize the notation used so far, let $\Upsilon^{\text{new}}_{\mathcal{P}}(\mathcal{T})$ with $\mathcal{P} = \{P, P', P'', \dots\}$ denote the prospective *collective* interference of an action $\mathcal{T}$ (i.e., a set of transformations $\{T, T', T'', \dots\}$) that is initiated by the leading participant P, but involves further participants (i.e., P', P'', etc.). Then, the comparison condition 7.73 can also be generalized such that it reads

$$\Upsilon^{\text{new}}_{\mathcal{P}}(\mathcal{T}_a) < \Upsilon^{\text{new}}_{\mathcal{P}}(\mathcal{T}_b). \tag{7.88}$$

In the same manner, but now to compare the *collective* relaxation delta, condition 7.77 becomes

$$\Delta\zeta_{\mathcal{P}}(\mathcal{T}_a) < \Delta\zeta_{\mathcal{P}}(\mathcal{T}_b) \tag{7.89}$$

while condition 7.78 –in order to compare the *collective* relative change of Υ and Θ– turns into

$$\delta\Upsilon_{\mathcal{P}}(\mathcal{T}_a) + \delta\Theta_{\mathcal{P}}(\mathcal{T}_a) < \delta\Upsilon_{\mathcal{P}}(\mathcal{T}_b) + \delta\Theta_{\mathcal{P}}(\mathcal{T}_b). \tag{7.90}$$

One might fall prey to the misbelief that the collective interference $\Upsilon^{\text{new}}_{\mathcal{P}}(\mathcal{T})$ is calculated by adding up *all* prospective interferences of the individual participants that perform a transformation due to the action. This *can* be correct in some cases such as for example in Figure 7.46 (a): regarding the potential *Pairing* of P with P', the *trouble* of P and the trouble of P' are disjunct and must both be added together. However, the situation is different if two (or more) of the involved participants interfere with each other. That is because in such a case, the mutual trouble would be wrongly included twice, as exemplified in Figure 7.46 (b), where P explores a *Pairing* with P' which involves an action correction to make sure that P' is *safe*: P overlaps P' and P' overlaps P, so they share the same trouble, which should therefore only be counted once.

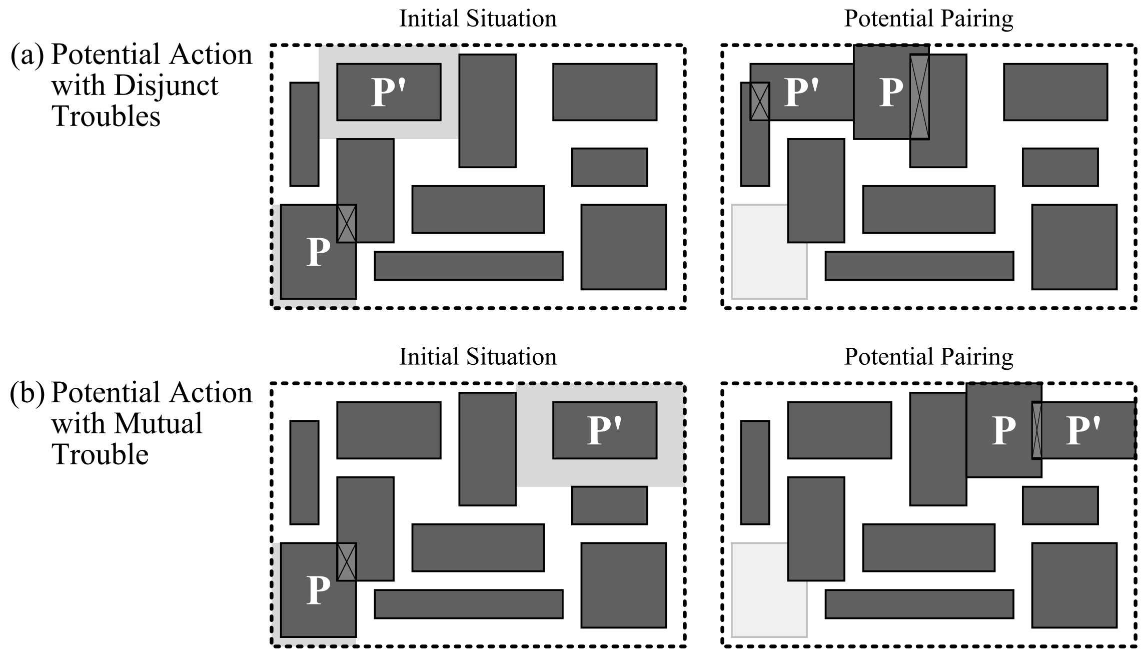

Initial Situation
Potential Pairing
(a) Potential Action with Disjunct Troubles
P'
P
P'
P
Initial Situation
Potential Pairing
(b) Potential Action with Mutual Trouble
P'
P
P
P'

Figure 7.46: In contrast to disjunct troubles (a), mutual trouble (b) must be counted only once.

Considering an action that involves two participants (i.e., the leading participant P and another participant P'), for example a *Pairing* (as in Figure 7.46) or a *Swapping*, the prospective collective interference is correctly calculated via

$$\Upsilon^{\text{new}}_{\{P,P'\}}(\mathcal{T}) = \sum_{P^* \in \mathcal{P}^* = P} \tau^{\text{new}}_{P,P^*}(\mathcal{T}) + \sum_{P^* \in \mathcal{P}^* \setminus \{P',P\}} \tau^{\text{new}}_{P',P^*}(\mathcal{T}) \tag{7.91}$$

where $\mathcal{P}^*$ represents the set of all participants and the notation $\tau^{\text{new}}_{P,P^*}(\mathcal{T})$ is used to denote the prospective trouble between two participants P and P^*. With this formula, potential trouble between the involved participants P and P' is included in the first addend but excluded from the second addend, and is thus only counted once, as desired. In this spirit, the formula can be extended to target the calculation of $\Upsilon^{\text{new}}_{\mathcal{P}}(\mathcal{T})$ in general, thus also addressing actions with $|\mathcal{T}| > 2$ (where more than two participants are involved) such that the prospective collective interference is correctly compared in condition 7.88.

Just as well, one must pay heed to this subtlety in the calculation of the present collective interference $\Upsilon_{\mathcal{P}}$, and –equivalently– in the calculation of the prospective collective turmoil $\Theta^{\text{new}}_{\mathcal{P}}(\mathcal{T})$ and the present collective turmoil $\Theta_{\mathcal{P}}$, such that a connection's tension is not counted more than once. This guarantees, that the collective relative change of Υ and Θ in condition 7.90 is correct. Finally, also the calculation of the collective relaxation delta $\Delta\zeta_{\mathcal{P}}(\mathcal{T})$ –used for the comparison in condition 7.89– must make sure not to doubly count a connection that becomes relaxed due to action $\mathcal{T}$.

7.3.4.2 Action Execution

The two tasks of exploring potential actions and choosing the most preferred action cannot be clearly separated from each other in terms of consecution. Instead, they proceed in a rather intermingled way before the chosen action is finally executed. This will now be explained by the subsequent description, which elucidates the liaison of the two tasks within the entirety of SWARM's action scheme. The prosaic description is literally "in line" with the more algorithmic representation provided by Algorithm 1.

Let there be a participant P, whose turn it is to take an action. Following the action scheme, P's **first measure** (line 1ff.) –as covered in Section 7.3.1– is to assess its condition in order to determine if

- P is clear: $\Upsilon_P = 0$ (see Section 7.3.1.1 on interference),
- P is relaxed: $\zeta_P = 0$ (see Section 7.3.1.2 on turmoil),
- P is safe: $\Psi_P = \varnothing$ (see Section 7.3.1.3 on protrusion),
- P is healthy: $\mathcal{W}^{\ddagger}_{P_{\bowtie}} = \emptyset$ (see Section 7.3.1.4 on wounds),
- P is compliant: $\mathcal{H}^{\ddagger}_{P} = \emptyset$ (see Section 7.3.1.5 on noncompliance).

Next, P's **second measure** (line 6) is to perceive its free peripheral space S_P according to the geometrical recipe given in Section 7.3.2.

Then, P's **third measure** is to explore and evaluate potential actions, whereby an *adjuvant* action can already be chosen for execution during the action exploration. Choosing an action is done by storing the action in a variable here denoted as $\mathcal{T}_{\text{final}}$, which is initialized with a null value in the beginning of the action exploration (line 7). As long as that variable is null, the action exploration continues, otherwise the participant proceeds with the execution of that action. Looking forward to the case that no action is chosen during the action exploration, every *valid* action is memorized in a set $\mathbb{T}_{\text{valid}}$ (initialized in line 8), from which the best action can then be chosen afterwards.

Custom actions: If custom actions such as the *Imitation* action from Section 7.3.3.2 are defined on P, these are explored first of all (see the for-loop in line 9ff.). When a custom action $\mathcal{T}_{\text{custom}}$ is deemed *acceptable*, the action is stored in $\mathcal{T}_{\text{final}}$ and the participant can immediately exit the for-loop.

Re-entering: Given that no custom action has been chosen, SWARM's native actions from Section 7.3.3.1 are explored. Starting off with the case that P finds itself being lost, then a *Re-entering* action $\mathcal{T}_{\text{Re-entering}}$ is determined and stored in $\mathcal{T}_{\text{final}}$ (line 17ff.). By definition, the action is *tolerable* because it eliminates P's protrusion.

Centering: Next, the participant –if it is not lost– explores potential *Centering* actions (line 21ff.). For all orientations $O \in \mathbb{O}$ and for all layout variants $V_P \in \widetilde{\mathcal{V}}_P$ that P can assume, an action $\mathcal{T}_{\text{Centering}}$ is determined and evaluated. If the prospective location is safe, healthy and compliant, then the action is valid and therefore memorized in $\mathbb{T}_{\text{valid}}$. If P would even become clear and relaxed, then the action is adjuvant and is thus stored in $\mathcal{T}_{\text{final}}$, letting the participant exit the for-loop.

Lingering: In case no adjuvant *Centering* action could be found, the participant checks if it is currently clear, relaxed, safe and compliant (line 32). If that is so, P is *contented* (this can be said without checking for wounds, since a participant is definitely healthy if it is clear). Then, an "empty"

Algorithm 1 SWARM's action scheme as acted upon by a participant P

```
Measure 1: Assessment of the Participant's Condition
 1: $\Upsilon_P \leftarrow$ interference
 2: $\zeta_P \leftarrow$ unrelaxed connections (turmoil)
 3: $\Psi_P \leftarrow$ protrusion
 4: $\mathcal{W}^{\ddagger}_{P_{\bowtie}} \leftarrow$ unhealthy wounds
 5: $\mathcal{H}^{\ddagger}_{P} \leftarrow$ dissatisfied constraints (noncompliance)
Measure 2: Perception of the Free Peripheral Space
 6: $S_P \leftarrow$ free peripheral space
Measure 3: Exploration and Evaluation of Possible Actions
 7: $\mathcal{T}_{\text{final}} \leftarrow$ null
 8: $\mathbb{T}_{\text{valid}} \leftarrow \{\}$
 9: for all kinds of custom actions defined on $P$ do
10:     if $\mathcal{T}_{\text{final}} =$ null then
11:         determine custom action $\mathcal{T}_{\text{custom}}$
12:         if $\mathcal{T}_{\text{custom}}$ is acceptable then        ▷ this means the action is valid, tolerable, or mandatory
13:             $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{custom}}$ ; exit for-loop
14:         end if
15:     end if
16: end for
17: if $\mathcal{T}_{\text{final}} =$ null AND $\Psi_P = \square P$ then
18:     determine Re-entering action $\mathcal{T}_{\text{Re-entering}}$        ▷ the action is tolerable since $\Psi^{\text{new}}_P = \varnothing$
19:     $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Re-entering}}$
20: end if
21: if $\mathcal{T}_{\text{final}} =$ null AND $\Psi_P \neq \square P$ then
22:     for all $(O, V_P) \in (\mathbb{O} \times \widetilde{\mathcal{V}}_P)$ do
23:         determine Centering action $\mathcal{T}_{\text{Centering}}$ with respect to $S_P$
24:         if $\Psi^{\text{new}}_P = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P_{\bowtie}} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P} = \emptyset$ then        ▷ this means the action is valid
25:             $\mathbb{T}_{\text{valid}} \leftarrow \mathbb{T}_{\text{valid}} + \mathcal{T}_{\text{Centering}}$
26:             if $\Upsilon^{\text{new}}_P = 0$ AND $\zeta^{\text{new}}_P = 0$ then        ▷ this means the action is adjuvant
27:                 $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Centering}}$ ; exit for-loop
28:             end if
29:         end if
30:     end for
31: end if
32: if $\mathcal{T}_{\text{final}} =$ null AND $\Upsilon_P = 0$ AND $\zeta_P = 0$ AND $\Psi_P = \varnothing$ AND $\mathcal{H}^{\ddagger}_{P} = \emptyset$ then        ▷ $P$ is contented
33:     $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Lingering}}$
34: end if
35: if $\mathcal{T}_{\text{final}} =$ null AND $\Psi_P = \varnothing$ then
36:     for all $N$ of $\square P$ do
37:         perceive secondary free peripheral space $S^N_P$
38:         for all $(O, V_P, X) \in (\mathbb{O} \times \widetilde{\mathcal{V}}_P \times \mathcal{X})$ do
39:             determine Budging action $\mathcal{T}_{\text{Budging}}$ with respect to $S^N_P$
40:             if $\Psi^{\text{new}}_P = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P_{\bowtie}} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P} = \emptyset$ then        ▷ this means the action is valid
41:                 $\mathbb{T}_{\text{valid}} \leftarrow \mathbb{T}_{\text{valid}} + \mathcal{T}_{\text{Budging}}$
42:                 if $\Upsilon^{\text{new}}_P = 0$ AND $\zeta^{\text{new}}_P = 0$ then        ▷ this means the action is adjuvant
43:                     $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Budging}}$ ; exit for-loops
44:                 end if
45:             end if
46:         end for
47:     end for
48: end if
```

Algorithm 2 SWARM's action scheme as acted upon by a participant P (continued)

```
49: if $\mathcal{T}_{\text{final}} = \texttt{null}$ AND $\Upsilon_P > 0$ AND $\Psi_P = \varnothing$ then
50:     determine *Hustling* action $\mathcal{T}_{\text{Hustling}}$ for participants $\mathcal{P}_\iota$
51:     if $\forall P_\iota \in \mathcal{P}_\iota, \Psi^{\text{new}}_{P_\iota} = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P_\iota \bowtie} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P_\iota} = \emptyset$ then    ▷ means the action is valid
52:         $\mathbb{T}_{\text{valid}} \leftarrow \mathbb{T}_{\text{valid}} + \mathcal{T}_{\text{Hustling}}$
53:         if $\forall P_\iota \in \mathcal{P}_\iota, \Upsilon^{\text{new}}_{P_\iota} = 0$ AND $\zeta^{\text{new}}_{P_\iota} = 0$ then    ▷ this means the action is adjuvant
54:             $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Hustling}}$ ; exit for-loop
55:         end if
56:     end if
57: end if
58: if $\mathcal{T}_{\text{final}} = \texttt{null}$ AND $\Psi_P = \varnothing$ then
59:     for all $P' \in \mathcal{P} - P$ do
60:         perceive free peripheral space $S_{P'}$ of $P'$
61:         if promising then
62:             for all $X \in \mathcal{X}$ do
63:                 determine *Swapping* action $\mathcal{T}_{\text{Swapping}}$ for $P$ and $P'$ with respect to $S_P$ and $S_{P'}$
64:                 if $\Psi^{\text{new}}_{P} = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P \bowtie} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P} = \emptyset$ then
65:                     if $\Psi^{\text{new}}_{P'} = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P' \bowtie} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P'} = \emptyset$ then
                                                                        ▷ this means the action is valid
66:                         $\mathbb{T}_{\text{valid}} \leftarrow \mathbb{T}_{\text{valid}} + \mathcal{T}_{\text{Swapping}}$
67:                         if $\Upsilon^{\text{new}}_{P} = 0$ AND $\zeta^{\text{new}}_{P} = 0$ then
68:                             if $\Upsilon^{\text{new}}_{P'} = 0$ AND $\zeta^{\text{new}}_{P'} = 0$ then    ▷ this means the action is adjuvant
69:                                 $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Swapping}}$ ; exit for-loops
70:                             end if
71:                         end if
72:                     end if
73:                 end if
74:             end for
75:         end if
76:     end for
77: end if
78: if $\mathcal{T}_{\text{final}} = \texttt{null}$ AND $\Psi_P = \varnothing$ then
79:     for all $P' \in \mathcal{P} - P$ do
80:         perceive free peripheral space $S_{P'}$ of $P'$    ▷ can be re-used from the *Swapping* exploration
81:         if promising then
82:             for all $X \in \mathcal{X}$ do
83:                 determine *Pairing* action $\mathcal{T}_{\text{Pairing}}$ for $P$ and $P'$ with respect to $S_{P'}$
84:                 if $\Psi^{\text{new}}_{P} = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P \bowtie} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P} = \emptyset$ then
85:                     if $\Psi^{\text{new}}_{P'} = \varnothing$ AND $\mathcal{W}^{\ddagger\ \text{new}}_{P' \bowtie} = \emptyset$ AND $\mathcal{H}^{\ddagger\ \text{new}}_{P'} = \emptyset$ then
                                                                        ▷ this means the action is valid
86:                         $\mathbb{T}_{\text{valid}} \leftarrow \mathbb{T}_{\text{valid}} + \mathcal{T}_{\text{Pairing}}$
87:                         if $\Upsilon^{\text{new}}_{P} = 0$ AND $\zeta^{\text{new}}_{P} = 0$ then
88:                             if $\Upsilon^{\text{new}}_{P'} = 0$ AND $\zeta^{\text{new}}_{P'} = 0$ then    ▷ this means the action is adjuvant
89:                                 $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Pairing}}$ ; exit for-loops
90:                             end if
91:                         end if
92:                     end if
93:                 end if
94:             end for
95:         end if
96:     end for
97: end if
```

Algorithm 3 SWARM's action scheme as acted upon by a participant P (continued)

```
98:  if $\mathcal{T}_{\text{final}} = \texttt{null}$ AND $\Psi_P \sqsubset \square P$ then
99:      for all $(O, V_P, X) \in (\mathbb{O} \times \widetilde{\mathcal{V}}_P \times \mathcal{X})$ do
100:         determine Evasion action $\mathcal{T}_{\text{Evasion}}$ with respect to $S_P$
101:         if $\Psi_P^{\text{new}} = \varnothing$ AND $\mathcal{W}_{P_{\bowtie}}^{\ddagger\ \text{new}} = \emptyset$ AND $\mathcal{H}_P^{\ddagger\ \text{new}} = \emptyset$ then    ▷ this means the action is valid
102:             $\mathbb{T}_{\text{valid}} \leftarrow \mathbb{T}_{\text{valid}} \uplus \mathcal{T}_{\text{Evasion}}$
103:             if $\Upsilon_P^{\text{new}} = 0$ AND $\zeta_P^{\text{new}} = 0$ then    ▷ this means the action is adjuvant
104:                 $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Evasion}}$ ; exit for-loop
105:             end if
106:         end if
107:     end for
108: end if
109: if $\mathcal{T}_{\text{final}} = \texttt{null}$ then
110:     if $\mathbb{T}_{\text{valid}} \neq \{\,\}$ then    ▷ this means that at least one valid action has been found
111:         sort $\mathbb{T}_{\text{valid}}$ by benefit    ▷ sorting is done according to the comparison metric
112:         store the best action in $\mathcal{T}_{\text{best}}$
113:         if $\mathcal{T}_{\text{best}}$ is beneficial OR $\Psi_P \neq \varnothing$ OR $\mathcal{W}_{P_{\bowtie}}^{\ddagger} \neq \emptyset$ OR $\mathcal{H}_P^{\ddagger} \neq \emptyset$ then
114:             $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{best}}$
115:         end if
116:     end if
117: end if
118: if $\mathcal{T}_{\text{final}} = \texttt{null}$ AND $\Upsilon_P > 0$ AND $\Psi_P \neq \square P$ then
119:     identify the polygonal yielding region $Y_P$
120:     determine Yielding action $\mathcal{T}_{\text{Yielding}}$ with respect to $Y_P$
121:     if $\Psi_P^{\text{new}} = \varnothing$ AND $\mathcal{W}_{P_{\bowtie}}^{\ddagger\ \text{new}} = \emptyset$ AND $\mathcal{H}_P^{\ddagger\ \text{new}} = \emptyset$ then    ▷ this means the action is valid
122:         $\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{Yielding}}$
123:     end if
124: end if
Measure 4: Execution of the Preferred Action
125: if $\mathcal{T}_{\text{final}} \neq \texttt{null}$ then
126:     if $\exists T \in \mathcal{T}_{\text{final}} : \sqrt{(\Delta x_T)^2 + (\Delta y_T)^2} \geq m \vee O_{P_T}^{\text{new}} \neq O_{P_T} \vee V_{P_T}^{\text{new}} \neq V_{P_T}$
           OR $\Psi_P \neq \varnothing$ OR $\mathcal{W}_{P_{\bowtie}}^{\ddagger} \neq \emptyset$ OR $\mathcal{H}_P^{\ddagger} \neq \emptyset$ then    ▷ $m$ is the minimal movement distance
127:         for all $T \in \mathcal{T}_{\text{final}}$ do
128:             apply transformation $T$ to the respective participant $P_T$
129:         end for
130:     end if
131: end if
```

Lingering action with $\Delta x = 0$ and $\Delta y = 0$ is stored in $\mathcal{T}_{\text{final}}$. Mathematically speaking, the *Lingering* action represents the *identity element* among SWARM's actions (i.e., a *neutral element* such as the ones used in algebraic structures).

Budging: Unless P lingers where it is, potential *Budging* actions are explored if the participant is safe (line 35ff.). For each vertex of P's bounding box $\square P$, represented as a *node* N, the secondary free peripheral space S_P^N is perceived for action exploration. How exhaustive that exploration is done, depends on the *exploration plan* which can be abstractly thought of as a set $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$. Therein, each X represents one possible *exploration scenario* with respect to the kind of action being explored. For a *Budging*, every exploration scenario specifies one possible edge or vertex of S_P^N that P can align itself with. So, for every possible orientation O, for any layout variant V_P of P, and for each exploration scenario X, an action $\mathcal{T}_{\text{Budging}}$ is determined and evaluated. If the action is valid, it is memorized in $\mathbb{T}_{\text{valid}}$; if the action is even adjuvant, it is stored in $\mathcal{T}_{\text{final}}$ (as also done in the exploration of *Centering* actions).

Hustling: Provided that no adjuvant action has yet been found, a *Hustling* action $\mathcal{T}_{\text{Hustling}}$ is explored (line 49ff. in Algorithm 2), subject to the condition that the participant is safe but not clear (otherwise the exploration of a *Hustling* action would be futile). As above, the explored action is memorized in $\mathbb{T}_{\text{valid}}$ in case it is valid; and is further stored in $\mathcal{T}_{\text{final}}$ if it is adjuvant – however, this has to take not only P into account, but all participants $\mathcal{P}_\iota$ that are involved in the action.[11] Computationally, determining a *Hustling* action is cheap (even if a more exhaustive exploration plan is pursued), therefore it is opportunely done *prior* to the subsequent exploration of the other kinds of actions, which raise higher combinatorial expenses (especially *Swapping* and *Pairing*).

Swapping: If P is safe, the action exploration continues with *Swapping* actions (line 58ff.). For every participant P' among the set $\mathcal{P} - P$ (the set of all participants except P), the free peripheral space $S_{P'}$ is perceived. Then, as discussed under *Swapping* in Section 7.3.3.1, the participant might try to predict whether a *Swapping* of P and P' is promising or not. If yes, then an action $\mathcal{T}_{\text{Swapping}}$ is explored for every exploration scenario X of the exploration plan $\mathcal{X}$. If the prospective locations are safe, healthy and compliant for *both* participants P and P', then the action is valid and hence gets memorized in $\mathbb{T}_{\text{valid}}$. If both participants would even become clear and relaxed, then the action is adjuvant and is thus used as the final action $\mathcal{T}_{\text{final}}$.

Pairing: In the same manner, potential *Pairing* actions are explored (line 78ff.). For every other participant P', the free peripheral space $S_{P'}$ is to be determined. In view of this, it is feasible to save the free peripheral spaces during the *Swapping* exploration such that they can now be retrieved without the need to perceive them anew. If a *Pairing* of P with P' looks promising (depending on $S_{P'}$), then for every exploration scenario X in the exploration plan $\mathcal{X}$ an action $\mathcal{T}_{\text{Pairing}}$ is explored. If the action is valid (again regarding *both* participants), it is memorized in $\mathbb{T}_{\text{valid}}$. If the action is adjuvant (regarding both participants), then it is used as the final action $\mathcal{T}_{\text{final}}$.

Evasion: Be it that still no adjuvant action is available and that the participant is prone, potential *Evasion* actions are finally explored by the participant (line 98ff. in Algorithm 3). For each orientation O, for any layout variant V_P of P and for every exploration scenario X in the exploration plan $\mathcal{X}$, an action $\mathcal{T}_{\text{Evasion}}$ is determined and evaluated. As with the other kinds of actions, the explored action is memorized in $\mathbb{T}_{\text{valid}}$ if it is valid; in case the action is even adjuvant, it is stored in $\mathcal{T}_{\text{final}}$.

Best action: In the event that the participant has not been able to find an adjuvant action, the best non-adjuvant valid action is chosen (line 109ff.). This presupposes that at least one valid action has been found (i.e., $\mathbb{T}_{\text{valid}} \neq \{\,\}$). The memorized actions of $\mathbb{T}_{\text{valid}}$ are sorted according to the comparison metric given in Section 7.3.4.1, such that the best action (the action with the greatest benefit) can be chosen and stored in $\mathcal{T}_{\text{best}}$. However, that action is only used as the final action $\mathcal{T}_{\text{final}}$ if (a) it is *beneficial* –see condition 7.75 and condition 7.87– or (b) if the participant's current location is not safe ($\Psi_P \neq \varnothing$), not healthy ($\mathcal{W}^{\ddagger}_{P_{\bowtie}} \neq \emptyset$), or not compliant ($\mathcal{H}^{\ddagger}_{P} \neq \emptyset$). Thus in case (b), the best action will be performed even if it is not beneficial, otherwise such a best action is rejected.

Yielding: Given that by now the action exploration has not put forth a valid action $\mathcal{T}_{\text{final}}$ to perform, the participant checks if it suffers from interference ($\Upsilon_P > 0$), because then a *Yielding* action can be explored. This is done if the participant is not lost (line 118ff.). For that purpose, the polygon Y_P (introduced as the *yielding region* in Section 7.3.3.1) is identified, which in turn allows P to determine the *Yielding* action $\mathcal{T}_{\text{Yielding}}$. If the action is valid, then it is used as the final action $\mathcal{T}_{\text{final}}$, regardless of whether this represents a beneficial action or not.

The action exploration is followed by the action execution, which marks P's **fourth measure** (line 125ff.). Executing an action first of all implies that a valid action has been determined (i.e., $\mathcal{T}_{\text{final}} \neq$ `null`). In that case, the participant then affirms that the action is not *trivial*.

Definition 7.6. *An action $\mathcal{T}$ is not trivial if at least one of its transformations $T \in \mathcal{T}$ is not trivial. A transformation T is not trivial (a) if the length of its movement vector M_T is not smaller than the minimal movement distance m, or (b) if T would turn the respective participant P_T into a different orientation*

[11]In the implementation, checking for $\Psi_P^{\text{new}} = \varnothing$ (prospective protrusion of the leading participant P) is in fact superfluous because P itself is already safe and does not move. However, checking for $\Upsilon_P^{\text{new}} = 0$ (prospective interference) *is* necessary since the *Hustling* action may be unsuccessful, either because P is *properly enclosed* by another participant (as explained in Section 7.3.3.1 under *Hustling*) or because an *action correction* is involved (see Section 7.3.1.3).

$O_{P_T}^{new}$ due to a rotation R_T, or (c) if T would change that participant into another layout variant $V_{P_T}^{new}$ due to a deformation D_T, or (d) if the leading participant P currently is in an invalid location (not safe, not healthy, or not compliant).

Trivial actions are discarded because their potential benefit is insignificant, which tends to retard the overall self-organization flow rather than to enliven it. If the action $\mathcal{T}_{\text{final}}$ is not trivial, then it is executed, i.e., every transformation T of the action is applied to the respective participant P_T. This concludes the way in which the leading participant P acts upon SWARM's action scheme. Directing the overall *inter*-action towards the desired layout outcome is up to the interaction control organ – SWARM's third core concept, that will now be addressed in Section 7.4.

7.4 Interaction Control

As exposed in Chapter 6, immense power can be ascribed to self-organizing, decentralized structures. But to tap the full potential of such structures, regarding the design of bottom-up systems for practical utilization, [333] points out: "the bottom is not enough. You need a bit of top-down as well." This is why SWARM includes an *interaction control* organ which supervises the modules' actions of Section 7.3 from a top-down perspective and carefully steers the interaction in order to enforce the emergence of a constraint-compliant, compact constellation that fits within the user-defined layout zone Z.

As illustrated in the control-flow depiction of Figure 7.3, the steering of the module interaction proceeds in an indirect fashion which is achieved by repeatedly changing the size of the user-defined layout zone. This represents the main task of the interaction control organ and will be explicated in Section 7.4.1. Extending these considerations, the notion of *transient tightening policies* is then introduced in Section 7.4.2. A further idea within that orbit is to envelop each participating module in a personal *comfort padding* correlated with the tightening of the layout zone. As will be discussed in Section 7.4.3, this idea is meant to smooth the self-organization flow and to equilibrate the evocation of emergent behavior throughout a SWARM run.

7.4.1 Scaling the Layout Zone

In line with the depiction of Figure 7.3, the interaction control organ's task of scaling the user-defined layout zone incorporates two discernible jobs: (1) setting and enlarging the zone in the beginning of the self-organization phase, and then (2) tightening the layout zone after each settlement. The former job will be addressed in Section 7.4.1.1, while Section 7.4.1.2 covers the details of the latter job. For arbitrarily rectilinear[12] layout zones, some particular considerations must be made, as Section 7.4.1.3 will discuss.

7.4.1.1 Setting and Enlarging the Layout Zone

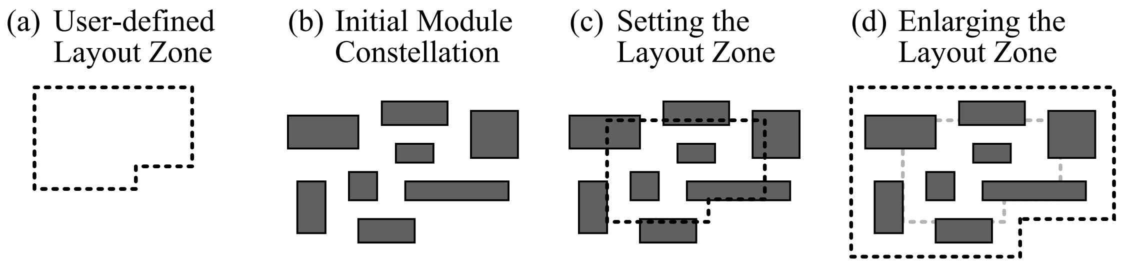

(a) User-defined Layout Zone
(b) Initial Module Constellation
(c) Setting the Layout Zone
(d) Enlarging the Layout Zone

Figure 7.47: Setting and enlarging the user-defined layout zone in the beginning of the self-organization.

Demarcating the layout territory that is available for the design problem at hand, the layout zone Z is expected to be given by the user as a rectangular or rectilinear polygon such as in Figure 7.47 (a). Then, for an initial module constellation –i.e., a set of n participants $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$– as in

[12]In the remainder of this thesis, the term *rectilinear* especially points to the case where the layout zone is *not rectangular*.

Figure 7.47 (b), the interaction control organ determines the rectangular bounding box $\boxdot\mathcal{P}$ around all participants, denoted as the *constellation frame* F and calculated via

$$F = \boxdot\mathcal{P} = \left(\left(\min_{\forall P \in \mathcal{P}}\left(\vdash(\boxdot P)\right), \min_{\forall P \in \mathcal{P}}\left(\bot(\boxdot P)\right)\right), \left(\max_{\forall P \in \mathcal{P}}\left(\dashv(\boxdot P)\right), \max_{\forall P \in \mathcal{P}}\left(\top(\boxdot P)\right)\right)\right) \tag{7.92}$$

in rectangle notation. To put Z into its initial location as in the example of Figure 7.47 (c), the interaction control organ moves it by

$$\left(\frac{1}{2}\left(\vdash F - \vdash Z + \dashv F - \dashv Z\right), \frac{1}{2}\left(\bot F - \bot Z + \top F - \top Z\right)\right) \tag{7.93}$$

which is the vectorial difference of the constellation frame's center point and the center point of the layout zone's bounding box. Throughout the remaining SWARM run, the layout zone is never again moved or transformed in another way apart from being resized. Foremost, prior to the first round of interaction, Z is enlarged such that the area of the layout zone becomes considerably larger than the totaled areas of the individual participants, as can be seen in Figure 7.47 (d).

It is important to note, that the aspect ratio of Z is not modified thereby. This implies that the enlargement of Z is not achieved through an equal, absolute dilation of its edges (as done by the *grow operator* $\circledcirc_\varepsilon$ on page 243), but via a scaling operation that works in relation to the current size of Z. Considering a geometrical shape G in general, this scaling operation takes a relative, percental *contraction amount* $\xi \in \mathbb{R}$ in the right-open interval $(-\infty, 0.5)$ and applies it to the shape quadrilaterally, i.e., in all four directions of the coordinate system (in the two-dimensional layout plane).

Figure 7.48 exemplifies this contraction operation with a given rectangle (a) of width w and height h: a contraction amount of $\xi = -0.5$ stretches the rectangle by an absolute *edge displacement* $\epsilon = 0.5 \cdot w$ to the left and to the right, whereas the vertical edge displacement is $0.5 \cdot h$ both upwards and downwards, such that the rectangle becomes four times as large in this particular case (b). The increase in size with respect to ξ can be universally given as the quotient of the shape's new area $\llbracket G_{\text{new}} \rrbracket$ and its old area $\llbracket G_{\text{old}} \rrbracket$ via the following formula (which holds true for $\xi \leq 0.5$):

$$\frac{\llbracket G_{\text{new}} \rrbracket}{\llbracket G_{\text{old}} \rrbracket} = \frac{(w - 2\xi w) \cdot (h - 2\xi h)}{w \cdot h} = \frac{w(1-2\xi) \cdot h(1-2\xi)}{w \cdot h} = (1-2\xi)^2. \tag{7.94}$$

Thus, as the term contraction suggests, the shape is enlarged by a negative contraction amount ($\xi < 0$) and made smaller by a positive contraction amount ($\xi > 0$). This is deliberate, owing to the fact that the layout zone Z is much more often subject to a tightening than to an enlargement.

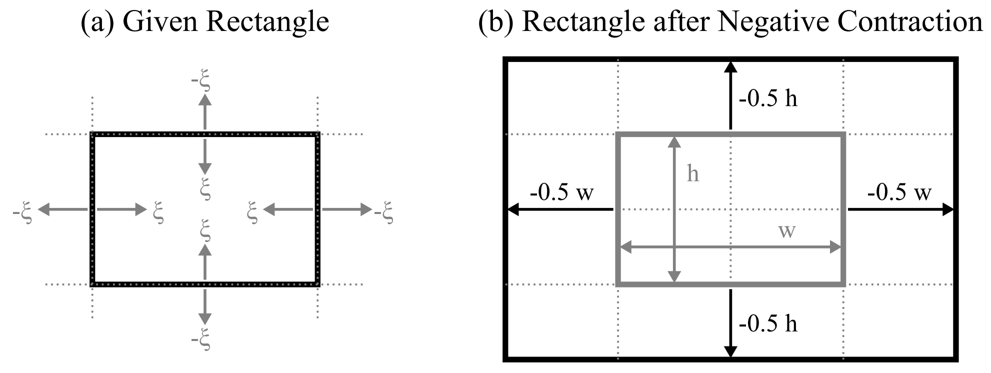


Figure 7.48: Relative enlargement of a given rectangle via negative contraction.

To generalize the contraction operation in regard of arbitrary polygons, the *contraction operator* $\circledast_\xi$ is introduced. Formally, describing the polygonal shape G as a closed polygonal chain, i.e., a sequence $G = (N_1, N_2, \dots)$, where each node N represents one vertex of G with coordinates (x_N, y_N), the contraction operator moves every node N to the new location N_{new} according to the directive

$$N_{\text{new}} = \Big((1-2\xi)\, x_N + \xi\,(\vdash G + \dashv G), (1-2\xi)\, y_N + \xi\,(\bot G + \top G)\Big). \tag{7.95}$$

This directive guarantees that the center point of the shape is not dislocated through the contraction. The proof can be given by inserting the center point's coordinates $x_N = \frac{1}{2}(\vdash G + \dashv G)$ and $y_N = \frac{1}{2}(\perp G + \top G)$ into equation 7.95, as this leads to the new location

$$N_{\text{new}} = \Big((1-2\xi)\,\tfrac{1}{2}(\vdash G + \dashv G) + \xi\,(\vdash G + \dashv G), (1-2\xi)\,\tfrac{1}{2}(\perp G + \top G) + \xi\,(\perp G + \top G)\Big) \quad (7.96)$$

$$= \Big((\tfrac{1}{2} - \tfrac{1}{2}2\xi + \xi)\,(\vdash G + \dashv G), (\tfrac{1}{2} - \tfrac{1}{2}2\xi + \xi)\,(\perp G + \top G)\Big) \quad (7.97)$$

$$= \Big(\tfrac{1}{2}\,(\vdash G + \dashv G), \tfrac{1}{2}\,(\perp G + \top G)\Big) \quad (7.98)$$

which is the same location as before. The preservation of the center point can also be observed in Figure 7.49, where a contraction operation with $\xi = -0.5$ is applied to a rectilinear octagon. Following the directive of equation 7.95, Table 7.9 lists how the coordinates of each node change from their old into their new values. Also shown is that –like in the example of Figure 7.48– the area of the polygon is quadruplicated in this case.

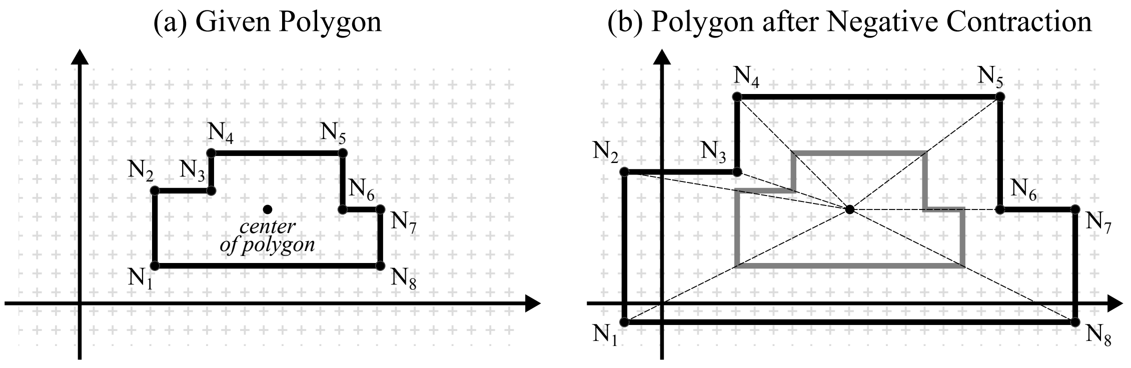


Figure 7.49: Relative enlargement of a given polygon via negative contraction.

Table 7.9: Coordinates of the octagon nodes from the negative contraction example of Figure 7.49.

Nodes	**Old**	**New**
N_1	$(4,2)$	$(-2,-1)$
N_2	$(4,6)$	$(-2,7)$
N_3	$(7,6)$	$(4,7)$
N_4	$(7,8)$	$(4,11)$
N_5	$(14,8)$	$(18,11)$
N_6	$(14,5)$	$(18,5)$
N_7	$(16,5)$	$(22,5)$
N_8	$(16,2)$	$(22,-1)$
center	$(10,5)$	$(10,5)$
area	60	240

By applying such a negative contraction, the enlargement of the layout zone's initial user-defined size $[Z_0]$ in the beginning of the self-organization –see Figure 7.47 (d)– can be correctly achieved. For that purpose, a *kickoff enlargement multiplier* $\varkappa \in \mathbb{R}$ with $\varkappa \geq 1$ allows to specify the size $[Z_1]$ of the layout zone for the 1st tightening-settlement cycle in relation to the *intrinsic minimum* $[⁙\mathcal{P}]$ (the sum of the participants' individual areas, as introduced in Definition 7.2). The kickoff enlargement multiplier is to be provided by the user as desired, considering the rule of thumb that Z should be made roughly thrice as large as the intrinsic minimum.

Internally, the respective contraction amount ξ must be calculated from the kickoff enlargement multiplier $\varkappa$. This relationship can be easily deduced from equation 7.94 as follows:

$$\frac{[Z_1]}{[Z_0]} = (1-2\xi)^2 \tag{7.99}$$

$$\frac{\varkappa \cdot [\boxdot\mathcal{P}]}{[Z_0]} = (1-2\xi)^2 \tag{7.100}$$

$$\sqrt{\frac{\varkappa \cdot [\boxdot\mathcal{P}]}{[Z_0]}} = 1-2\xi \tag{7.101}$$

$$\xi = \frac{1}{2} - \frac{1}{2}\sqrt{\frac{\varkappa \cdot [\boxdot\mathcal{P}]}{[Z_0]}}. \tag{7.102}$$

It should be mentioned, that no $\pm\sqrt{...}$ distinction must be made in the root extraction of equation 7.101 because the contraction amount is always $\xi < 0.5$ (which means that the term $1 - 2\xi$ above cannot be less than zero). An exemplary kickoff enlargement is given in Figure 7.50: the individual areas of the seven participants (a) add up to the intrinsic minimum $[\boxdot\mathcal{P}] = 18$, while the initial size of the layout zone (b) with width $w = 6$ and height $h = 4$ is $[Z_0] = 24$. For a kickoff enlargement of $\varkappa = 3$, the contraction amount is calculated as

$$\xi = \frac{1}{2} - \frac{1}{2}\sqrt{\frac{3\cdot 18}{24}} = \frac{1}{2} - \frac{1}{2}\sqrt{\frac{9}{4}} = \frac{1}{2} - \frac{1}{2}\frac{3}{2} = -\frac{1}{4}. \tag{7.103}$$

With this negative contraction, the new dimensions (c) of the layout zone are $w = 9$ and $h = 6$ such that its area becomes $[Z_1] = 9 \cdot 6 = 54$ which is thrice the intrinsic minimum, as desired.

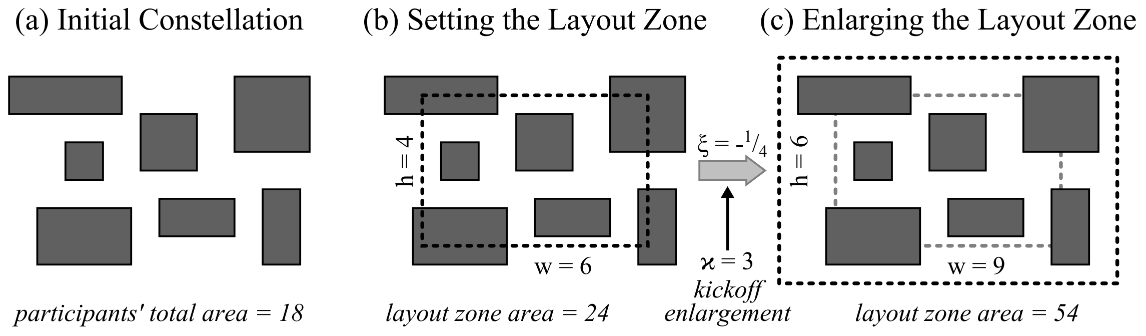

(a) Initial Constellation
(b) Setting the Layout Zone
(c) Enlarging the Layout Zone
h = 4
w = 6
ξ = -1/4
ϰ = 3
kickoff enlargement
h = 6
w = 9
participants' total area = 18
layout zone area = 24
layout zone area = 54

Figure 7.50: Exemplary illustration of a layout zone's kickoff enlargement.

In the initial module constellation, the participants are expected to be loosely scattered inside the kickoff zone Z_1 (as in the example of Figure 7.50). Just as well, the situation is fine if the participants are heaped up in the middle, overlapping each other. In that case, the participants will first disperse across Z_1 in the incipient tightening-settlement cycle before successively congregating in the remaining tightening-settlement cycles (as illustrated in Figure 7.4). However, one might also encounter quite the opposite case where the participants are initially located so far apart that some of them even lie beyond Z_1. In that situation, it may be preferable to overrule $\varkappa$ in order to enlarge the layout zone such that all participants are definitely *safe* before the interaction begins.

This can be achieved by determining the difference in height between the initial layout zone Z_0 and the constellation frame F (which defines the *vertically minimal* kickoff enlargement), as well as the respective difference in width (which defines the *horizontally minimal* kickoff enlargement). Including the contraction amount calculated from the user-specified kickoff enlargement multiplier $\varkappa$, the safe contraction amount ξ_{safe} can be obtained from the following formula:[13]

$$\xi_{\text{safe}} = \min\left(\underbrace{\frac{1}{2} - \frac{1}{2}\sqrt{\frac{\varkappa \cdot [\boxdot\mathcal{P}]}{[Z_0]}}}_{\substack{\text{user-specified}\\ \text{kickoff enlargement}}}, \underbrace{\frac{\top Z_0 - \top F}{\top Z_0 - \bot Z_0}, \frac{\bot F - \bot Z_0}{\top Z_0 - \bot Z_0}}_{\substack{\text{vertically minimal}\\ \text{kickoff enlargement}}}, \underbrace{\frac{\dashv Z_0 - \dashv F}{\dashv Z_0 - \vdash Z_0}, \frac{\vdash F - \vdash Z_0}{\dashv Z_0 - \vdash Z_0}}_{\substack{\text{horizontally minimal}\\ \text{kickoff enlargement}}}\right). \tag{7.104}$$

Figure 7.51 gives an example. Presented in image (a) are eight participants with a total area of $[\boxdot\mathcal{P}] = 44$, with the initial layout zone Z_0 being a 6 by 8 rectangle and the constellation frame F having a width of 12 and a height of 14. For a user-specified kickoff enlargement multiplier $\varkappa = 3$, the contraction amount is $\xi = \frac{1}{2} - \frac{1}{2}\sqrt{(3 \cdot 44)/(6 \cdot 8)} = -0.33$, but one participant would be completely lost thereby while three participants would be prone, as can be seen in image (b). With the vertically minimal kickoff enlargement $\varkappa = 3.34$ –achieved via $\xi = \frac{-3}{8} = -0.375$ and illustrated in image (c)– two participants would still be prone. However, a contraction amount of $\xi = \frac{-3}{6} = -0.5$ leads to the horizontally minimal kickoff enlargement $\varkappa = 4.36$ which finally produces a zone size for Z_1 where all participants are safe, as shown in image (d), and thus represents the safe contraction amount in this case.

(a) Layout Zone and Constellation Frame (b) User-specified Kickoff Enlargement (c) Vertically Minimal Kickoff Enlargement (d) Horizontally Minimal Kickoff Enlargement

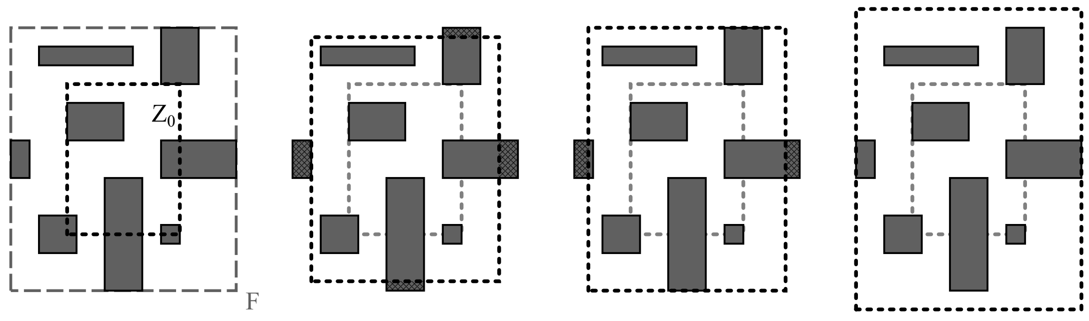


Figure 7.51: Safe contraction amount, corresponding to the horizontally minimal kickoff enlargement.

7.4.1.2 Tightening the Layout Zone

Now that a negative contraction provided the kickoff enlargement which turned Z_0 into Z_1, the module interaction begins. Therein, the interaction control organ applies positive contraction operations to the layout zone in order to achieve a tightening of Z_1, Z_2, Z_3 and so on after each settlement. Here, the respective contraction amount defines how rigorous the *tightening policy* is – considering the following two tenets, which represent two extremes in this regard:

- First, the tightening policy should sufficiently "put the screws" on the participants, which is to say that at least one participant should be *prone* after each tightening. Otherwise, i.e., if all participants are completely *safe* after each tightening, the tightening policy is too *lenient* and the layout zone does never reach its initial size but will instead approach a limit much greater than $[Z_0]$. This would even be true in the hypothetical case that the minimal movement distance m is set to zero. The final layout zone size approached by Z would depend on the constellation itself, as exemplified in Figure 7.52.
- Second, the tightening policy should refrain from overpacing, which is to say that no participant should be *lost* after a tightening. Otherwise, the lost participants are forced to perform a *Re-entering* move instead of being able to choose the best action from a set of alternative options. Altough this is not a dealbreaking issue, such a tightening policy can in general be considered too *aggressive* as it tends to bring about unwanted chaotic behavior.

[13]In fact, the two terms for the vertically minimal kickoff enlargement are redundant in equation 7.104 because the layout zone Z_0 is centered on the constellation frame F. The same is true regarding the two terms for the horizontally minimal kickoff enlargement.

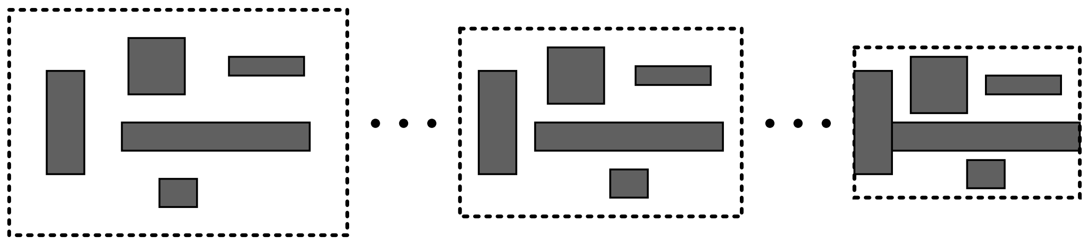


Figure 7.52: If the tightening policy is too lenient, the desired zone size cannot be reached.

Whether the tightening policy is too lenient or too aggressive, the ultimate implications are the same: the module interaction does not converge towards the desired goal. Hence, for every i^{th} tightening-settlement cycle (with $i \in \mathbb{N}^{+}$) the two tenets above define the task of finding a contraction amount ξ_i which tightens the layout zone from Z_i into Z_{i+1} after the settlement such that

$$\forall P \in \mathcal{P},\ (\exists P : \Psi_P \sqsubset \square P) \wedge (\nexists P : \Psi_P = \square P). \tag{7.105}$$

To facilitate a concise specification of the tightening policy's rigorousness, the *pressing rate* $\varpi \in \mathbb{R}$ in the open interval $(0, 1)$ is introduced. It allows to set ξ_i depending on the desired *protrusion extent* ψ of the participants (see Section 7.3.1.3), wherein each ψ is calculated relative to the participant's dimensions. This means, ξ_i is chosen such that the layout zone is tightened as much as possible without letting any participant P have a horizontal protrusion extent $\psi_{x,P}$ larger than ϖ-times the width w_P of P, nor a vertical protrusion extent $\psi_{y,P}$ larger than ϖ-times the height h_P of P.

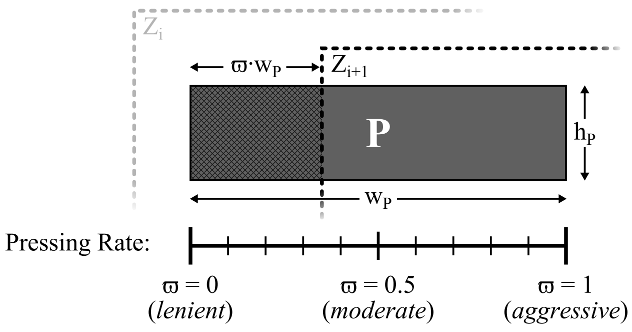


Figure 7.53: Visualization of a tightening policy's rigorousness, depending on the pressing rate.

In this manner, the pressing rate makes it possible to fine-tune the tightening policy in a continuous range between the two excluded endpoints $\varpi = 0$ (lenient) and $\varpi = 1$ (aggressive), as illustrated in Figure 7.53. In general, it is sensible to work with a pressing rate near $\varpi = 0.5$, which is referred to as a *moderate* tightening policy. For a given ϖ, the interaction control organ's consequential task per each tightening is to minimize the area of the layout zone while making sure that

$$(\forall P \in \mathcal{P},\ \psi_{x,P} \leq \varpi \cdot w_P \wedge \psi_{y,P} \leq \varpi \cdot h_P) \wedge (\exists P \in \mathcal{P} : \psi_{x,P} = \varpi \cdot w_P \vee \psi_{y,P} = \varpi \cdot h_P). \tag{7.106}$$

This is achieved by calculating the respective contraction amount ξ_i according to the following formula:

$$\begin{aligned}\xi_i = \min\Bigg(&\min_{\forall P \in \mathcal{P}} \Bigg(\overbrace{\frac{\top Z_i - \top(\square P) + \varpi \top(\square P) - \varpi \bot(\square P)}{\top Z_i - \bot Z_i}}^{\text{northern edge}} \Bigg), \min_{\forall P \in \mathcal{P}} \Bigg(\overbrace{\frac{\bot(\square P) + \varpi \top(\square P) - \varpi \bot(\square P) - \bot Z_i}{\top Z_i - \bot Z_i}}^{\text{southern edge}} \Bigg), \\ &\min_{\forall P \in \mathcal{P}} \Bigg(\underbrace{\frac{\dashv Z_i - \dashv(\square P) + \varpi \dashv(\square P) - \varpi \vdash(\square P)}{\dashv Z_i - \vdash Z_i}}_{\text{eastern edge}} \Bigg), \min_{\forall P \in \mathcal{P}} \Bigg(\underbrace{\frac{\vdash(\square P) + \varpi \dashv(\square P) - \varpi \vdash(\square P) - \vdash Z_i}{\dashv Z_i - \vdash Z_i}}_{\text{western edge}} \Bigg) \Bigg).\end{aligned} \tag{7.107}$$

This formula bears some resemblance to the formula for ξ_{safe} in Section 7.4.1.1, because the first two fractions here (which concern the northern edge and the southern edge) represent the equivalent to those of the vertically minimal kickoff enlargement in equation 7.104 while the other two fractions (eastern edge and western edge) correspond to those of the horizontally minimal kickoff enlargement therein. However, it is important to note that –in contrast to equation 7.104– the terms are not redundant here for two reasons: (1) the layout zone is not perfectly centered on the constellation since the minimal movement distance is larger than zero, and (2) the maximally permitted protrusion extent is relative to a participant's dimensions, as already mentioned above.

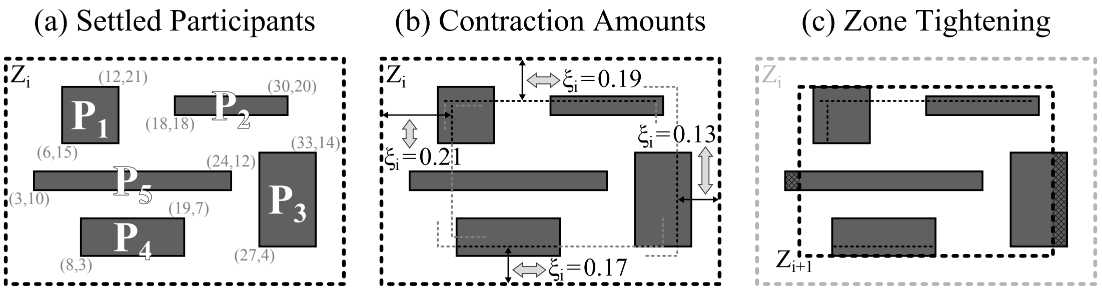


Figure 7.54: Exemplary illustration of a zone tightening, calculated from the pressing rate.

Table 7.10: Bounding box coordinates of the objects in the tightening example of Figure 7.54.

Object	**Bounding Box**
Z	$((0,0),(36,24))$
P_1	$((6,15),(12,21))$
P_2	$((18,18),(30,20))$
P_3	$((27,4),(33,14))$
P_4	$((8,3),(19,7))$
P_5	$((3,10),(24,12))$

To illustrate the calculation of the contraction amount via equation 7.107, Figure 7.54 shows an exemplary zone tightening for a constellation of five settled participants inside a layout zone Z_i, as given in image (a). The coordinates of the layout zone rectangle and those of the particicpants' bounding boxes are given in Table 7.10. With a –rather lenient– pressing rate of $\varpi = 0.25$ the contraction amount according to equation 7.107 becomes

$$\begin{aligned}\xi_i = \min\big(&\min(\overbrace{0.19, 0.19}^{P_1,P_2}, 0.52, 0.75, 0.52), \min(0.69, 0.77, 0.27, \overbrace{0.17}^{P_4}, 0.44),\\ &\min(0.71, 0.25, \underbrace{0.13}_{P_3}, 0.55, 0.48), \min(\underbrace{0.21}_{P_1}, 0.58, 0.79, 0.3, 0.23)\big)\end{aligned} \tag{7.108}$$

where the five values in each of the four ancillary min-terms correspond to the five participants (P_1, P_2, P_3, P_4, P_5) respectively. In the first term (concerning the northern edge), the smallest contraction amount is given by P_1 and P_2, in the remaining terms it is given by P_4 (southern edge), P_3 (eastern edge), and again P_1 (western edge), as indicated in image (b). Among these four values, the smallest one is

$$\xi_i = \min(0.19, 0.17, 0.13, 0.21) = 0.13 \tag{7.109}$$

which is the contraction amount construed from the eastern edge. Using this value, image (c) shows how the layout zone is tightened into Z_{i+1}. As can be seen, P_3 now has a horizontal protrusion extent of $\psi_{x,P_3} = \varpi \cdot w_{P_3}$ while all other participants have lesser protrusion extents, which means that condition 7.106 is fulfilled.

7.4.1.3 Considering Rectilinear Layout Zones

The tightening formulas described so far work well for rectangular layout zones. But for rectilinear (i.e., nonrectangular) layout zones one can inadvertently ensnarl in situations where condition 7.106 is violated. This is not overly problematic because the surplus protrusion extent usually is rather small; and even if a participant is completely lost, a *Re-entering* move helps to bring it back into the layout zone. On this account, the tightening formulas above can be adequately used for rectangular as well as for rectilinear layout zones. Implementing a safe tightening procedure (where all participants definitely adhere to the maximally permitted protrusion extent) is more elaborate in the rectilinear case. To explicate this topic, the following ruminations sketch out the subtleties that must be paid attention to.[14]

Figure 7.55 (a) displays a settlement of seven participants inside a hexagonal layout zone Z_i. Due to the notch, there are two edges (named E_1 and E_2) which deviate from the bounding box $\square Z_i$. Applying the formula of equation 7.107 for a pressing rate of $\varpi = 0.5$ would lead to the tightened layout zone Z^*_{i+1} by which P_2 violates its permitted protrusion extent ψ_{y,P_2} in vertical direction. This violation – referred to as the *protrusion excess*– is colored deep black in image (b). To obtain the correct contraction amount, equation 7.107 must be enhanced such that it considers all edges of the layout zone. However, one further issue must be taken into account: when determining a potential contraction amount with respect to a particular edge E and a particular participant P, it may be that E does not even touch P because the length of E is smaller than the layout zone's respective dimension (i.e., the width of Z_i if E is a horizontal edge, the height of Z_i if E is a vertical edge). In that case, the potential contraction amount is to be discarded. Image (b) exemplifies such a situation: calculatorily, P_1 and E_2 determine the maximal contraction amount, but it is discarded since E_2 remains completely aloof from P_1 (not only for that contraction amount, but even regardless of ξ – the trajectory in the image visualizes how the first node of E_2 wanders distant to P_1 when scaling the layout zone). Next in size is the contraction amount determined by P_2 and E_2. In contrast to the (P_1, E_2) pair, E_2 is in touch with P_2 for that contraction amount and is therefore called *afferent* to P_2. In fact, this value is the smallest among all determinable contraction amounts and thus represents the desired ξ_i. The resultant zone tightening is illustrated in image (c): as can be seen, the desired protrusion extent is now met by P_2 and undercut by all other participants as should be.

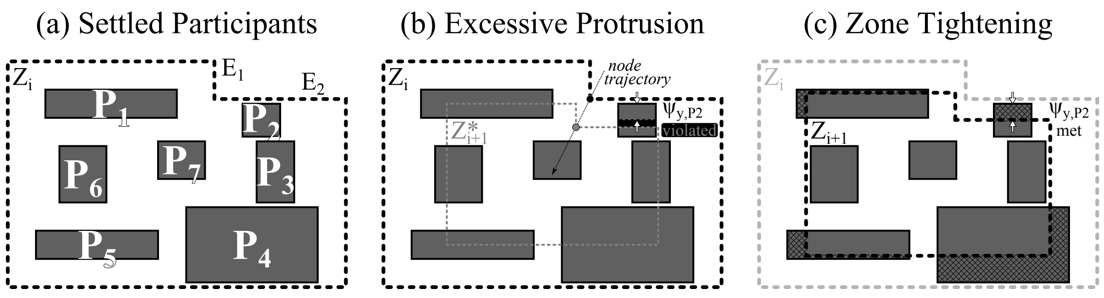


Figure 7.55: Exemplary illustration of a zone tightening for a rectilinear layout zone.

For rectilinear layout zones, another circumstance has to be contemplated regarding the fact that a participant can protrude the layout zone at one of its concave vertices due to a tightening. But first, row [a] in Figure 7.56 discriminates the possible cases of protrusion at a convex vertex as known from rectangular layout zones. Each of these cases represents a subsidiary offshoot of case (d2) in Figure 7.23. For a supposed pressing rate of $\varpi = 0.5$, neither ψ_x nor ψ_y violate the permitted protrusion extent in image [a1] of Figure 7.56. Hence, both are said to be *admissible*, which makes the overall protrusion Ψ admissible as well. In [a2], both ψ_x and ψ_y violate the permitted protrusion extent and are therefore called *excessive* (in line with the blackened protrusion excess outlined in the image). In that case, the overall protrusion is also excessive, i.e., the inherent contraction amount is too large and the tightening would be more rigorous than desired. In [a3], ψ_x is excessive while ψ_y is admissible; in [a4], the situation is vice versa. It goes without saying, that in those two cases again the overall protrusion is excessive.

If the protrusion occurs at a concave vertex (see row [b] in Figure 7.56, covering the different offshoots of case (d1) in Figure 7.23), the affair is the same as above if both ψ_x and ψ_y are admissible [b1] or if both are excessive [b2]. However, things are different if the protrusion extent is only violated in

[14]Without going into a detailed elaboration, the considerations made in this Section 7.4.1.3 about the tightening of a rectilinear layout zone can likewise be applied to its kickoff enlargement.

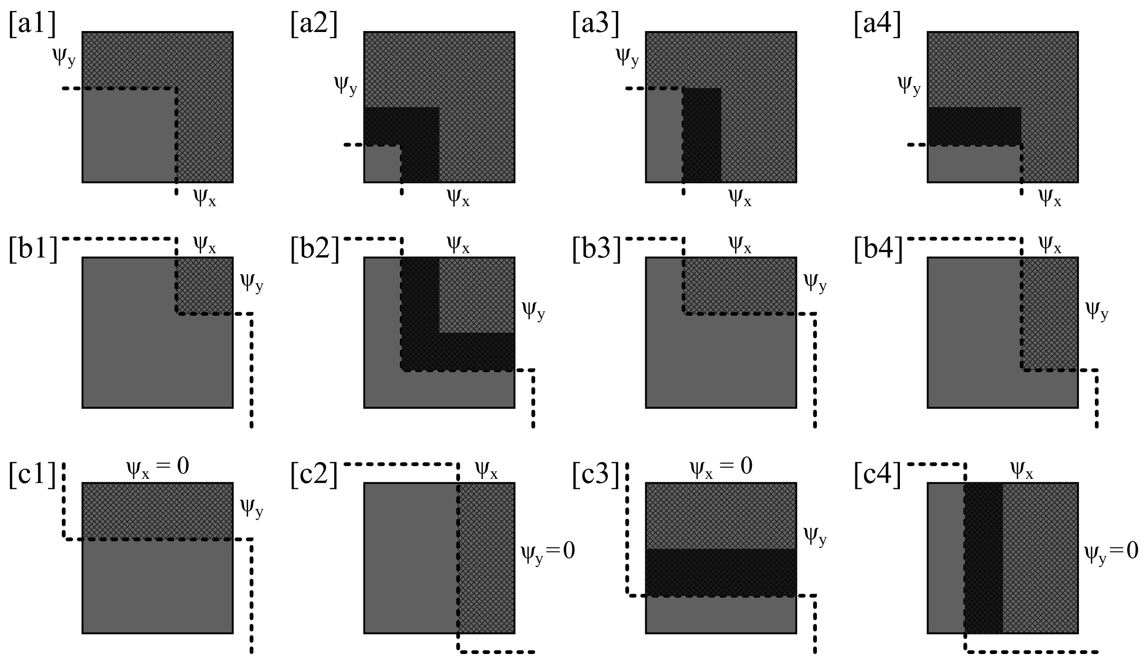


Figure 7.56: Cases of protrusion after a tightening: [a] convex vertex, [b] concave vertex, [c] edge.

one direction. In [b3], ψ_x is excessive but the overall protrusion is admissible because ψ_y is admissible. In [b4], the overall protrusion is admissible since ψ_x is admissible although ψ_y is excessive. These two cases are justified by looking at row [c], which displays the offshoots of case (c2) from Figure 7.23, where the protrusion does not occur at a vertex but at an edge of the layout zone. In [c1], the protrusion is admissible because ψ_y is admissible. Although the participant protrudes the layout zone with its entire width, the horizontal protrusion extent is $\psi_x = 0$ (according to equation 7.53). The anomaly with this reckoning is, that in case [b3] –where ψ_y is the same and ψ_x is much greater– the protrusion area is less than in [c1]. Yet, this conclusion logically justifies why the protrusion is admissible in [b3]. The preceding considerations about [b3] and [c1] can be applied to [b4] and [c2] in the orthogonal direction to reason that the protrusion in [b4] is admissible as well. For the sake of completeness, the remaining two images in Figure 7.56 present cases of excessive edge protrusion through ψ_y in [c3] and through ψ_x in [c4].

Table 7.11 lists the admissibility/excessiveness for the three superordinate cases of protrusion in a matrix-like fashion, depending on the dimensionality of the protrusion excess (i.e., whether there is excess neither through ψ_x nor through ψ_y, excess both by ψ_x and ψ_y, or excess in either ψ_x or ψ_y exclusively). Following [334], protrusion at a convex vertex is denoted as *ear protrusion* while protrusion at a concave vertex is referred to as *mouth protrusion* in the table.

Table 7.11: Cases of protrusion after a tightening. The signifiers in square brackets refer to the images in Figure 7.56.

	Protrusion Excess		
	none	ψ_x **and** ψ_y	ψ_x **or** ψ_y
Ear Protrusion (convex vertex)	admissible [a1]	excessive [a2]	excessive [a3],[a4]
Mouth Protrusion (concave vertex)	admissible [b1]	excessive [b2]	admissible [b3],[b4]
Edge Protrusion	admissible [c1],[c2]	*not applicable*	excessive [c3],[c4]

The findings of Table 7.11 deliver an important insight: in the case of mouth protrusion, the greater one of the two potential contraction amounts is to be favored over the lesser one. This is in contrast to equation 7.107 (where the smallest value for ξ_i was sought) and further distinguishes it from the case of ear protrusion, where the lesser contraction amount is to be taken instead of the greater one. To illuminate this statement, Figure 7.57 (a1) shows a participant P next to an ear (convex vertex) of the layout zone Z_i. The two edges E_1 and E_2 of the ear offer two potential contraction amounts. For a $\varpi = 0.5$ pressing rate, image (a2) illustrates the potential tightening into Z^*_{i+1} based on the ear's horizontal edge E_1. As can be seen, the overall protrusion for this contraction amount ξ_i^{P,E_1} is admissible. But based on the ear's vertical edge E_2, the other potential tightening entails an overall excessive protrusion (a3). Hence, the respective contraction amount ξ_i^{P,E_2} is to be rejected in favor of the smaller ξ_i^{P,E_1}. In (b1) however, the participant lies next to a mouth (concave vertex) of Z_i. As Table 7.11 tells right away, both potential tightenings (b2) and (b3) definitely lead to an overall admissible protrusion – the protrusion *cannot* be excessive in these cases because either ψ_x (b2) or ψ_y (b3) is congenitally admissible. For that reason, the larger contraction amount (ξ_i^{P,E_2} in this example) may overtrump the smaller one (ξ_i^{P,E_1} here) in the case of mouth protrusion.

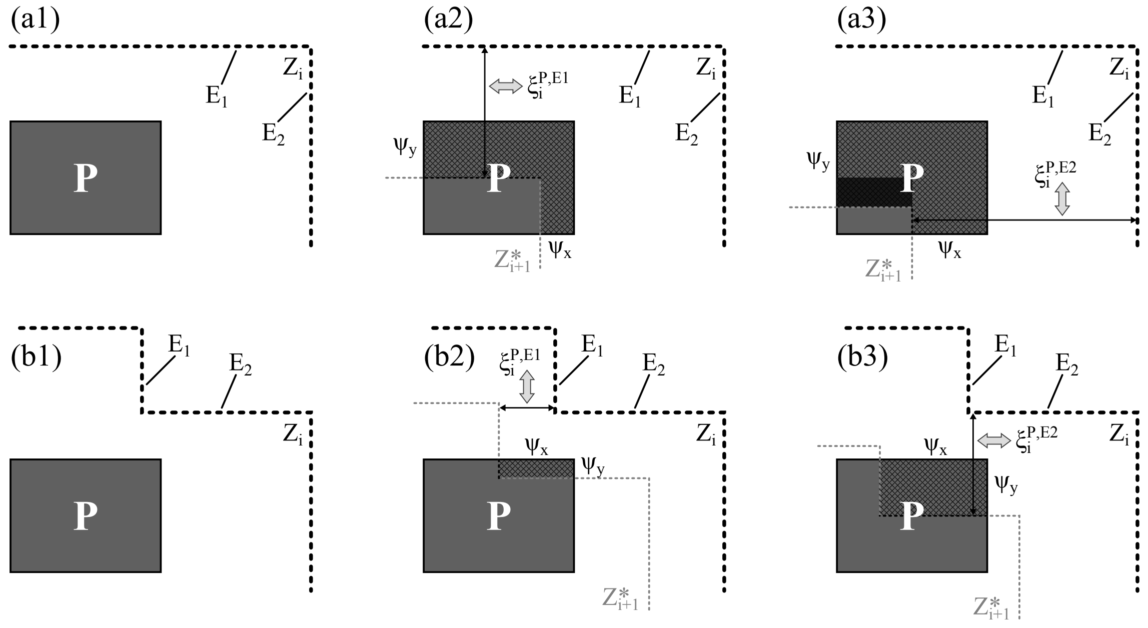


Figure 7.57: Depending on the case of protrusion, the smaller or the greater contraction amount matters.

In a nutshell, four issues have been revealed in this Section 7.4.1.3: for the tightening of an arbitrarily rectilinear layout zone (1) all possible participant/edge pairs have to be regarded, (2) it is necessary to check whether the respective edge is afferent to the participant, (3) it must be determined if the prospective protrusion of the participant is admissible at all, and (4) in the case of mouth protrusion the larger contraction amount is to be favored over the smaller one. Each of these issues can be found in Algorithm 4, which provides a pseudocode description of the contraction amount calculation in the rectilinear case. A couple of even further details need to be paid attention to, as the following explanation of Algorithm 4 points out.

Explanation of Algorithm 4

First of all, a set Ξ_{all} is initialized in line 1 that will later be used to gather all potential contraction amounts. Then, for each participant P among the set $\mathcal{P}$ of all participants, every node N of the layout zone Z_i is traversed. Every node N joins two edges: a horizontal edge denoted as $E_{\bar{h}}^N$ and a vertical edge referred to as E_v^N. If $E_{\bar{h}}^N$ is a northern edge[15] of Z_i whose vertical coordinate $y_{E_{\bar{h}}^N}$ lies above the center of Z_i, then the potential contraction amount $\xi_i^{P,E_{\bar{h}}^N}$ is calculated as shown in line 6. If $E_{\bar{h}}^N$ is a

[15]To determine if a horizontal edge $E_{\bar{h}}$ of a rectilinear polygon is a northern edge of that polygon, one can simply draw a vertical line through $E_{\bar{h}}$. If the line crosses an *even* number of other horizontal edges *above* $E_{\bar{h}}$ (and thus an *odd* number

southern edge of Z_i and lies below the center of Z_i, then the potential contraction amount is calculated as in line 8. If neither is true, then $\xi_i^{P,E_{\bar{h}}{}^N}$ is set to a null value (line 10) – no contraction amount is calculated in this case because it would falsely lead to an enlargement of the layout zone instead of a tightening, as illustrated in Figure 7.58 (a).[16] The formulas in line 6 and line 8 correspond to the terms found in equation 7.107 for a rectangular layout zone's northern edge and southern edge. However, there is one pivotal difference: in the rectilinear case, the contraction amount must be projected from the considered edge onto the respective side of the layout zone's bounding box. For that purpose, the *projection divisor* χ is introduced. Figure 7.59 amplifies this on a rectilinear layout zone Z_i with height h_{Z_i}. For any contraction amount ξ_i, the absolute *edge displacement* of the layout zone's northernmost and southernmost edge is $\epsilon = \xi_i \cdot h_{Z_i}$. Since edge E is nearer to the center point of Z_i, its edge displacement ϵ_E is smaller by the projection divisor χ:

$$\epsilon_E = \frac{\epsilon}{\chi}. \tag{7.110}$$

Looking at the two plotted node trajectories and the horizontal line through the center point of Z_i, all three of which are intercepted by the pair of auxiliary parallels drawn into Figure 7.59, the projection divisor can be deduced from the relations between the four line segments a, b, c, d via Thales' theorem (intercept theorem) as follows:

$$\frac{a}{b} = \frac{c}{d} \tag{7.111}$$

$$\frac{a}{c} = \frac{b}{d} \tag{7.112}$$

$$1 - \frac{a}{c} = 1 - \frac{b}{d} \tag{7.113}$$

$$\frac{c-a}{c} = \frac{d-b}{d} \tag{7.114}$$

$$\frac{c-a}{d-b} = \frac{c}{d} \tag{7.115}$$

$$\frac{c-a}{d-b} + 1 = \frac{c}{d} + 1 \tag{7.116}$$

$$\frac{c-a+d-b}{d-b} = \frac{c+d}{d} \tag{7.117}$$

wherein the four terms of the fractions can be substituted to produce the following equation

$$\frac{\epsilon}{\epsilon_E} = \frac{\frac{1}{2}\, h_{Z_i}}{y_E - \frac{1}{2}(\bot Z_i + \top Z_i)} \tag{7.118}$$

which represents the projection divisor χ. To calculate the projected contraction amount from a given edge displacement ϵ_E, equation $\epsilon = \xi_i \cdot h_{Z_i}$ can now be rearranged into

$$\xi_i = \frac{\epsilon}{h_{Z_i}} = \frac{\epsilon_E \cdot \chi}{h_{Z_i}} = \frac{\epsilon_E}{h_{Z_i}} \cdot \frac{\frac{1}{2}\, h_{Z_i}}{y_E - \frac{1}{2}(\bot Z_i + \top Z_i)} = \frac{\frac{1}{2}\, \epsilon_E}{y_E - \frac{1}{2}(\bot Z_i + \top Z_i)} \tag{7.119}$$

$$= \frac{\epsilon_E}{2 \cdot y_E - \bot Z_i - \top Z_i}. \tag{7.120}$$

of other horizontal edges *below* $E_{\bar{h}}$), then $E_{\bar{h}}$ is a northern edge. Otherwise, the line would cross an *odd* number of other horizontal edges *above* $E_{\bar{h}}$ (and an *even* number of other horizontal edges *below* $E_{\bar{h}}$), indicating that $E_{\bar{h}}$ is a southern edge.

[16] As desired, the potential contraction amount calculated from the (P_1, E_1) pair in Figure 7.58 (a) would result in a tightening because E_1 is a northern edge of Z_i above its center point. In contrast, the (P_2, E_2) pair would lead to an enlargement because E_2 is a northern edge below the center point of Z_i. For that reason, E_2 is to be ignored in the calculation of the contraction amount. One might say that E_2 should not be completely ignored because the (P_3, E_2) pair would then be elided – yet, this *is* legitimate because the (P_3, E_3) pair would define the correct contraction amount in that case.
Analogous to these considerations about northern edges below the center of the layout zone, southern edges above the center of the layout zone also need to be ignored in the calculation of the contraction amount for the same reason.

For a northern edge of Z_i and a participant P, the absolute edge displacement is

$$\epsilon_E = y_{E_{\bar{h}}} - \top(\Box P) + \varpi \cdot \top(\Box P) - \varpi \cdot \bot(\Box P) \tag{7.121}$$

which explicates line 6 in Algorithm 4. Since $y_{E_{\bar{h}}}$ corresponds to $\top Z_i$ in the dividend of the northern-edge term in equation 7.107, the formula in Algorithm 4 differs from that term by the factor

$$\frac{\top Z_i - \bot Z_i}{2 \cdot y_E - \bot Z_i - \top Z_i} = \frac{h_{Z_i}}{2 \cdot y_E - \bot Z_i - \top Z_i} = \frac{\frac{1}{2}\, h_{Z_i}}{y_E - \frac{1}{2}(\bot Z_i + \top Z_i)} \tag{7.122}$$

which –as expected– is precisely the projection divisor χ deduced above (see equation 7.118). For a southern edge of Z_i, the projection divisor becomes

$$\chi = \frac{\frac{1}{2}\, h_{Z_i}}{\frac{1}{2}(\bot Z_i + \top Z_i) - y_E} \tag{7.123}$$

which is reflected in line 8 of Algorithm 4. Considering the vertical edge E_v^N of the traversed node N, the formulas in line 14, line 16, and line 18 –which are analogous to those from the horizontal edge $E_{\bar{h}}{}^N$ (line 6, line 8, and line 10 respectively)– provide the potential contraction amount ξ_i^{P,E_v^N} for the (P, E_v^N) pair. Equivalently, if E_v^N is an eastern edge to the left of Z_i's center or if E_v^N is a western edge to the right of Z_i's center, then it is discarded because it would lead to an enlargement of the layout zone (as visualized in Figure 7.58 (b)). It should be noted that the calculations in line 6, line 8, line 14, and line 16 cannot run into a *division by zero* because the respective if-conditions guarantee that $y_{E_{\bar{h}}{}^N} \neq \frac{1}{2}(\bot Z_i + \top Z_i)$ and $x_{E_v^N} \neq \frac{1}{2}(\vdash Z_i + \dashv Z_i)$.

Next (line 21 in Algorithm 4), those contraction amounts among $\{\xi_i^{P,E_{\bar{h}}{}^N}, \xi_i^{P,E_v^N}\}$ that have indeed been calculated (i.e., are not `null`) will be stored in a set $\Xi_{P,N}$ if the respective edge E^N is *afferent* to P and if the prospective protrusion Ψ_P^{new} is *admissible*. If $\Xi_{P,N}$ is not empty, the appropriate contraction amount must be chosen from that set. In the case that node N is a mouth, then the larger value found in $\Xi_{P,N}$ is added to the set Ξ_{all} in line 24 (concordant with the ruminations made in Figure 7.57); otherwise, N is an ear and the smaller value in $\Xi_{P,N}$[17] is taken (line 26). After all nodes have been traversed for each participant, the smallest value from the set Ξ_{all} is chosen in line 31 as the final contraction amount ξ_i for the subsequent tightening of the rectilinear layout zone.

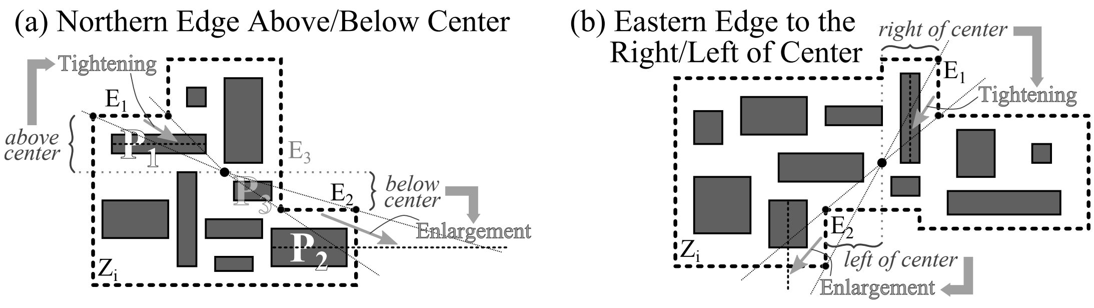


Figure 7.58: Certain edges of a rectilinear layout zone –such as (a) northern edges below the zone center and (b) eastern edges to the left of the zone center– need to be discarded when calculating the contraction amount.

7.4.2 Transient Tightening Policies

Although the successive tightening based on a single pressing rate ϖ is a quite dynamic ingredient of a SWARM run, it can still be regarded as being a bit too coarse because the layout zone gets tightened

[17] If node N is an ear, then $\Xi_{P,N}$ is in fact a *singleton* here. To understand the reason for this, two cases are to be distinguished: if the node would lie aloof from participant P after the prospective tightening, one of the two potential contraction amounts has been discarded in line 21 because the respective edge is not *afferent* to P; otherwise (i.e., if the node would lie on P after the tightening) one of the two potential contraction amounts has been discarded in line 21 because the prospective protrusion of P would not be *admissible*.

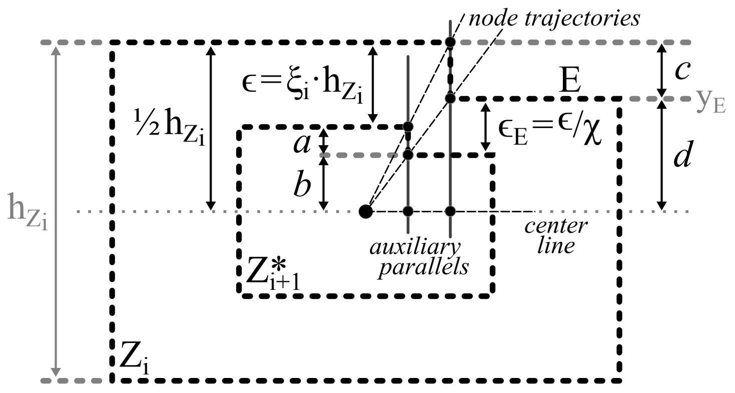


Figure 7.59: Exemplification of edge projection, as necessary in the case of a rectilinear layout zone.

Algorithm 4 Contraction amount for the tightening of a rectilinear layout zone

```
$\Xi_{\text{all}} \leftarrow \{\}$
**for each** participant $P \in \mathcal{P}$ **do**
    **for each** node $N$ of $Z_i$ **do**
        ▷ **Horizontal edge of the node ($E_{\bar{h}}{}^N$):**
        **if** $E_{\bar{h}}{}^N$ is a northern edge of $Z_i$ **AND** $y_{E_{\bar{h}}{}^N} > \frac{1}{2}(\perp Z_i + \top Z_i)$ **then**
            $\xi_i^{P,E_{\bar{h}}{}^N} \leftarrow (y_{E_{\bar{h}}{}^N} - \top(\square P) + \varpi\top(\square P) - \varpi\perp(\square P))/(2 \cdot y_{E_{\bar{h}}{}^N} - \perp Z_i - \top Z_i)$
        **else if** $E_{\bar{h}}{}^N$ is a southern edge of $Z_i$ **AND** $y_{E_{\bar{h}}{}^N} < \frac{1}{2}(\perp Z_i + \top Z_i)$ **then**
            $\xi_i^{P,E_{\bar{h}}{}^N} \leftarrow (\perp(\square P) + \varpi\top(\square P) - \varpi\perp(\square P) - y_{E_{\bar{h}}{}^N})/(\perp Z_i + \top Z_i - 2 \cdot y_{E_{\bar{h}}{}^N})$
        **else**
            $\xi_i^{P,E_{\bar{h}}{}^N} \leftarrow \texttt{null}$
        **end if**
        ▷ **Vertical edge of the node ($E_v^N$):**
        **if** $E_v^N$ is an eastern edge of $Z_i$ **AND** $x_{E_v^N} > \frac{1}{2}(\vdash Z_i + \dashv Z_i)$ **then**
            $\xi_i^{P,E_v^N} \leftarrow (x_{E_v^N} - \dashv(\square P) + \varpi\dashv(\square P) - \varpi\vdash(\square P))/(2 \cdot x_{E_v^N} - \vdash Z_i - \dashv Z_i)$
        **else if** $E_v^N$ is a western edge of $Z_i$ **AND** $x_{E_v^N} < \frac{1}{2}(\vdash Z_i + \dashv Z_i)$ **then**
            $\xi_i^{P,E_v^N} \leftarrow (\vdash(\square P) + \varpi\dashv(\square P) - \varpi\vdash(\square P) - x_{E_v^N})/(\vdash Z_i + \dashv Z_i - 2 \cdot x_{E_v^N})$
        **else**
            $\xi_i^{P,E_v^N} \leftarrow \texttt{null}$
        **end if**
        ▷ **Memorizing the appropriate contraction amount**
        $\Xi_{P,N} \leftarrow \left\{\xi \in \{\xi_i^{P,E_{\bar{h}}{}^N}, \xi_i^{P,E_v^N}\} \mid \xi \neq \texttt{null} \wedge E^N \text{ is afferent to } P \wedge \Psi_P^{\text{new}} \text{ is admissible}\right\}$
        **if** $|\Xi_{P,N}| > 0$ **then**
            **if** $N$ is a mouth **then**
                $\Xi_{\text{all}} \leftarrow \Xi_{\text{all}} \uplus \max(\Xi_{P,N})$
            **else**
                $\Xi_{\text{all}} \leftarrow \Xi_{\text{all}} \uplus \min(\Xi_{P,N})$
            **end if**
        **end if**
    **end for**
**end for**
$\xi_i \leftarrow \min(\Xi_{\text{all}})$
```

only *after* each individual settlement. This manner of *abrasive tightening* can be visualized in form of a staircase-shaped *tightening profile*, as exemplarily done in Figure 7.60: every tightening (henceforth also denoted as a *major tightening*) can be recognized as an abrupt change in the size of the layout zone. To obtain a more fine-grained tightening characteristic, the idea of *transient tightening policies* is introduced. A transient tightening policy is a tightening policy where changes in the size of the layout zone may occur *during* a tightening-settlement cycle. Basically, two kinds of transient tightening policies can be conceived: a *progressive tightening policy* and a *regressive tightening policy*, both of which will be discussed in Section 7.4.2.1 and Section 7.4.2.2 respectively.

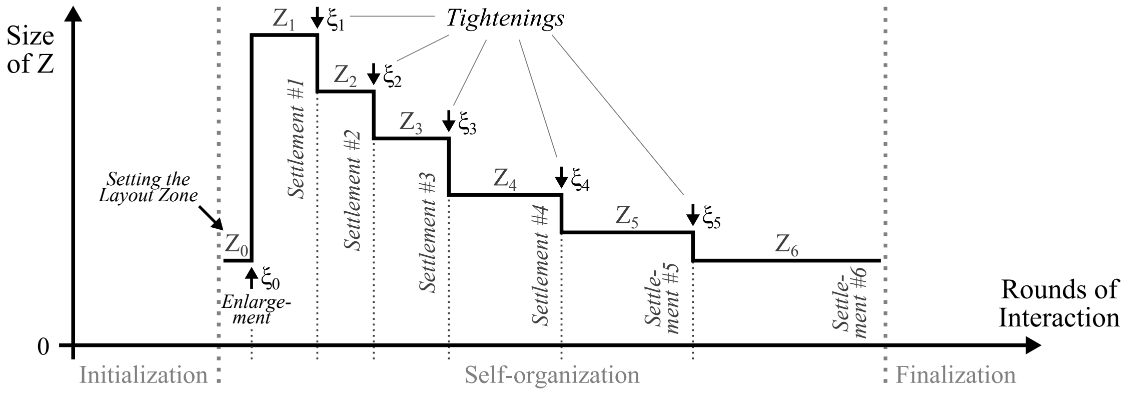


Figure 7.60: Exemplary tightening profile of an abrasive tightening policy.

7.4.2.1 Progressive Tightening

A *progressive tightening policy* is a transient tightening policy where each major tightening is in fact chopped up into several *minor tightenings*. Thus, the tightening policy is not specified solely through the pressing rate ϖ, but may for example be defined via (ϖ, u) wherein $u \in \mathbb{N}^+$ denotes the number of minor tightenings. With this tightening policy, every contraction amount ξ_i that was calculated from the desired pressing rate ϖ is simply divided into u minor contraction amounts $\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,u}$ all of which reduce the area of the layout zone by the same magnitude. These minor contraction amounts are then successively applied to the layout zone –one after each round of interaction– until u minor tightenings have been performed. In the remaining rounds of interaction, the layout zone is not tightened any further before all participants have settled. The respective tightening profile of this *linear progressive tightening policy* looks like the one displayed in Figure 7.61 for an exemplary value of $u = 4$ as the number of minor tightenings.

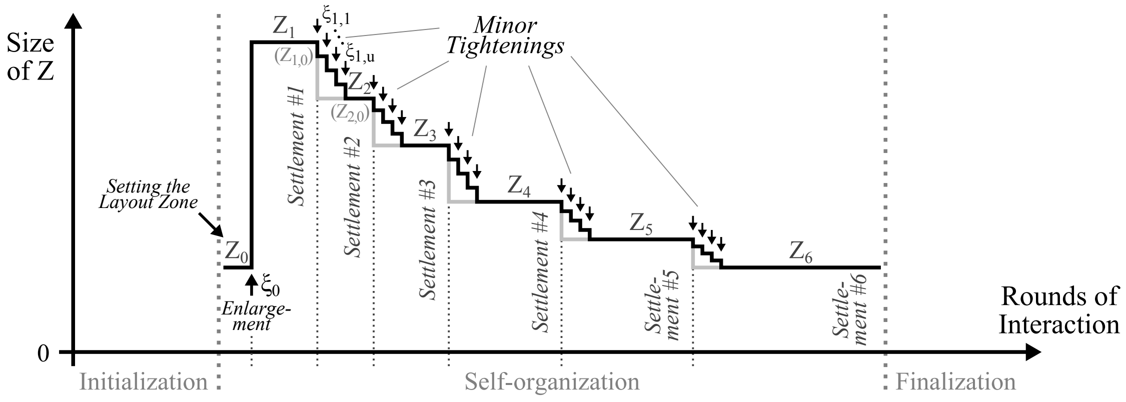


Figure 7.61: Exemplary tightening profile of a linear progressive tightening policy.

The first minor contraction amount $\xi_{i,1}$ can be calculated by setting the areic difference $[Z_{i+1}]-[Z_i]$ based on $\xi_{i,1}$ equal to the desired fraction of the areic difference based on ξ_i:

$$w(1-2\xi_{i,1})\cdot h(1-2\xi_{i,1})-w\cdot h=\frac{w(1-2\xi_i)\cdot h(1-2\xi_i)-w\cdot h}{u} \tag{7.124}$$

$$wh\cdot\big((1-2\xi_{i,1})^2-1\big)=wh\cdot\big((1-2\xi_i)^2-1\big)\cdot\frac{1}{u} \tag{7.125}$$

$$1-4\xi_{i,1}+4\xi_{i,1}^2-1=(1-4\xi_i+4\xi_i^2-1)\cdot\frac{1}{u} \tag{7.126}$$

$$-\xi_{i,1}+\xi_{i,1}^2=(-\xi_i+\xi_i^2)\cdot\frac{1}{u} \tag{7.127}$$

$$\xi_{i,1}^2-\xi_{i,1}+\frac{1}{u}(\xi_i-\xi_i^2)=0. \tag{7.128}$$

This quadratic equation can be easily solved with the well-known *quadratic formula*, which produces

$$\xi_{i,1}=\frac{1\pm\sqrt{1-\frac{4}{u}(\xi_i-\xi_i^2)}}{2}. \tag{7.129}$$

It should be noted that $\xi_i-\xi_i^2\leq 0.25$ for any ξ_i (and that $u\geq 1$ as already stated), which means that the discriminant of the polynomial in equation 7.129 never becomes negative and the equation is therefore always solvable (without consulting complex numbers). But, concerning the case where the square root is *added* in the dividend of equation 7.129, the minor contraction amount is $\xi_{i,1}\geq 0.5$ for any u and for any ξ_i, and thus lies out of the valid range (defined in Section 7.4.1.1). Hence, the minor contraction amount is unequivocally given only by the case where the square root is *subtracted* in the dividend:

$$\xi_{i,1}=\frac{1-\sqrt{1-\frac{4}{u}(\xi_i-\xi_i^2)}}{2}. \tag{7.130}$$

To calculate the second minor contraction amount $\xi_{i,2}$ one may contemplate that the total *edge displacement* of the western edge[18] is destined to be $\epsilon_i=\xi_i\cdot w_{i,0}$ where $w_{i,0}$ denotes the width w of the layout zone's bounding box before the first minor tightening. After the first minor tightening, the edge displacement obtained so far is $\epsilon_{i,1}=\xi_{i,1}\cdot w_{i,0}$, which means that the remaining edge displacement is $\epsilon_i-\epsilon_{i,1}=(\xi_i-\xi_{i,1})\cdot w_{i,0}$. If this remaining edge displacement was to be achieved with one single tightening, the *residual contraction amount* $\xi_{i,[2,u]}$ could be determined via the following equation, where $w_{i,1}$ with $w_{i,1}=w_{i,0}\cdot(1-2\xi_{i,1})$ denotes the width of the layout zone's bounding box after the first minor tightening:

$$\xi_{i,[2,u]}\cdot w_{i,1}=(\xi_i-\xi_{i,1})\cdot w_{i,0} \tag{7.131}$$

$$\xi_{i,[2,u]}=(\xi_i-\xi_{i,1})\cdot\frac{w_{i,0}}{w_{i,1}} \tag{7.132}$$

$$\xi_{i,[2,u]}=(\xi_i-\xi_{i,1})\cdot\frac{w_{i,0}}{w_{i,0}\cdot(1-2\xi_{i,1})} \tag{7.133}$$

$$\xi_{i,[2,u]}=\frac{\xi_i-\xi_{i,1}}{1-2\xi_{i,1}}. \tag{7.134}$$

With this value, equation 7.130 can be used to calculate the second minor contraction amount $\xi_{i,2}$, wherein the number of minor tightenings now must be $u-1$ since one minor tightening has already been performed. On that account, equation 7.130 becomes

$$\xi_{i,2}=\frac{1-\sqrt{1-\frac{4}{u-1}(\xi_{i,[2,u]}-\xi_{i,[2,u]}^2)}}{2}. \tag{7.135}$$

[18]Taking the western edge here is arbitrary. Any other edge (eastern, southern, northern) would ultimately lead to the same result in equation 7.134.

Based on this proceeding, equation 7.130 can be written for $j = 1 \ldots u$ in general form as

$$\xi_{i,j} = \frac{1 - \sqrt{1 - \frac{4}{u+1-j}(\xi_{i,[j,u]} - \xi^2_{i,[j,u]})}}{2} \tag{7.136}$$

to calculate the minor contraction amounts from $\xi_{i,1}$ to $\xi_{i,u}$. To provide the residual contraction amounts $\xi_{i,[j,u]}$ that appear on the right-hand side of equation 7.136, equation 7.134 can be generalized into

$$\xi_{i,[j,u]} = \frac{\xi_i}{\prod_{k=1}^{j-1}(1 - 2\xi_{i,k})} - \sum_{k=1}^{j-1} \frac{\xi_{i,k}}{\prod_{n=k}^{j-1}(1 - 2\xi_{i,n})}. \tag{7.137}$$

As a plausibility check, evaluating equation 7.136 for $j = u$ unfurls into

$$\begin{aligned}
\xi_{i,u} &= \frac{1 - \sqrt{1 - \frac{4}{u+1-u}(\xi_{i,[u,u]} - \xi^2_{i,[u,u]})}}{2} && (7.138)\\
&= \frac{1 - \sqrt{1 - 4\xi_{i,[u,u]} + 4\xi^2_{i,[u,u]}}}{2} && (7.139)\\
&= \frac{1 - \sqrt{(1 - 2\xi_{i,[u,u]})^2}}{2} && (7.140)\\
&= \frac{1 - 1 + 2\xi_{i,[u,u]}}{2} && (7.141)\\
&= \xi_{i,[u,u]} && (7.142)
\end{aligned}$$

which means, that the last minor contraction amount $\xi_{i,u}$ is equal to the last residual contraction amount $\xi_{i,[u,u]}$ (as expected). Furthermore, a spot sample can be evaluated from equation 7.137 for $j = 1$: comprehending that the *empty product* is by convention equal to the multiplicative identity 1 just as the *empty sum* is by convention equal to the additive identy 0, the spot sample folds up into

$$\begin{aligned}
\xi_{i,[1,u]} &= \frac{\xi_i}{\prod_{k=1}^{1-1}(1 - 2\xi_{i,k})} - \sum_{k=1}^{1-1} \frac{\xi_{i,k}}{\prod_{n=k}^{1-1}(1 - 2\xi_{i,n})} && (7.143)\\
&= \frac{\xi_i}{1} - 0 = \xi_i && (7.144)
\end{aligned}$$

which explains the appearance of ξ_i instead of $\xi_{i,[j,u]}$ in equation 7.130.

An even sleeker tightening characteristic than the one of the linear progressive tightening policy above can be obtained with an *exponential progressive tightening policy*. Such a tightening policy is defined through (ϖ, q) where q in the open interval $(0, 1)$ represents the so-called *contraction quotient*. With this tightening policy, every contraction amount ξ_i that is calculated from the pressing rate ϖ, is spread over multiple minor contraction amounts all of which reduce the area of the layout zone by the same factor. One by one, these minor contraction amounts are then applied to the layout zone after each round of interaction until all participants have settled. For a contraction quotient of $q = 0.5$ the tightening profile of this tightening policy is exemplarily depicted in Figure 7.62.

The first minor contraction amount $\xi_{i,1}$ can be calculated from equation 7.130 by setting $u = \frac{1}{q}$, which leads to

$$\xi_{i,1} = \frac{1 - \sqrt{1 - 4q \cdot (\xi_i - \xi_i^2)}}{2}. \tag{7.145}$$

After the first minor tightening, the second minor contraction amount $\xi_{i,2}$ likewise is

$$\xi_{i,2} = \frac{1 - \sqrt{1 - 4q \cdot (\xi_{i,[2,u]} - \xi^2_{i,[2,u]})}}{2}. \tag{7.146}$$

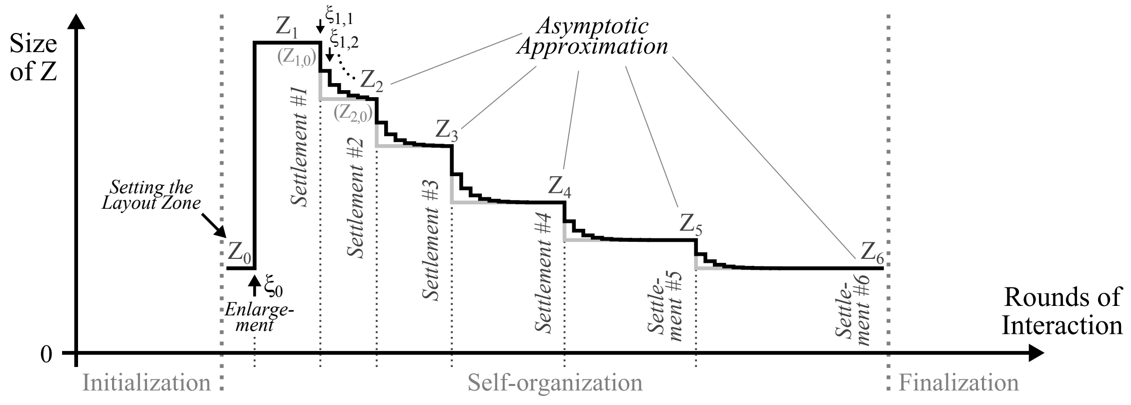


Figure 7.62: Exemplary tightening profile of an exponential progressive tightening policy.

So, it becomes obvious that the generalized formula for calculating all minor contraction amounts is

$$\xi_{i,j} = \frac{1 - \sqrt{1 - 4q \cdot (\xi_{i,[j,u]} - \xi^2_{i,[j,u]})}}{2} \tag{7.147}$$

for which the residual contraction amounts $\xi_{i,[j,u]}$ can again be determined via equation 7.137 as before. Alternatively, equation 7.137 can also be turned into a recursive formula as follows:

$$\begin{aligned}
\xi_{i,[j,u]} &= \frac{\xi_i}{\prod_{k=1}^{j-1}(1-2\xi_{i,k})} - \sum_{k=1}^{j-1} \frac{\xi_{i,k}}{\prod_{n=k}^{j-1}(1-2\xi_{i,n})} && (7.148)\\
&= \frac{\xi_i}{(1-2\xi_{i,j-1}) \cdot \prod_{k=1}^{j-2}(1-2\xi_{i,k})} - \sum_{k=1}^{j-2} \frac{\xi_{i,k}}{(1-2\xi_{i,j-1}) \cdot \prod_{n=k}^{j-2}(1-2\xi_{i,n})} - \frac{\xi_{i,j-1}}{1-2\xi_{i,j-1}} && (7.149)\\
&= \frac{1}{1-2\xi_{i,j-1}} \cdot \left(\underbrace{\frac{\xi_i}{\prod_{k=1}^{j-2}(1-2\xi_{i,k})} - \sum_{k=1}^{j-2} \frac{\xi_{i,k}}{\prod_{n=k}^{j-2}(1-2\xi_{i,n})}}_{=\xi_{i,[j-1,u]}} - \xi_{i,j-1} \right) && (7.150)\\
&= \frac{\xi_{i,[j-1,u]} - \xi_{i,j-1}}{1-2\xi_{i,j-1}} && (7.151)
\end{aligned}$$

which is applicable for $j > 1$. With this case distinction, the formula for all $j = 1 \dots u$ becomes

$$\xi_{i,[j,u]} = \begin{cases} \xi_i & \Leftrightarrow \quad j = 1 \\ \frac{\xi_{i,[j-1,u]} - \xi_{i,j-1}}{1-2\xi_{i,j-1}} & \Leftrightarrow \quad j > 1. \end{cases} \tag{7.152}$$

With an exponential progressive tightening policy, the layout zone gets arbitrarily close to its designated size during each tightening-settlement cycle. This can be proven as follows: for the j^{th} minor tightening, the change in the size of the layout zone is

$$\lceil Z_{i,j} \rfloor - \lceil Z_{i,j-1} \rfloor = q \cdot (\lceil Z_{i+1} \rfloor - \lceil Z_{i,j-1} \rfloor). \tag{7.153}$$

For $j - 1$ it can thus equally be said that

$$\begin{aligned}
\lceil Z_{i,j-1} \rfloor - \lceil Z_{i,j-2} \rfloor &= q \cdot (\lceil Z_{i+1} \rfloor - \lceil Z_{i,j-2} \rfloor) && (7.154)\\
\lceil Z_{i,j-1} \rfloor &= q \cdot (\lceil Z_{i+1} \rfloor - \lceil Z_{i,j-2} \rfloor) + \lceil Z_{i,j-2} \rfloor && (7.155)
\end{aligned}$$

which may be used to substitute $[Z_{i,j-1}]$ on the right-hand side of equation 7.153 to produce

$$\begin{aligned}
[Z_{i,j}] - [Z_{i,j-1}] &= q \cdot ([Z_{i+1}] - [Z_{i,j-1}]) && (7.156)\\
&= q \cdot ([Z_{i+1}] - (q \cdot ([Z_{i+1}] - [Z_{i,j-2}]) + [Z_{i,j-2}])) && (7.157)\\
&= q \cdot ([Z_{i+1}] - [Z_{i,j-2}] - q \cdot ([Z_{i+1}] - [Z_{i,j-2}])) && (7.158)\\
&= q \cdot (1-q) \cdot ([Z_{i+1}] - [Z_{i,j-2}]) && (7.159)\\
&= q \cdot (1-q)^2 \cdot ([Z_{i+1}] - [Z_{i,j-3}]) && (7.160)\\
&\vdots && (7.161)\\
&= q \cdot (1-q)^{j-1} \cdot ([Z_{i+1}] - [Z_{i,0}]). && (7.162)
\end{aligned}$$

Now, the layout zone's total change in size after j minor tightenings can be expressed as the sum

$$[Z_{i,j}] - [Z_{i,0}] = [Z_{i,j}] \underbrace{-[Z_{i,j-1}] + [Z_{i,j-1}]}_{=0} \underbrace{-[Z_{i,j-2}] + [Z_{i,j-2}]}_{=0} - \ldots \underbrace{-[Z_{i,1}] + [Z_{i,1}]}_{=0} - [Z_{i,0}] \qquad (7.163)$$

$$\begin{aligned}
&= \sum_{k=1}^{j} [Z_{i,k}] - [Z_{i,k-1}] && (7.164)\\
&= \sum_{k=1}^{j} q \cdot (1-q)^{k-1} \cdot ([Z_{i+1}] - [Z_{i,0}]) && (7.165)\\
&= q \cdot ([Z_{i+1}] - [Z_{i,0}]) \cdot \sum_{k=0}^{j-1} (1-q)^k && (7.166)
\end{aligned}$$

and the value $[Z_{i,\infty}]$ that the zone size tends to, can be calculated through the mathematical limit

$$[Z_{i,\infty}] = [Z_{i,0}] + q \cdot ([Z_{i+1}] - [Z_{i,0}]) \cdot \lim_{j\to\infty} \sum_{k=0}^{j-1} (1-q)^k. \qquad (7.167)$$

In mathematics, the term q^k for $k \geq 0$ is denoted as a *geometric sequence*, while the sum $\sum \mathrm{q}^k$ represents its *geometric series*. The mathematical limit of such a geometric series is known to be

$$\sum_{k=0}^{\infty} = \frac{1}{1-\mathrm{q}} \qquad (7.168)$$

for $|\mathrm{q}| < 1$. Since the contraction quotient q must be $0 < q < 1$ by definition (as already stated), it is sure that $|1-q| < 1$ which allows to resolve equation 7.167 into

$$\begin{aligned}
[Z_{i,\infty}] &= [Z_{i,0}] + q \cdot ([Z_{i+1}] - [Z_{i,0}]) \cdot \lim_{j\to\infty} \sum_{k=0}^{j-1} (1-q)^k && (7.169)\\
&= [Z_{i,0}] + q \cdot ([Z_{i+1}] - [Z_{i,0}]) \cdot \Big(\underbrace{\lim_{j\to\infty} \sum_{k=0}^{j} (1-q)^k}_{=\frac{1}{1-(1-q)} = \frac{1}{q}} - \underbrace{\lim_{j\to\infty} (1-q)^j}_{=0} \Big) && (7.170)\\
&= [Z_{i,0}] + q \cdot ([Z_{i+1}] - [Z_{i,0}]) \cdot \frac{1}{q} && (7.171)\\
&= [Z_{i+1}]. && (7.172)
\end{aligned}$$

Thus (in contrast to a geometric series, where the limit depends on the base of the exponentiation), the q and $\frac{1}{q}$ cancel each other out in equation 7.171 such that the layout zone asymptotically approaches its designated size $[Z_{i+1}]$ independent of the contraction quotient. So, compared to a linear progressive tightening policy, an exponential progressive tightening policy has the advantage that it is more rigorous in the beginning of a settlement and then lets the layout zone snuggle down towards the desired dimensions quite smoothly, not abruptly.

With an exponential progressive tightening policy, one might come to believe that the participants will never be able to conclude a settlement because the layout zone gets tightened unceasingly. However, this is prevented by the *minimal movement distance* (see Section 7.3.3.1): eventually, there is a minor tightening in each tightening-settlement cycle whose contraction amount is so small that no participant performs an action because the length of the action's movement vector is smaller than the minimal movement distance.

7.4.2.2 Regressive Tightening

A *regressive tightening policy* is a transient tightening policy where a major tightening is indeed performed after every settlement, but is then followed by several *minor enlargements*. In this way, every tightening is at first as rigorous as with an abrasive tightening policy – however, the reins are loosened afterwards, which can be helpful if the participants have trouble finding a viable constellation. Although various kinds of regressive tightening policies can be conceived in theory (such as a linear or an exponential one, in the style of those discussed in Section 7.4.2.1), three particular traits should be envisaged here per tightening-settlement cycle: (1) immediately after the major tightening, the minor enlargements should be rather slight so as not to depart from the designated zone size too quickly, (2) then, the minor enlargements should become more and more lavish to oblige the rivaling participants, and (3) towards the end of the tightening-settlement cycle, the layout zone should be prevented from ever attaining the size that it had before the major tightening – otherwise, the flow of self-organization would not make any headway towards the final layout zone size.

A transient tightening policy that exhibits these three traits can be deduced from a *logistic function* and is therefore denoted as a *logistic regressive tightening policy*. A logistic function has a characteristic sigmoid curve, as illustrated in Figure 7.63 (a). The tightening policy is specified by (ϖ, z, z^*), where z and z^* are *zone size quotients*: z defines the maximal minor enlargement that is to occur during a tightening-settlement cycle, and z^* specifies the first minor enlargement after each major tightening. Both z and z^* relate the desired zone size change to the total difference in zone size that results from the major tightening. In mathematics, a logistic function is in general defined by the formula

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{7.173}$$

and can now be carried over to SWARM in order to describe the desired layout zone size $\lceil Z_{i,j} \rfloor$ for all $j > 1$ (i.e., the zone size after every minor enlargement). As will now be explained in detail, carrying over the formula involves three steps, displayed in Figure 7.63 (b)–(d).

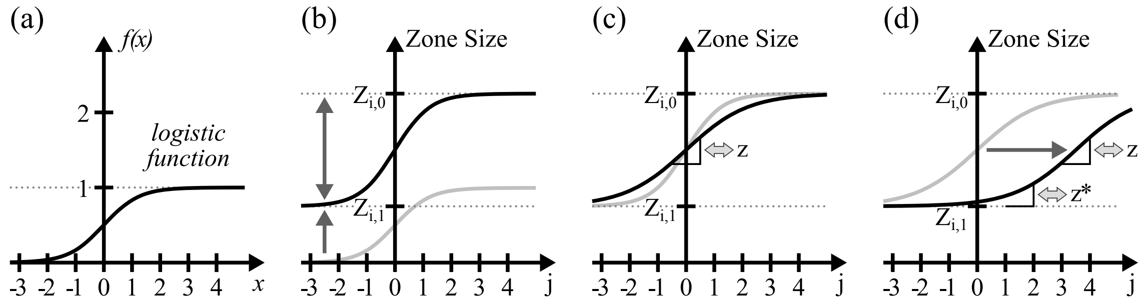


Figure 7.63: Utilizing a logistic function's sigmoid curve for a logistic regressive tightening policy.

First, the sigmoid curve has to be stretched and shifted in vertical direction as shown in Figure 7.63 (b), such that the layout zone size regresses from $\lceil Z_{i,1} \rfloor$ (zone size after the major tightening) towards $\lceil Z_{i,0} \rfloor$ (zone size before the major tightening). Thus, the formula becomes

$$\lceil Z_{i,j} \rfloor = \frac{\lceil Z_{i,0} \rfloor - \lceil Z_{i,1} \rfloor}{1 + \mathrm{e}^{-j}} + \lceil Z_{i,1} \rfloor \tag{7.174}$$

where the dividend $[Z_{i,0}] - [Z_{i,1}]$ stretches the curve and the addend $[Z_{i,1}]$ shifts the curve upwards.

Second, referring to Figure 7.63 (c), the sigmoid curve must be stretched horizontally to meet the given zone size quotient z. For that purpose, the formula is rewritten with a different base β as

$$[Z_{i,j}] = \frac{[Z_{i,0}] - [Z_{i,1}]}{1 + \beta^{-j}} + [Z_{i,1}] \tag{7.175}$$

where β can now be deduced from z to obtain the desired steepness. The derivate of the logistic function $f(x)$ above is largest at $x = 0$ (the abscissa of the function's inflection point). Hence, the greatest vertical difference Δy between two points on $f(x)$ that are horizontally apart by $\Delta x = 1$ is found between $x = -\frac{1}{2}$ and $x = \frac{1}{2}$ and amounts to

$$\Delta y = f\left(\tfrac{1}{2}\right) - f\left(-\tfrac{1}{2}\right). \tag{7.176}$$

This equation can be expressed in a different way because $f(x)$ possesses point symmetry around the point $(0, f(0))$. The proof is given by checking if $f(x) - f(0) = -f(-x) + f(0)$:

$$\frac{1}{1+\mathrm{e}^{-x}} - \frac{1}{1+\mathrm{e}^{0}} = -\frac{1}{1+\mathrm{e}^{x}} + \frac{1}{1+\mathrm{e}^{0}} \tag{7.177}$$

$$\frac{1}{1+\frac{1}{\mathrm{e}^{x}}} - \frac{1}{2} = -\frac{1}{1+\mathrm{e}^{x}} + \frac{1}{2} \tag{7.178}$$

$$\frac{\mathrm{e}^{x}}{\mathrm{e}^{x}+1} - \frac{1}{2} = -\frac{1}{1+\mathrm{e}^{x}} + \frac{1}{2} \tag{7.179}$$

$$\frac{\mathrm{e}^{x}}{\mathrm{e}^{x}+1} + \frac{1}{1+\mathrm{e}^{x}} = \frac{1}{2} + \frac{1}{2} \tag{7.180}$$

$$\frac{\mathrm{e}^{x}+1}{\mathrm{e}^{x}+1} = 1. \quad \square \tag{7.181}$$

Now, due to this point symmetry, the function slope expressed in equation 7.176 can be rewritten as

$$\Delta y = 2 \cdot \left(f\left(\tfrac{1}{2}\right) - f(0)\right) \tag{7.182}$$

which represents the maximal minor enlargement, and here also constitutes the zone size quotient z because the height of the $f(x)$ curve is 1. Substituting base e with a variable β allows to attain a specific z, for which β can be deduced as follows:

$$\Delta y = 2 \cdot \left(f\left(\tfrac{1}{2}\right) - f(0)\right) \tag{7.183}$$

$$z = 2 \cdot \left(\frac{1}{1+\beta^{-\frac{1}{2}}} - \frac{1}{2}\right) \tag{7.184}$$

$$z + 1 = \frac{2}{1+\beta^{-\frac{1}{2}}} \tag{7.185}$$

$$1+\beta^{-\frac{1}{2}} = \frac{2}{z+1} \tag{7.186}$$

$$\beta = \left(\frac{2}{z+1} - 1\right)^{-2} \tag{7.187}$$

$$\beta = \left(\frac{1+z}{1-z}\right)^{2}. \tag{7.188}$$

Inserting this term in equation 7.175 to replace the base variable β leads to the new formula

$$[Z_{i,j}] = \frac{[Z_{i,0}] - [Z_{i,1}]}{1 + \left(\frac{1+z}{1-z}\right)^{-2j}} + [Z_{i,1}]. \tag{7.189}$$

According to Figure 7.63 (d), the third step is to shift the sigmoid curve in horizontal direction such that the first minor enlargement after the major tightening is achieved as was specified through z^*. A look at Figure 7.63 (c) suggests that the curve must be shifted rightwards by the distance between the vertical axis and the abscissa at which $f(x)$ veers away from $[Z_{i,1}]$ by the specified zone size difference. This can be formally expressed as follows:

$$\frac{[Z_{i,0}]-[Z_{i,1}]}{1+\left(\frac{1+z}{1-z}\right)^{-2j}}+[Z_{i,1}]-[Z_{i,1}]=z^{*}\cdot([Z_{i,0}]-[Z_{i,1}]) \tag{7.190}$$

$$\frac{[Z_{i,0}]-[Z_{i,1}]}{z^{*}\cdot([Z_{i,0}]-[Z_{i,1}])}=1+\left(\frac{1+z}{1-z}\right)^{-2j} \tag{7.191}$$

$$\frac{1}{z^{*}}-1=\left(\frac{1+z}{1-z}\right)^{-2j} \tag{7.192}$$

$$-2j=\log_{\frac{1+z}{1-z}}\left(\frac{1}{z^{*}}-1\right). \tag{7.193}$$

As is the case with the contraction quotient q in an exponential progressive tightening policy, both z and z^{*} lie in the open interval $(0,1)$. This means, that computing the logarithm is indeed allowed here because $\frac{1+z}{1-z}>1$ and $\frac{1}{z^{*}}-1>0$ for all values that z and z^{*} can assume. So, equation 7.193 can now be solved with the aid of the logarithmic laws, leading to

$$-2j=\frac{\ln\left(\frac{1}{z^{*}}-1\right)}{\ln\left(\frac{1+z}{1-z}\right)} \tag{7.194}$$

$$j=\frac{\ln\left(\frac{1}{z^{*}}-1\right)}{-2\cdot\ln\left(\frac{1+z}{1-z}\right)}. \tag{7.195}$$

This value for j delivers the desired horizontal shift of the sigmoid curve.[19] But it should be noted, that in effect the curve must be shifted even further since the specified zone size difference is to be effectuated by the first minor enlargement, which in turn occurs after the initial major tightening. To be consistent with the indexing of the kickoff enlargement (where the index i is 0) and the major contraction amounts (with $i\geq 1$), the additional horizontal shift must be 2. Hence, equation 7.189 finally becomes

$$[Z_{i,j}]=\frac{[Z_{i,0}]-[Z_{i,1}]}{1+\left(\frac{1+z}{1-z}\right)^{-2\left(j+\frac{\ln\left(\frac{1}{z^{*}}-1\right)}{-2\cdot\ln\left(\frac{1+z}{1-z}\right)}-2\right)}}+[Z_{i,1}]. \tag{7.196}$$

which –by expanding the exponent in the divisor– can be simplified into

$$[Z_{i,j}]=\frac{[Z_{i,0}]-[Z_{i,1}]}{1+\left(\frac{1+z}{1-z}\right)^{-2j\,+\,\ln\left(\frac{1}{z^{*}}-1\right)/\ln\left(\frac{1+z}{1-z}\right)\,+\,4}}+[Z_{i,1}]. \tag{7.197}$$

For example, let there be a logistic regressive tightening policy with $z=\frac{1}{4}$ and $z^{*}=\frac{1}{40}$, wherein a major tightening occurs from $[Z_{i,0}]=23$ to $[Z_{i,1}]=19$. Table 7.12 shows the zone sizes that are (based on equation 7.197) calculated for $1<j\leq 10$ and the corresponding curve is depicted in Figure 7.64. As can be seen, the first minor enlargement increases the size of the layout zone by 0.1 (which is precisely $\frac{1}{40}$ of $[Z_{i,0}]-[Z_{i,1}]=4$). The greatest minor enlargement in this example occurs between $j=5$ and $j=6$ and amounts to 0.9982 (which is near to $\frac{1}{4}$ of $[Z_{i,0}]-[Z_{i,1}]$). The reason why this minor enlargement does not exactly match the given zone size quotient z here, is that the greatest minor enlargement may – depending on $[Z_{i,0}]$, $[Z_{i,1}]$, and z^{*}– also occur between non-integral values for j (in this example between $j=5.0859$ and $j=6.0859$, where the zone size changes from 20.5 to 21.5 understandably).

Now that the desired zone sizes can be expressed via equation 7.197, the respective contraction amounts are to be calculated. In line with the determination in Section 7.4.1.2 that a major contraction

[19] A plausibility check is to set $z^{*}=0.5$ which correctly results in $j=0$ as expected.

Table 7.12: Calculation values of a tightening example with a logistic regressive tightening policy.

Round j	**Zone Size** $[Z_{i,j}]$	**Absolute Change** $[Z_{i,j}] - [Z_{i,j-1}]$	**Contraction Amount** $\xi_{i,j}$
0	23.0000	—	0.0456
1	19.0000	-4.0000	-0.0013
2	19.1000	0.1000	-0.0022
3	19.2660	0.1660	-0.0051
4	19.6607	0.3947	-0.0095
5	20.4186	0.7580	-0.0121
6	21.4168	0.9982	-0.0095
7	22.2367	0.8199	-0.0050
8	22.6870	0.4503	-0.0021
9	22.8814	0.1944	-0.0008
10	22.9565	0.0751	-0.0003

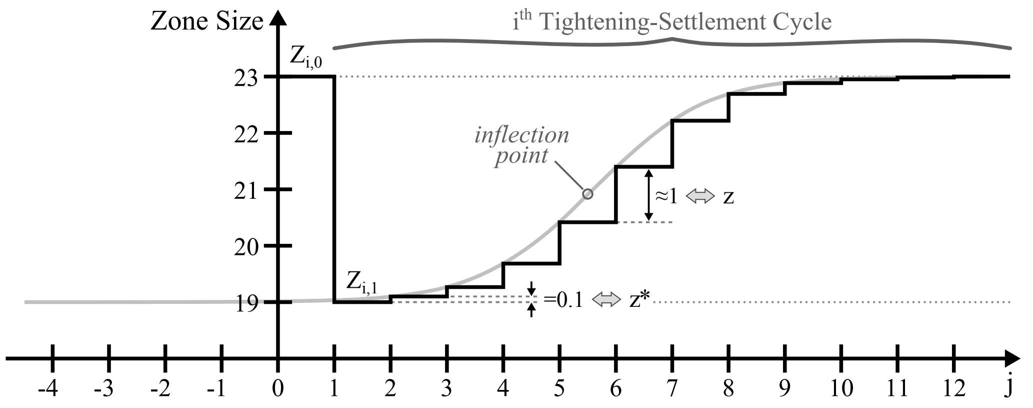


Figure 7.64: Zone size curve of the logistic regressive tightening example from Table 7.12.

amount ξ_i tightens the layout zone from Z_i into Z_{i+1}, a minor contraction amount $\xi_{i,j}$ here changes the layout zone from $Z_{i,j}$ into $Z_{i,j+1}$. Following the previous ruminations, such a contraction amount $\xi_{i,j}$ scales both the width and the height of the layout zone by a factor of $(1 - 2\xi_{i,j})$ respectively. The combined change of width and height must of course be equal to the relative change in zone size. Formally articulated, this allows to calculate the sought contraction amount as follows:

$$(1 - 2\xi_{i,j})^2 = \frac{[Z_{i,j+1}]}{[Z_{i,j}]} \tag{7.198}$$

$$1 - 2\xi_{i,j} = \sqrt{\frac{[Z_{i,j+1}]}{[Z_{i,j}]}} \tag{7.199}$$

$$2\xi_{i,j} = 1 - \sqrt{\frac{[Z_{i,j+1}]}{[Z_{i,j}]}} \tag{7.200}$$

$$\xi_{i,j} = \frac{1}{2} - \frac{1}{2}\sqrt{\frac{[Z_{i,j+1}]}{[Z_{i,j}]}}. \tag{7.201}$$

Regarding the given example, the contraction amounts calculated with this formula are also listed in Table 7.12. Naturally, $\xi_{i,0}$ is positive because the contraction is indeed a tightening, whereas the other contraction amounts are negative since they correspond to the minor enlargements of the layout zone.

As it is the case in Figure 7.64, z^* is supposed to be much smaller than z. This statement calls for attention to one further subtlety: if z^* is too large, then the tightening profile can look awkward because the change from $Z_{i,1}$ to $Z_{i,2}$ is greater than the change from $Z_{i,2}$ to $Z_{i,3}$. Such a situation, referred to as *hyperregression*, is depicted in Figure 7.65 (a). On the other hand, it may be possible to find a certain value z^*_{iso} for which there is a situation of *isoregression*, where the zone size differences in these two minor enlargements are equal, as in image (b). If z^* is smaller than that value, then the first minor enlargement is lesser than the second one, which is denoted as *hyporegression* (c).

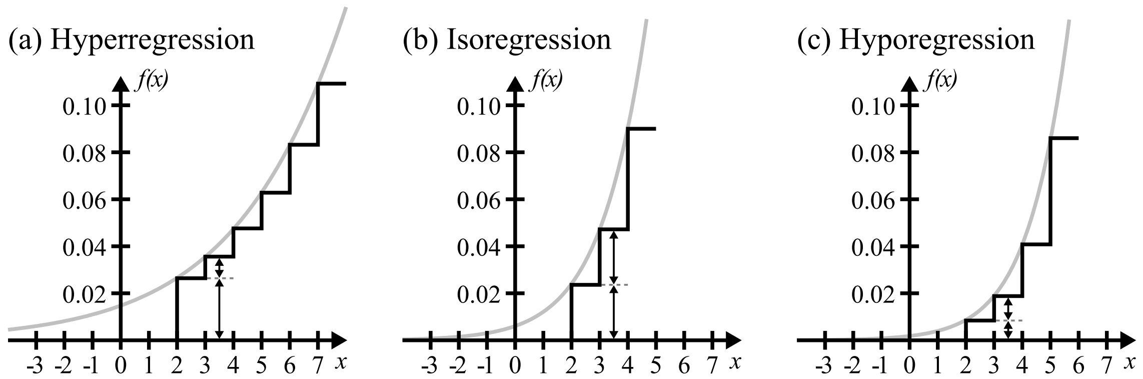


Figure 7.65: Depending on the first minor enlargement, three types of regression can be discerned.

In principle, Figure 7.65 makes it obvious that finding z^*_{iso} is possible by taking equation 7.173 and determining the abscissa x at which the difference between $f(x+1)$ and $f(x)$ is equal to $f(x)$. Again, the base may be a variable β, which formally produces:

$$\frac{1}{1+\beta^{-(x+1)}} - \frac{1}{1+\beta^{-x}} = \frac{1}{1+\beta^{-x}} \tag{7.202}$$

$$\frac{1}{1+\beta^{-x-1}} - \frac{2}{1+\beta^{-x}} = 0 \tag{7.203}$$

$$1+\beta^{-x} - 2\cdot\left(1+\beta^{-x-1}\right) = 0 \tag{7.204}$$

$$1+\beta^{-x} - 2 - 2\cdot\beta^{-x}\cdot\beta^{-1} = 0 \tag{7.205}$$

$$\beta^{-x}\cdot\left(1-\frac{2}{\beta}\right) = 1 \tag{7.206}$$

$$\beta^{-x} = \frac{1}{\frac{\beta-2}{\beta}} \tag{7.207}$$

$$x = -\log_\beta\left(\frac{\beta}{\beta-2}\right). \tag{7.208}$$

Having β as the base is fine for the logarithm because $\beta = \left(\frac{1+z}{1-z}\right)^2$ and $\frac{1+z}{1-z} > 1$ (as already mentioned before), which means that $\beta > 1$. However, the argument $\frac{\beta}{\beta-2}$ is only positive for $\beta > 2$ (and for $\beta < 0$ but that is not relevant here since $\beta > 1$), as visualized by the graph in Figure 7.66. This means, that a value for z^*_{iso} can only be found if $\beta > 2$, which in turn implies that z must be[20]

$$\left(\frac{1+z}{1-z}\right)^2 > 2 \tag{7.209}$$

$$\frac{1+z}{1-z} > \sqrt{2} \tag{7.210}$$

$$1+z > \sqrt{2} - \sqrt{2}z \tag{7.211}$$

$$z\cdot(1+\sqrt{2}) > \sqrt{2}-1 \tag{7.212}$$

$$z > \frac{\sqrt{2}-1}{\sqrt{2}+1} \tag{7.213}$$

$$z > 0.1716 \tag{7.214}$$

[20]The multiplication with $1-z$ in condition 7.210 does not involve an inversion of the inequation because $1-z>0$.

which can also be seen in Figure 7.66. For convenience, equation 7.208 can be simplified into

$$x = -\log_\beta\left(\frac{\beta}{\beta-2}\right) \tag{7.215}$$

$$= -\frac{\ln\left(\frac{\beta}{\beta-2}\right)}{\ln(\beta)} \tag{7.216}$$

$$= -\frac{\ln(\beta) - \ln(\beta-2)}{\ln(\beta)} \tag{7.217}$$

$$= \frac{\ln(\beta-2)}{\ln(\beta)} - 1. \tag{7.218}$$

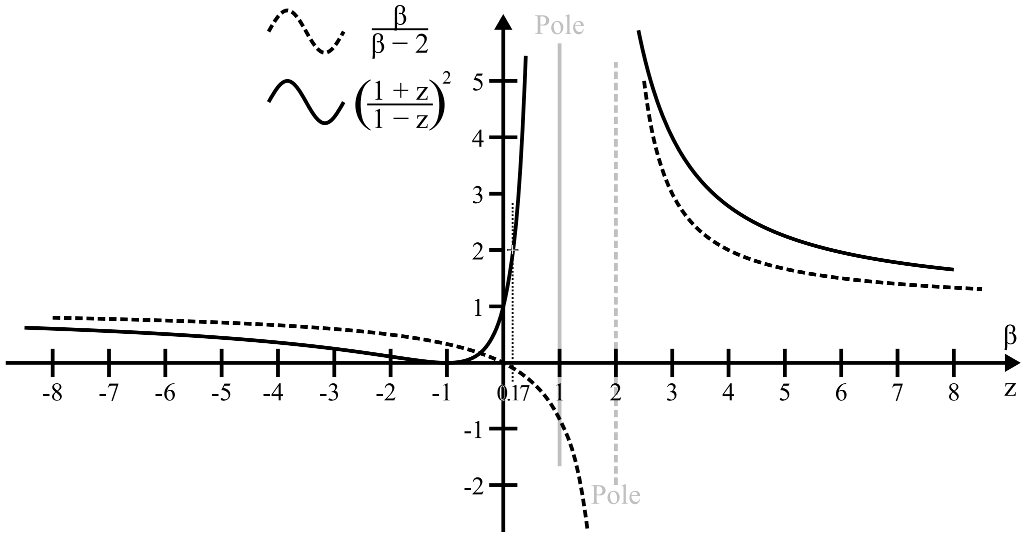


Figure 7.66: Relevant graphs for the calculation of isoregression, showing that β must be greater than 2 (to achieve that $\frac{\beta}{\beta-2} > 0$) and that z must be greater than 0.17 (such that $\beta = (\frac{1+z}{1-z})^2 > 2$).

To illustrate the matter that isoregression requires $\beta > 2$, subtracting the left-hand side of equation 7.202 from its right-hand side produces the function $\Delta_{\text{regression}}(x)$:

$$\Delta_{\text{regression}}(x) = \frac{2}{1+\beta^{-x}} - \frac{1}{1+\beta^{-x-1}}. \tag{7.219}$$

Now, the curve family in Figure 7.67 is obtained by plotting $\Delta_{\text{regression}}(x)$ for various β values. As the image illustrates, the curves for $\beta \leq 2$ do not cross the x-axis, whereas each of the curves for $\beta > 2$ has exactly one zero point, depicted as white circles on the x-axis (for $\beta = 2.78$, $\beta = 4$, and $\beta = 9$). The abscissae of these zero points represent the x-values of isoregression (for the given curves, according to equation 7.218, these are: $x = -1.24$, $x = -0.5$, and $x = -0.11$). To the right of such a zero point, the figure shows that $\Delta_{\text{regression}} > 0$, which signifies hyperregression. To the left of such a zero point, there is hyporegression since the respective curve runs below the x-axis.

The zero points from Figure 7.67 are confirmed by Figure 7.68, where equation 7.218 is plotted in the lower-right quadrant as a function $x(\beta)$. The upper-left quadrant displays the function $x^{-1}(\beta) = \beta(x)$ which represents the inverse function of $x(\beta)$. The white circles from Figure 7.67 can also be found in Figure 7.68 to visualize that the vertical lines through these circles cross the $\beta(x)$ function precisely at $\beta = 2.78$, $\beta = 4$, and $\beta = 9$ (as expected).

Based on these considerations, the calculation of z^*_{iso} can now be achieved by inserting the term from equation 7.208 into the logistic function of equation 7.173 (again, with e substituted by β). This delivers

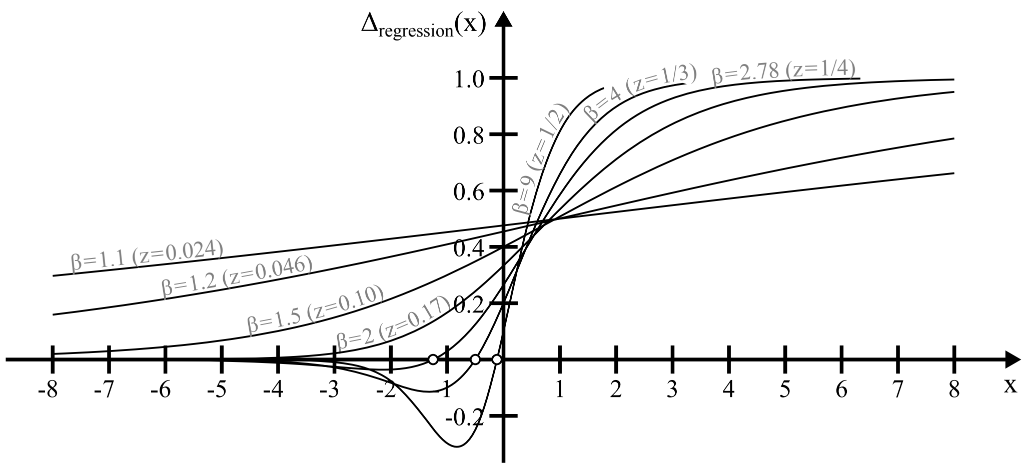


Figure 7.67: Curve family to illustrate that isoregression can only be obtained if β is greater than 2.

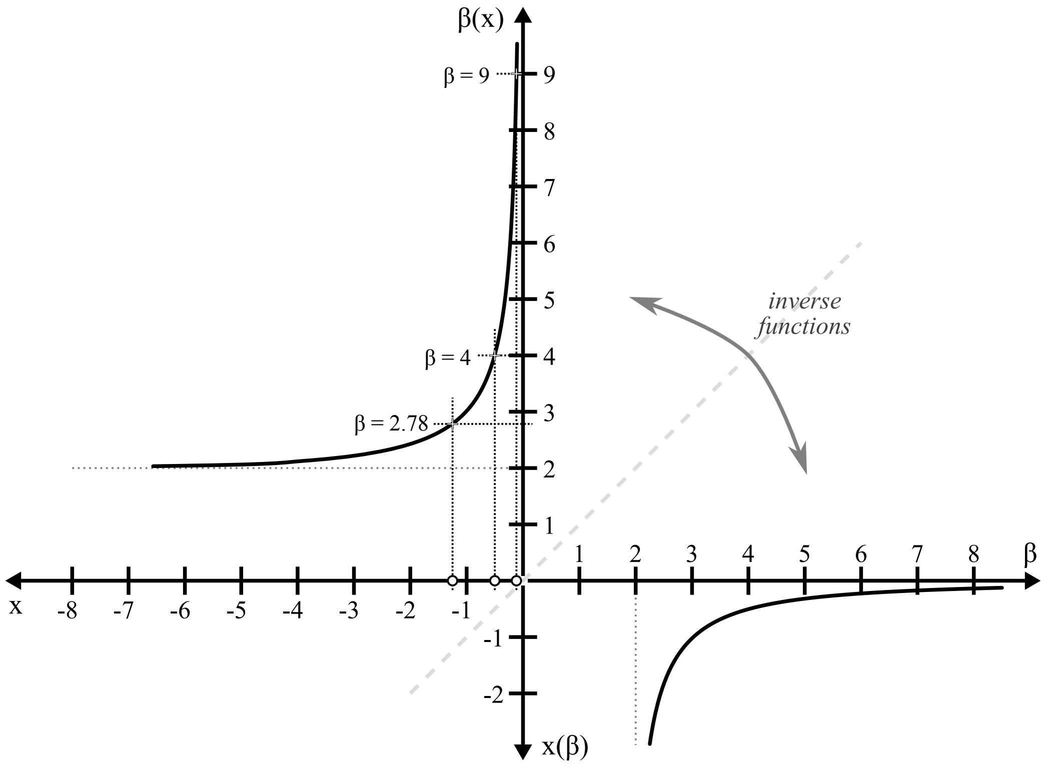


Figure 7.68: Plot of the isoregression equation 7.218 as a function $x(\beta)$ and its inverse function $\beta(x)$.

the abscissa's ordinate, which simultaneously represents the sought value for z^*_{iso}:

$$z^*_{\text{iso}} = \frac{1}{1+\beta^{-\left(-\log_\beta\left(\frac{\beta}{\beta-2}\right)\right)}} \tag{7.220}$$

$$= \frac{1}{1+\frac{\beta}{\beta-2}} \tag{7.221}$$

$$= \frac{\beta-2}{2\beta-2}. \tag{7.222}$$

Figure 7.69 illustrates z^*_{iso} as a function of β. Since $\beta > 1$ by definition, legal values for z^*_{iso} (with $0 < z^*_{\text{iso}} < 1$) are only obtained for $\beta > 2$ (as already discussed before). With respect to z, the isoregression value z^*_{iso} is given by the formula

$$z^*_{\text{iso}} = \frac{\left(\frac{1+z}{1-z}\right)^2 - 2}{2\left(\frac{1+z}{1-z}\right)^2 - 2} \tag{7.223}$$

$$= \frac{\left(\frac{1+2z+z^2}{1-2z+z^2}\right) - 2}{2\left(\frac{1+2z+z^2}{1-2z+z^2}\right) - 2} \tag{7.224}$$

$$= \frac{\frac{1+2z+z^2-2(1-2z+z^2)}{1-2z+z^2}}{\frac{2(1+2z+z^2)-2(1-2z+z^2)}{1-2z+z^2}} \tag{7.225}$$

$$= \frac{-z^2+6z-1}{8z} \tag{7.226}$$

$$= -\frac{z}{8} + \frac{3}{4} - \frac{1}{8z} \tag{7.227}$$

which is also depicted in Figure 7.69. Here, the legal values for z^*_{iso} are found between the first zero crossing of the curve and $z = 1$. The zero crossings occur at

$$-z^2 + 6z - 1 = 0 \tag{7.228}$$

$$z = \frac{-6 \pm \sqrt{6^2 - 4}}{-2} \tag{7.229}$$

$$z = 3 \mp \sqrt{8} \tag{7.230}$$

and the first of these lies at $3-\sqrt{8} = 0.1716$ (which equals the value from equation 7.214 of course). The fact, that the $z^*_{\text{iso}}(z)$ curve runs below the first median (i.e., the bisecting line of the Cartesian system's upper-right quadrant), indicates that z is always greater than its z^*_{iso} value.

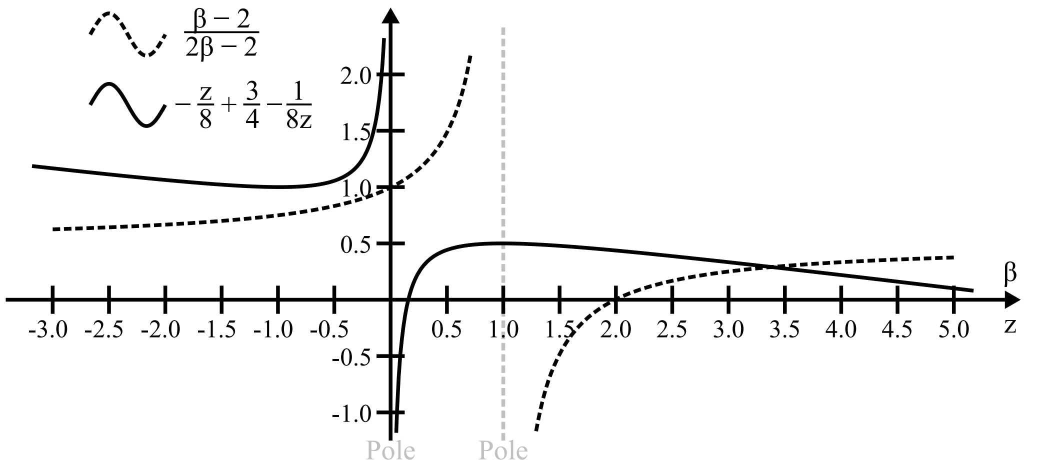


Figure 7.69: The isoregression zone size quotient z^*_{iso} as a function of β and as a function of z.

As an example for calculating z^*_{iso}, let there be a logistic regressive tightening policy with $z = \frac{1}{5}$. According to equation 7.227, isoregression is then effecuated for $z^* = -\frac{1}{5\cdot 8} + \frac{3}{4} - \frac{5}{8} = \frac{-1+30-25}{40} = \frac{1}{10}$. Imagining a major tightening from $[Z_{i,0}] = 20$ to $[Z_{i,1}] = 15$, equation 7.197 produces $[Z_{i,2}] = 15.5$ and $[Z_{i,3}] = 16.0$ (so, since $[Z_{i,2}] - [Z_{i,1}] = [Z_{i,3}] - [Z_{i,2}]$, isoregression has been achieved here).

With the formulas above, it is possible to give evidence of a statement that was made earlier about Figure 7.65 without any proof: if z^* is smaller than z^*_{iso}, then the obtained regression is hyporegression (otherwise hyperregression). This can be attested by looking at equation 7.220: a value for z^* that is smaller (greater) than z^*_{iso} implies that $-\log_\beta\left(\frac{\beta}{\beta-2}\right)$ must become smaller (greater).[21] As known from

[21]For an in-depth analysis, a case distinction can be made:

equation 7.208, this term corresponds to the abscissae in Figure 7.67, for which it is obvious that a smaller (greater) x value –to the left (right) of a zero point– leads to hyporegression (hyperregression), as already pointed out before.

Altogether, the tightening profile of a logistic regressive tightening policy is illustrated in Figure 7.70. This exemplary depiction reflects three characteristic *episodes* of a SWARM run's self-organization phase. In the beginning, when there is still a lot of free space available, the module interaction proceeds comparably quickly because the rivalry is low in this first episode. Later, the ongoing tightening of the layout zone effectuates increasingly competitive situations, such that more and more rounds of interaction are required to achieve a settlement wherein all participants are contented. Typically it is this second episode where the main hurdles of the problem need to be overcome and which decides whether the self-organization will be ultimately successful or not. If the participants manage to get through, then the remaining tightening-settlement cycles involve less and less disruptive actions which affect the layout arrangement only slightly. In this third episode, especially close to the end, it can usually be observed that only *Centering* actions are performed, such that the relative locations of the participants are maintained while their overall constellation is successively driven towards its final size.

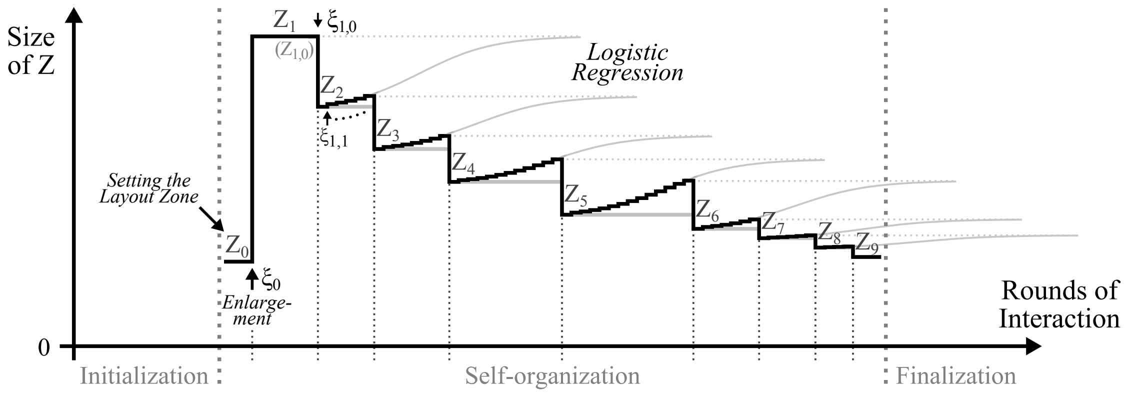


Figure 7.70: Exemplary tightening profile of a logistic regressive tightening policy.

The characteristics of that third episode can be important for transient tightening policies. In the case of a logistic regressive tightening policy, Figure 7.70 indicates that the last tightening must be abrasive, not regressive; otherwise, the tightening policy might be unable to reach the final layout size. Thus, since the interaction control organ does not loosen the layout zone during the last tightening-settlement cycle, it is vital that only marginal actions are performed therein.

7.4.3 Comfort Padding

Another topic which can be discussed within the interaction control organ's scope of activities, is the idea of *comfort padding*. Simply put, comfort padding allows to preserve layout space around a participant during the self-organization. SWARM distinguishes two forms of comfort padding that will be described in Section 7.4.3.1 and Section 7.4.3.2: *solid comfort padding* and *volatile comfort padding*.

7.4.3.1 Solid Comfort Padding

Solid comfort padding preserves a fix amount of layout space around a participant, which means that the amount of layout space that is to be preserved does not change throughout a SWARM run. Figuratively

- If $-\log_\beta\left(\frac{\beta}{\beta-2}\right)$, subsequently referred to as x (from equation 7.208), is negative and becomes smaller (greater), then its absolute value $|x|$ becomes greater (smaller), β^{-x} is $\beta^{|x|}$ and becomes greater (smaller) since $\beta > 1$, and so the whole term of equation 7.220 becomes smaller.
- If $-\log_\beta\left(\frac{\beta}{\beta-2}\right)$, again referred to as x, is positive and becomes smaller (greater), then β^{-x} is $\frac{1}{\beta^{|x|}}$ and becomes greater (smaller) since $\beta^{|x|}$ becomes smaller (greater), and so the whole term of equation 7.220 becomes smaller.

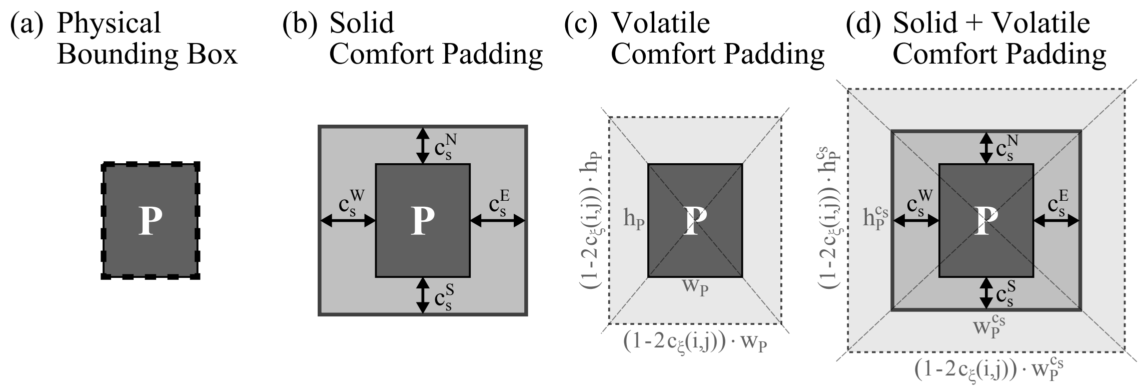

(a) Physical Bounding Box
(b) Solid Comfort Padding
(c) Volatile Comfort Padding
(d) Solid + Volatile Comfort Padding
P
c_s^N
c_s^W
c_s^E
c_s^S
h_P
w_P
$(1-2c_\xi(i,j)) \cdot h_P$
$(1-2c_\xi(i,j)) \cdot w_P$
$h_P^{c_s}$
$w_P^{c_s}$
$(1-2c_\xi(i,j)) \cdot h_P^{c_s}$
$(1-2c_\xi(i,j)) \cdot w_P^{c_s}$

Figure 7.71: Depiction of the different forms of comfort padding around a participant.

speaking, the consideration of comfort padding is achieved by treating the participant as if it were bigger than it actually is. In a formal way, the expression of equation 7.23 (which takes the participant's physically relevant shapes into account) can be said to define only the *physical bounding box* $\square_{\mathrm{p}} P$ of a participant P, as emphasized in Figure 7.71 (a). Then, the consideration of solid comfort padding can be achieved by specifying the bounding box operator $\square P$ (for usage throughout all the formulas in Chapter 7) to be

$$\square P \leftarrow \underset{\text{p+s}}{\square P}. \tag{7.231}$$

If the amount of solid comfort padding is identical on each side of the participant, this *solid bounding box* operator $\underset{\text{p+s}}{\square P}$ can be defined as

$$\underset{\text{p+s}}{\square P} = \circledcirc_{c_{\mathrm{s}}}(\square_{\mathrm{p}} P) \tag{7.232}$$

whereby the physical bounding box of P is simply enlarged in all directions by the same *solid comfort padding amount* c_{s}. If different amounts of solid comfort padding are desired on each side of P (referred to as $c_{\mathrm{s}}^{\mathrm{N}}$, $c_{\mathrm{s}}^{\mathrm{E}}$, $c_{\mathrm{s}}^{\mathrm{S}}$, and $c_{\mathrm{s}}^{\mathrm{W}}$ for the northern, eastern, southern, and western side respectively), the operator can be formally defined in rectangle notation via

$$\underset{\text{p+s}}{\square P} = \Big(\big(\vdash(\square_{\mathrm{p}} P) - c_{\mathrm{s}}^{\mathrm{W}}, \perp(\square_{\mathrm{p}} P) - c_{\mathrm{s}}^{\mathrm{S}} \big), \big(\dashv(\square_{\mathrm{p}} P) + c_{\mathrm{s}}^{\mathrm{E}}, \top(\square_{\mathrm{p}} P) + c_{\mathrm{s}}^{\mathrm{N}} \big) \Big) \tag{7.233}$$

which is illustrated in Figure 7.71 (b). The idea of solid comfort padding is useful when it is desired to preserve a certain amount of layout space between the interacting participants for a subsequent routing step that will then be performed in the finalization phase. This is also about to be demonstrated by the examples of Section 8.3.

7.4.3.2 Volatile Comfort Padding

In contrast to solid comfort padding, the volatile comfort padding of a participant changes throughout the self-organization phase of a SWARM run. The change depends on the size of the layout zone, so this is how the idea of comfort padding is correlated with the activities of the interaction control organ. Like solid comfort padding, volatile comfort padding preserves layout space around a participant during the module interaction – however, this is not done for the sake of the layout space, but with the intention to streamline the flow of self-organization. In fact, the amount of layout space preserved by the volatile comfort padding approaches zero towards the end of the self-organization phase. As above, volatile comfort padding can likewise be incorporated by specifying

$$\square P \leftarrow \underset{\text{p+v}}{\square P} \tag{7.234}$$

for the formulas of Chapter 7. Similar to equation 7.232, the already known contraction operator $\circledast$ can be used to define the *volatile bounding box* operator $\underset{\text{p+v}}{\square P}$ as

$$\underset{\text{p+v}}{\square P} = \circledast_{c_\xi(i,j)}(\square_{\mathrm{p}} P) \tag{7.235}$$

but in contrast to c_s in equation 7.232, the *volatile comfort padding factor* $c_\xi(i,j)$ here is not a constant value but changes with each major zone tightening i (and every minor zone tightening j, if applicable). The $\circledast$ operator satisfies the notion that volatile comfort padding is supposed to keep the aspect ratio of the participant, as it is shown in Figure 7.71 (c). It may also be the case that both solid and volatile comfort padding are requested. For that purpose, the bounding box operator can be specified as

$$\square P \leftarrow \underset{\text{p+s+v}}{\square P} \tag{7.236}$$

with

$$\underset{\text{p+s+v}}{\square P} = \circledast_{c_\xi(i,j)}\left(\underset{\text{p+s}}{\square P}\right) \tag{7.237}$$

which signifies that the volatile comfort padding is clasped (as a kind of "outer" padding) around the participant's solid comfort padding (serving as its "inner" padding). This can also be seen in Figure 7.71 (d), where the combination of solid and volatile comfort padding is depicted. The following ruminations also concentrate on this case of combined (solid plus volatile) comfort padding since equation 7.237 can be considered a generalized form of equation 7.235.[22]

To determine the padding factor $c_\xi(i,j)$ for equation 7.237, the *volatile comfort padding share* c_v with $c_\text{v} \geq 0$ is introduced. It allows to specify how much of the vacant space inside the layout zone $Z_{i,j}$ should be occupied by the volatile comfort padding, totaled over all participants $\mathcal{P}$. In formal terms, this can be expressed as

$$\sum_{P\in\mathcal{P}} \left[\underset{\text{p+s+v}}{\square P}\right] - \sum_{P\in\mathcal{P}} \left[\underset{\text{p+s}}{\square P}\right] = c_\text{v} \cdot \left(\left[Z_{i,j}\right] - \sum_{P\in\mathcal{P}} \left[\underset{\text{p+s}}{\square P}\right]\right). \tag{7.238}$$

According to equation 7.94, the contraction operator $\circledast$ in equation 7.237 increases the size of a participant by a factor of

$$\frac{\left[\underset{\text{p+s+v}}{\square P}\right]}{\left[\underset{\text{p+s}}{\square P}\right]} = \big(1 - 2\cdot c_\xi(i,j)\big)^2 \tag{7.239}$$

and it goes without saying, that this can likewise be said about the sum of the participants' sizes:

$$\frac{\sum_{P\in\mathcal{P}} \left[\underset{\text{p+s+v}}{\square P}\right]}{\sum_{P\in\mathcal{P}} \left[\underset{\text{p+s}}{\square P}\right]} = \big(1 - 2\cdot c_\xi(i,j)\big)^2. \tag{7.240}$$

Now, inserting equation 7.240 into equation 7.238 and replacing $\sum_{P\in\mathcal{P}} \left[\underset{\text{p+s}}{\square P}\right]$ with the more compact notation $\left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]$ produces

$$\big(1 - 2\cdot c_\xi(i,j)\big)^2 \cdot \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right] - \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right] = c_\text{v} \cdot \left(\left[Z_{i,j}\right] - \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]\right) \tag{7.241}$$

$$\left(\big(1 - 2\cdot c_\xi(i,j)\big)^2 - 1\right) \cdot \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right] = c_\text{v} \cdot \left(\left[Z_{i,j}\right] - \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]\right) \tag{7.242}$$

$$\big(-4\cdot c_\xi(i,j) + 4\cdot c_\xi(i,j)^2\big) \cdot \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right] = c_\text{v} \cdot \left(\left[Z_{i,j}\right] - \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]\right) \tag{7.243}$$

$$-4\cdot c_\xi(i,j) + 4\cdot c_\xi(i,j)^2 = \frac{c_\text{v} \cdot \left(\left[Z_{i,j}\right] - \left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]\right)}{\left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]} \tag{7.244}$$

$$4\cdot c_\xi(i,j)^2 - 4\cdot c_\xi(i,j) + c_\text{v} \cdot \left(1 - \frac{\left[Z_{i,j}\right]}{\left[\underset{\text{p+s}}{\boxdot \mathcal{P}}\right]}\right) = 0 \tag{7.245}$$

[22] If only volatile comfort padding (without solid comfort padding) is concerned, the subsequent formulas still hold true, with $\underset{\text{p+s+v}}{\square P}$ effectively being $\underset{\text{p+v}}{\square P}$ and $\underset{\text{p+s}}{\square P}$ being $\underset{\text{p}}{\square P}$.

and this quadratic equation can be solved with the *quadratic formula* to calculate the padding factor as

$$c_\xi(i,j) = \frac{4 - \sqrt{16 - 16 \cdot c_\text{v} \cdot \left(1 - \frac{[Z_{i,j}]}{[\text{⁙}\mathcal{P}]_{\text{p+s}}}\right)}}{8} \tag{7.246}$$

which can be simplified to

$$c_\xi(i,j) = \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{1}{4} \cdot c_\text{v} \cdot \left(1 - \frac{[Z_{i,j}]}{[\text{⁙}\mathcal{P}]_{\text{p+s}}}\right)} \tag{7.247}$$

$$= \frac{1}{2} - \frac{1}{2} \cdot \sqrt{1 - c_\text{v} \cdot \left(1 - \frac{[Z_{i,j}]}{[\text{⁙}\mathcal{P}]_{\text{p+s}}}\right)}. \tag{7.248}$$

As can be seen in equation 7.246, the other solution for $c_\xi(i,j)$ –where the root is added in the dividend, not subtracted– is discarded here, because the padding factor must be $c_\xi(i,j) < 0$ to obtain an enlargement (instead of a contraction). A closer look at equation 7.248 confirms, that –since the area of the layout zone is always greater than the sum of the participants' areas (without volatile comfort padding)– the padding factor is definitely smaller than zero (unless $c_\text{v} = 0$, in which case the padding factor also is zero):

$$c_\xi(i,j) = \underbrace{\frac{1}{2} - \frac{1}{2} \cdot \underbrace{\sqrt{1 - \underbrace{c_\text{v} \cdot \left(1 - \underbrace{\frac{[Z_{i,j}]}{[\text{⁙}\mathcal{P}]_{\text{p+s}}}}_{>1}\right)}_{\leq 0}}}_{\geq 1}}_{\leq 0}. \tag{7.249}$$

Since $c_\xi(i,j)$ depends on $[Z_{i,j}]$, the sum of the participants' areas with their volatile comfort padding changes by every zone tightening. Without volatile comfort padding, the sum of the participants' areas would remain constant throughout a SWARM run, irrespective of the layout zone which closes in towards its final size as illustrated in Figure 7.72 (a). Remembering the considerations at the end of Section 7.4.2, a characteristic drawback of such a tightening approach is that the fierceness in the competition of the interacting participants usually peaks out in the middle (i.e., in the second episode) of the self-organization phase.

This is where the idea of volatile comfort padding can help to equilibrate the evocation of emergent behavior. Figure 7.72 (b) shows an exemplary graph for a volatile comfort padding share of $c_\text{v} = \frac{1}{2}$. In mathematical terms, one could say that *without* volatile comfort padding, the derivative of the participants' totaled areas with respect to the progression (regression) of the major and minor tightenings (enlargements) –indexed with i and j– is zero, whereas *with* volatile comfort padding, the derivative is

$$\frac{\mathrm{d}}{\mathrm{d}(i,j)}[\text{⁙}\mathcal{P}]_{\text{p+s+v}} = c_\text{v} \cdot \frac{\mathrm{d}}{\mathrm{d}(i,j)}[Z_{i,j}]. \tag{7.250}$$

The positive effect of such a correlation is, that –on the one hand– the occurrence of the more competitive interaction situations is preponed and thus tackled sooner, while –on the other hand– the pressure on the interacting participants not only increases with every zone tightening but simultaneously decreases due to the diminution of the volatile comfort padding.

This statement is exemplified in Figure 7.73. For comparison, one may first consider a situation *without* volatile comfort padding as in (a1). After the tightening of the layout zone (a2), the ten participants pass a round of interaction. The primary intention of the participants is to get inside the layout zone, which leads to a constellation teeming with interference (a3). In particular, interference on a corner of a participant is impedimental since it rules out a possible *Budging* action. Thus, the participants

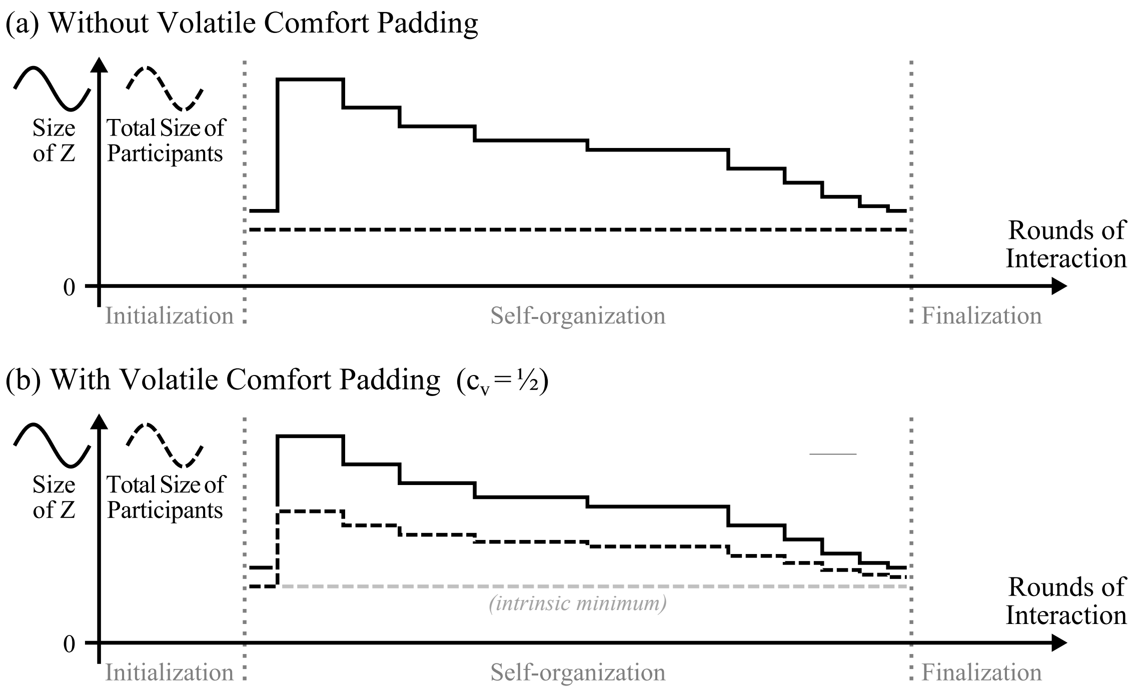


Figure 7.72: Total size of the participants (a) without and (b) with ~~vo~~latile comfort padding.

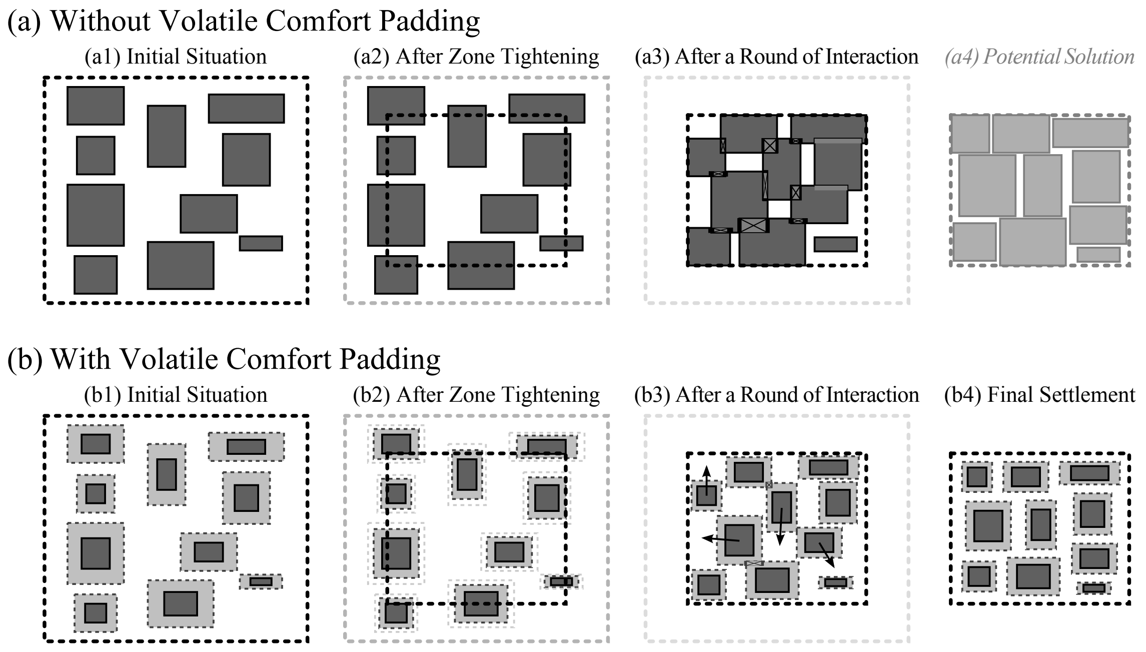

(a) Without Volatile Comfort Padding
(a1) Initial Situation
(a2) After Zone Tightening
(a3) After a Round of Interaction
(a4) Potential Solution
(b) With Volatile Comfort Padding
(b1) Initial Situation
(b2) After Zone Tightening
(b3) After a Round of Interaction
(b4) Final Settlement

Figure 7.73: Two equivalent interaction situations showing the effect of volatile comfort padding.

in this example have immense difficulty to unravel the abundance of conflicts, even though the current constellation is quite close to a potential solution (a4).

Now, Figure 7.73 (b) devises an analogous example *with* volatile comfort padding. For that purpose, the ten participants are assumed to be identical to those of example (a), while the layout zone is initially larger. To make a point, one may consider a situation such as the one in (b1) which is –due to the volatile comfort padding of the participants– equivalent to (a1) from SWARM's point of view, albeit on a larger areal scale.[23] In temporal regard, this means that situation (b1) is encountered sooner than situation (a1) within the SWARM run. The zone tightening, shown in image (b2), is accompanied by a

[23]Figuratively speaking, image (b1) in Figure 7.73 portrays the initial situation of example (b) from a farther distance than image (a1) portrays the initial situation of example (a).

diminishment of the participants' volatile comfort padding, and for that reason the subsequent struggle is not as competitive as in example (a). Thus, after completing one round of interaction, the participants attain the constellation of (b3) which exhibits only two occurrences of interference. With just four *Budging* and *Centering* actions (see arrows), easily determined by the respective participants, the way is paved to reach the final settlement (b4). This constellation is equivalent to the potential solution of (a4), so the rest of the self-organization phase is trivial and demands only a couple of further *Centering* actions.

7.5 Final Remarks About the Conception of SWARM

Now that the layout automation methodology of SWARM has been described in detail, the following Section 7.5.1 throws a glance at a couple of methodological aspects in comparison to similar conceptions found in classical optimization algorithms (as covered in Section 3.1.1). Then, Section 7.5.2 is about to elucidate where SWARM shares common ground with existing models of decentralized systems (as discussed in Section 6.5) and where it takes a different route.

7.5.1 Comparison with Optimization Algorithms

So far, this chapter may have left the impression that the SWARM methodology covertly borrows a couple of already-known ideas from the existing crop of optimization-based automation approaches. Although this is not entirely unfounded, a distinct line of demarcation can still be drawn between SWARM and those classical optimization algorithms, as the following explanations –covering several aspects– are about to show.

Nested Loops

In its basic form, as depicted in Figure 3.1, an optimization algorithm iteratively explores the solution space to determine possible candidate solutions and evaluates these candidates to assess their quality. Depending on the respective algorithm, this iterative strategy can involve multiple loops whereat such a loop may also be nested inside another one. For example, the partitioning algorithms referred to as *Kernighan-Lin* (KL) [335] and *Fiduccia-Mattheyses* (FM) [336] are both built upon a pair of nested loops. This set-up bears a striking resemblance to SWARM, whose self-organization phase recursively iterates over a loop that is nested inside another loop, as shown in Figure 7.3. But a detailed look on the respective mechanisms reveals the distinction between these partitioning algorithms and SWARM.

The KL algorithm works by repeatedly swapping two cells from different partitions while FM consecutively moves cells from one partition to the other. This is fictitiously done in an inner loop until all cells have been swapped/moved. After that, only some of these swaps/moves are effectively performed. This represents a so-called *pass*, which is followed by further passes due to the outer loop. In SWARM, the inner loop encompasses n rounds of interaction wherein each participant may act n times (whereas in KL or FM each cell is only swapped/moved once –at most– within one pass). Another distinction is that there are no fictitious actions in SWARM – an action is either performed or not. SWARM's outer loop comprises the tightening-settlement cycles. As the name implies, the layout zone gets tightened between these cycles, i.e., the situation is changed prior to the outer loop's next iteration. In contrast, the input solution to a pass in KL and FM is always equal to the output solution of the previous pass.

Condition of a Participant

One elementary trademark of an optimization algorithm can be found in the formal metric used to rate the quality of a candidate solution [337]. As already mentioned, this is typically done via a cost function which incorporates several optimization goals and adds up to a numerical value enabling a relational comparison (as covered in [338], for example). This is reminiscent of a participant's *condition* in SWARM (see Section 7.3.1), a construct used to encapsulate several *influencing factors*. Two of these, *interference* and *turmoil*, are –as with a cost function– even used for a numerical comparison (Section 7.3.4.1), although other ones, such as *wounds*, affect the module interaction in a more elaborate way.

A decisive difference between SWARM and optimization algorithms in this regard is that in SWARM, following the idea of decentralization, a participant's condition (or, more precisely, the benefit of its prospective condition in relation to its present condition) merely pertains to that specific participant and affects nothing but that participant's next action. So, at no point during a SWARM run is the fitness of the overall layout solution quantified as a whole, since each participant only needs to care for its own personal well-being (namely described by its condition).

Handling Overlaps

The matter behind SWARM's influencing factor *interference* (Section 7.3.1.1) is also a common element of optimization-based placement algorithms: an overlap of design components. Yet again, this apparent similarity comes into play in a different way here. In a placement algorithm, the occurrence of overlaps depends on the representation used to describe the positions of the components. In a *topological representation*, where the component positions are described relative to each other (as in the *slicing model* introduced by [339]), the components cannot overlap. In contrast, an *absolute representation* (introduced in [340]) permits illegal overlaps during the optimization, taken into consideration by a (typically quadratic) penalty cost term that is to be minimized. In that case, the overlaps may even subsist in the final solution and need to be eliminated through a dedicated post-processing step [341].

SWARM differs from both of these practices as overlaps are –in contrast to placement algorithms based on topological representations– indeed allowed during the module interaction, but are –in contrast to placement algorithms based on absolute representations– definitely reduced until obliteration during each tightening-settlement cycle. Furthermore, the overlaps are not incorporated in a simple linear or quadratic fashion, but involve further expedients (such as *aversion*) shown to have a positive impact on the overall flow of self-organization. And above all, the overlaps in SWARM are not just an undesired nuisance, but also an essential driving force behind its module interaction.

Weight Versus Emphasis

The influencing factor *turmoil* (Section 7.3.1.2) includes a quantity named *emphasis* to specify the importance of a connection between two participants in a SWARM run. This may bring a similar notion to mind, generally known from optimization algorithms under the term *weight* or *weighting factor* [342]. Most prominently, each individual summand of an optimization algorithm's cost function is usually multiplied with a certain weighting factor. This way of controlling how the evaluated solution quality depends on a certain criterion allows to prioritize the various optimization goals that the algorithm is supposed to pursue. Since a cost function usually embraces the solution as a whole, the targets of these optimization goals can be quite diverse in nature. For example, the three penalty addends of the cost function in *TimberWolf* [343] –(1) total wirelength, (2) cell overlaps, and (3) inequality of row lenghts– represent different quantities with different dimensions.

In SWARM however, the emphasis is –unlike a weight– not applied to diverse optimization goals in general, but only with respect to a participant's connections in order to determine the participant's overall turmoil. Furthermore, a look at equation 7.34 shows that the emphasis does not appear therein as a numerator, but as a denominator (to calculate the so-called *relaxation threshold*) and can therefore not be considered a weighting factor in that sense. In equation 7.48, the emphasis does indeed appear as a numerator to compute the *tension* in a connection, but not to play different kinds of opposing optimization goals off against each other in order to make a compromise on the global scale. Instead, suchlike reciprocities are globally resolved by the locally incentivized actions of SWARM's participants.

Tractive Forces

As discussed in Section 7.3.1.2, the concept of *tension* influences a participant's actions like the tractive force of a rubber strap. The same suggestion also lies behind *Force-Directed Placement* [45], an optimization-based approach where –viewed from the physics of classical mechanics– the placement problem is modeled as a spring-mass system. For each pair of connected components, this approach lets the two components attract each other as if joined by an extension spring, which means that the tractive force is linearly proportional to the spring's deflection (given by the distance between the components).[24] Thus, the placement problem is solved by finding the *mechanical equilibrium*, i.e., by calculating a location for each component such that the *net force* on it becomes zero (or at least minimal).

Despite the striking similarity, SWARM's employment of a tractive force is quite different. Above all, the concept of tension in SWARM only belongs to one of several influencing factors. Apart from this, the magnitude of the force does not obey *Hooke's law* [344], but follows a combined linear and quadratic characteristic (Figure 7.20), also featuring a quantity called *strength* to consider a participant's total number of connections (thus streamlining the self-organization). Furthermore, the net force in SWARM is not calculated vectorially but represents a scalar value and can therefore never be zero (unless all participants lie centered on top of each other). Instead, SWARM defines the *relaxation threshold* which is correlated with the size of the layout zone –to account for its repetitive tightening– and facilitates overlap-free arrangements without the need to withdraw a component from an already occupied location (as would be necessary in Force-Directed Placement).

Free Peripheral Space and Prime Rectangles

Presumably, SWARM's conception of the so-called *free peripheral space* –as covered in Section 7.3.2– looks kind of crude for the reason that it may not be really "free" due to *pervasion* (for example because of its *blind spots*). Stating that this observation is not that much of a problem (see Section 7.3.2.2) may also have a stale aftertaste to it, sounding as if this deficiency has to be conceded due to the lack of a better conception. In contrast, other works such as [345] resort to the seemingly neater notion of *prime rectangles* since such a prime rectangle has the valuable predicate that is is always surely vacant – by definition [346].

Yet, the introduction of free peripheral space in SWARM is justified for two major motives. First, the geometrical recipe for perceiving the free peripheral space (Section 7.3.2.1) is relatively simple and can even be formulated as a closed functional expression. Hence, it does not necessitate any programming constructs (such as iteration loops and conditional statements) and gets by with quite simple graphical operations (e.g., determining the intersection of two rectangles). Second, there is –by definition– always exactly one free peripheral space per participant at any given point during a SWARM run (whereas there could be a multitude of prime rectangles). In that regard, a participant's doing is well-determined and straightforward. Altogether, both motives help to keep a participant's efforts low on the local scale, which is quite in line with SWARM's philosophy of decentralization and its aim of provoking emergent behavior on the global level. With this mindset, thinking of SWARM as a *complex system*, interference due to blind spots might even be crucial to achieve self-organization because –as already stated before– component overlaps represent a fundamental impetus for the overall flow of module interaction.

Actions Versus Perturbations

Some of SWARM's native actions –described in Section 7.3.3.1– might be evocative of similar implementations found in classical optimization algorithms. In particular, a counterpart of the *Swapping* action (whereby a participant trades places with another participant in SWARM) is commonly used by placement approaches based on *Simulated Annealing* to generate a new placement state with every iteration. For example, the TimberWolf tool referenced above can (apart from attempting a single-cell displacement) employ such a modification –in the context of Simulated Annealing typically called a *perturbation* [347]– to perform a pairwise interchange of cells [348]. That interchange may also include a change in orientation (as done by SWARM's *rotation* morphism). Thus, the idea of a cell interchange is nearly identical to SWARM's *Swapping* action – however, it differs in the way that it is utilized.

Simulated Annealing works stochastically, i.e., the decision whether to perform a single-cell displacement or a pairwise interchange (or, in general, any other perturbation) is random [349]. Likewise, the cell to displace or –accordingly– the pair of cells that are to be interchanged is selected randomly. In contrast, the action exploration (including the choice of involved participants) in SWARM proceeds pursuant to a completely deterministic agenda. Just as well, the action to perform in SWARM is picked

[24] In fact, the analogy with springs is not entirely correct because a spring's ends already have a certain distance in the spring's resting state.

according to a well-defined and unambiguous comparison metric (see Section 7.3.4.1). Concerning the single-cell displacement, it must be further recognized that SWARM does not feature only one suchlike perturbation but a couple of actions where the number of acting participants amounts to 1 (e.g., *Centering*, *Budging*, and *Yielding*). Choosing an action, every participant in SWARM follows clear preferences in its decision-making because every participant is meant to embody a rational agent (see Section 6.5.3). In Simulated Annealing however, the acceptance or rejection of a (worse) solution is again a question of probability and therefore not "rational" nor deterministic at all.

Greediness

The willingness of temporarily accepting an interim solution that is worse than the current candidate, represents a particular feature of Simulated Annealing and a couple of other optimization algorithms (such as the partitioning algorithms KL and FM mentioned above). In contrast, many layout automation algorithms are *greedy*: they only permit an improvement of the current layout solution by always taking a better neighboring candidate with every iteration (and coming to a stop if no immediate improvement can be found) [350]. Due to this attitude, such algorithms are not able to overcome local optima and therefore they often fail to find the globally optimal solution [351]. Since the participants in a SWARM run always strive for *beneficial* actions (to improve their condition), SWARM also seems to be greedy in this algorithmic sense. However, this can not be definitely said for several reasons.

First, it should be understood that the idea of *aversion* (see Section 7.3.1.1) and the influencing factor *wounds* (Section 7.3.1.4) have a kind of memory effect on their participants. Because of this, an action that might have been rejected some rounds of interaction before may now become beneficial. So, although that action is now –from the participant's subjective point of view– better than before, it is –objectively perceived– still the same as it was back then. Thus, whether to call the participant greedy or not, becomes a rather moot question. Second, while a participant always chooses the best action explored, it can be the case that no beneficial action has been found at all. Then, as mentioned in Section 7.3.4.1, the participant deliberately executes a *Yielding* even if that is not beneficial. So, the participant acts non-greedily regarding its worsened prospective condition, but –on the other hand– yields on purpose (to revive the flow of self-organization). Third, it must be emphasized that all these considerations always apply to the decisions of an individual participant, investigating its local options within its current situation. For that reason alone, the term *greedy* (or non-greedy), which generally refers to the solution as a whole, is inappropriate to describe a decentralized methodology such as SWARM.

Placement Templates

Section 7.3.4.1 states that the locations of the participants in a SWARM run, relative to each other, can be predefined in advance via a placement template, thus enforcing a desired overall arrangement. This idea has already been exploited by a couple of optimization-based layout automation works in the past. For example, the "design by example" approach of [352] requires a sample layout –the template– to automatically produce other layout variants by availing the expert knowledge embedded in that template (such as the device placement, for example). The authors of [89] praise the good quality of the layouts obtained with this technique, but also criticize that a new sample layout must be created manually for each type of circuit. This is different in SWARM, where the relative placement of the participants is specified via constraints (as will be discussed in Section 8.3.2). Apart from this, there are –despite the common idea of using a template– a couple of further distinctions to be revealed on closer examination.

In [352], the placement topology of the sample layout is turned into a *slicing floorplan* description. To also include those device arrangements that would be ruled out by confining the multitude of potential solutions to a single slicing structure, the tool in fact considers an entire set of differing slicing structures. Since there exists an exponential number of slicing structure alternatives for a given placement topology, this in turn led to the introduction of so-called *option slices* (representing alternative binary slicing compositions) to cope with the significant increase in memory size and processing time. In contrast, the constraint-based template description of SWARM not only bypasses the need for suchlike tricks but also covers an even more exhaustive set of possible constellations because it allows for slicing and nonslicing arrangements alike. The tool of [353] converts an existing layout into a symbolic template described via *corner-stitching* [354], providing a separate representation for each mask layer to describe that plane in terms of rectangular *tiles*. While this representation also involves constraints, these are not used to specify the relative positions of design components as in SWARM, but to enforce technological design rules (such as a minimum spacing between two tiles on the same mask layer).

In general, SWARM sets itself apart from other template-based approaches due to the fact that the placement template in SWARM does not apply to primitive devices (nor basic shapes) but compound modules. The relative locations of these modules are explicitly enforced through constraints, while the positioning of the individual devices –and thereby caring for detailed layout issues such as the spacing between these devices– is implicitly managed by the respective module along with the module-internal wiring. This is the power of delegating such low-level layout tasks down to the governing modules in a decentralized fashion. Speaking of decentralization, the following Section 7.5.2 is about to compare SWARM with other decentralized systems.

7.5.2 Comparison with Decentralized Systems

Comparing SWARM with cellular automata (Section 6.5.1), game theory (Section 6.5.2), multi-agent systems (Section 6.5.3), and agent-based models of collective motion (Section 6.5.4), the major similarities and the major differences can be summarized as in Table 7.13.

Table 7.13: Comparison of SWARM with existing models of decentralized systems.

	Cellular Automata	**SWARM Methodology**
Similarities:	• Realizes an interaction of cells on a usually planar two-dimensional grid.	• Realizes an interaction of modules in the two-dimensional layout plane.
	• Complex geometrical patterns dynamically emerge from the interaction of the cells.	• The overall layout solution is expected to emerge from the interaction of the modules.
	• A cell's state can change (e.g., from "dead" to "alive"), which influences its subsequent transitions.	• A module's condition can change (e.g., from "wounded" to "healed"), which influences its further actions.
	• A cell's transition to the next state involves the cell's local neighborhood (its adjacent cells).	• A module's decision-making involves the module's local surrounding (its free peripheral space).
Differences:	○ All cells are arrayed in a strictly tiled, space-discrete manner.	○ The layout plane is continuous (down to the marginally small design grid).
	○ The location, form, and size of a cell is fix and does not change.	○ The modules can move through the layout plane and deform themselves.
	○ The evolvement of patterns can be highly dependent on the initial population (e.g., in Conway's Game of Life).	○ The emergence of the final layout solution is desired to be largely independent of the initial constellation.
	Game Theory	**SWARM Methodology**
Similarities:	• Used to study the strategic interaction among independent players.	• Can be regarded as an infinitely repeated game played by the modules.

to be continued on next page

Table 7.13: Comparison of SWARM with existing models of decentralized systems (continued).

	• Distinguishes between different criteria by which various types of games can be classified.	• Each round of interaction represents a noncooperative, discrete, asymmetric, non-constant-sum, sequential, perfect-information stage game.
	• Is particularly targeted at situations where the desires of the different players conflict.	• The interaction of the participating modules is built on competition, not on cooperation.
	• Each player is self-interested and wants to maximize his or her own payoff.	• Each module is self-interested and wants to improve its own situation as much as possible.
Differences:	○ A player's desires are expressed via a utility function, mapping the player's preferences to a real number.	○ A module's desires are modeled through its condition, which in turn depends on five influencing factors.
	○ A player's decision-making may consider the potential actions of the other players (referred to as a *complete-information game*).	○ A module's decision-making ignores the potential subsequent actions of the other modules (denoted as an *incomplete-information game*).
	○ Noncooperation often leads to an outcome that is inferior for all players (e.g., see the Nash equilibrium in the Prisoner's Dilemma).	○ The noncooperative, competitive setting is supposed to result in an outcome that is optimal for all modules.
	Multi-Agent Systems	**SWARM Methodology**
Similarities:	• An agent is able to perform actions upon its environment.	• A module can see its layout context and react to environmental changes (such as a zone tightening).
	• An agent is autonomous in its decisions about how to act.	• Each module chooses its own actions by itself (possibly restricted by design constraints).
	• An agent has a particular degree of intelligence (and often mobility).	• The modules are implemented as computational entities and can move through the layout plane.
	• An agent has a limited viewpoint with only incomplete information about its environment.	• A module's free peripheral space is locally confined by the module's nearest obstacles.
	• An agent is rational in that it adheres to a goal-directed behavior following clear preferences.	• Every module acts in a clear and deterministic way, pursuing the goal of becoming (or staying) contented.
Differences:	○ The agents typically co-exist beside the problem that they solve (e.g., an object that they control).	○ The modules do not only help in solving the problem but also represent a part of its solution.

to be continued on next page

Table 7.13: Comparison of SWARM with existing models of decentralized systems (continued).

	○ Often involve control architectures for coordinating the agents and reaching consensus.	○ The interacting modules do not exchange information and do not rely on mutual consent.
	○ Usually each agent is implemented as a single and enclosed entity.	○ Multiple modules can be organized as a hierarchical module association.
	Agent-Based Models	**SWARM Methodology**
Similarities:	• Used to animate a spatially explicit, collective motion of an ensemble through a geographical sphere.	• Used to evoke a spatially explicit, collective motion of layout modules through the layout plane.
	• Inspired by the movement of large animal groups such as a flock of birds or a school of fish.	• Inspired by the movement of livestock during roundup (for example, a herd of sheep).
	• The ensemble members are mainly driven by biological emotions and sensations (e.g., fear and hunger).	• The modules are influenced by natural feelings and impacts (including tension, aversion and wounds).
	• A member's behavior can change over time due to the inclusion of memory into its behavioral model.	• Aversions and wounds abate only slowly and thus have a memory effect on a module's decision-making.
	• Interested in simulating how the ensemble manages to avoid collision with other moving environmental obstacles.	• The modules constantly need to adapt themselves to their environment which changes with every zone tightening.
Differences:	○ All ensemble members are created equal.	○ The modules may have individual looks, tasks, goals and abilities.
	○ The ensemble members are typically modeled as dot-like incorporeal entities.	○ Each module has a "body" with a nonzero width and height (and thus an area).
	○ The overall motion of the ensemble is a polarized coherent movement with smooth shifts in direction.	○ The overall motion of the modules is a centripetal aggregation with turbulences such as jumps and swaps.

As Table 7.13 shows, SWARM embeds a lot of ideas and concepts found in existing models of decentralized systems. Apart from this, every principle of self-organization covered in Section 6.4 can also be encountered in the SWARM methodology (see Figure 7.74):

The Basic Constituents of Self-organization: Above all, the three core concepts of SWARM embody the basic constituents of self-organization introduced in Section 6.4.1: the *responsive modules* correspond to the *workers* (and indeed perform the actual layout work), the *module interaction* follows well-defined behavioral *rules* of action (in accordance with each module's individual desires and abilities), and the *interaction control* organ steers the overall flow towards the desired outcome by exerting *pressure* (and imposing the *goals* as constraints).

Operational Closure and Structural Coupling: The notions of *operational closure* and *structural coupling* (Section 6.4.2) can be identified in the limited authority of the interaction control organ which is not able to manipulate the module interaction directly but can only spur the modules by changing their environment (i.e., decreasing the size of the layout zone). Thereby, the interaction control

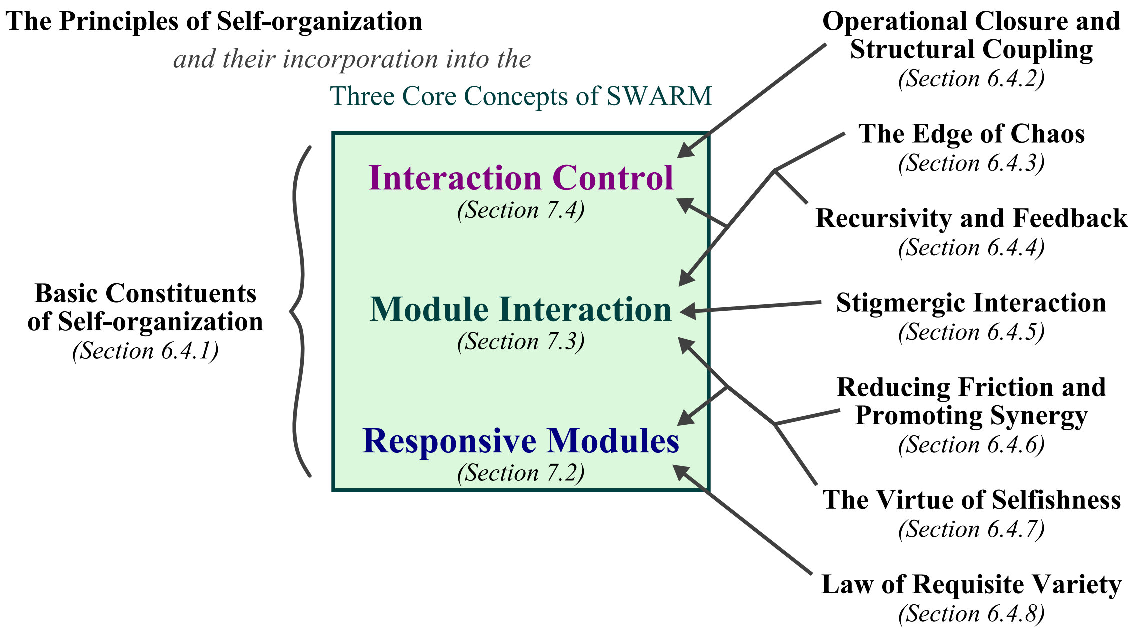


Figure 7.74: Incorporation of the principles of self-organization into the three core concepts of SWARM.

organ leaves it up to the *cognitively open* network of interacting modules to adapt itself to the new situation by changing its internal structure (i.e., the layout arrangement) on its own behalf, using innate operations (i.e., the modules' catalog of actions).

The Edge of Chaos: Operating at the *edge of chaos* (Section 6.4.3) is a principle of self-organization which affects both the interaction control organ and the module interaction. Concerning the interaction control organ, the edge of chaos corresponds to a moderate tightening policy that is neither too lenient (which would risk to let the modules drift towards stagnation) nor too aggressive (which could lead the interaction into an erratic mêlée). Regarding the module interaction, the edge of chaos is found in the fine-tuned taring of aversions and wounds, whereby the self-organization proceeds on the thin red line between overly static (i.e., firm but unfertile) behavior and overly dynamic (i.e., fruitful but frantic) behavior.

Recursivity and Feedback: *Recursivity* (Section 6.4.4) can be ascribed to the way in which the interaction control organ drives the self-organization through successive tightenings of the layout zone. Therein, the output of each tightening (i.e., the settlement) serves as the input for the next tightening. One might claim that this is no different from any iterative algorithm, but whenever a layout constellation in SWARM is again encountered a second time, the subsequent actions will probably deviate from those of the first time due to the memory effect of aversions and wounds (internal quantities that make the system behavior history-dependent). The module interaction's *feedback* to the control organ is the difficulty (i.e., the number of necessary interaction rounds) to obtain a settlement. Feedback can also be observed in the interaction itself, both positive and negative. A module moving into a position of interference may be repulsed and driven back again (thus withdrawing the move = negative feedback) or may cause the other modules to relent and back off (thus endorsing the move = positive feedback).

Stigmergic Interaction: In contrast to request-and-response schemes commonly employed in multiagent systems, the module interaction in SWARM is not based on direct communication but on *stigmergy* (Section 6.4.5). Just like the movement of an ant in an ant colony inevitably leads to a deposition of pheromones which in turn attracts other ants (see Figure 6.4), a module's actions inevitably modify the environment of the others (e.g., the free peripheral space of the module's neighbors). Furthermore, the modules also coordinate themselves via the influencing factors that they want to alleviate (such as overlaps and tension), equivalent to the stigmergic way in which the traffic lights of Figure 6.10 co-control each other via the traffic that they regulate.

Reducing Friction and Promoting Synergy: The idea to reduce *friction* and promote *synergy* (Section 6.4.6) has been worked into SWARM's comparison metric for evaluating explored actions (and thus sits on the fence between a responsive module's decision-making and its inherent effect on the overall module interaction). Actions that involve multiple participants are rated by determining the collective interference, turmoil, or relaxation delta of all involved participants (not just the leading participant). Hence, synergistic actions with mutual benefit for multiple participants are automatically favored over frictional actions by which the leading participant only profits from the sacrifices of others.

The Virtue of Selfishness: The virtue of *selfishness* (Section 6.4.7) is another self-organization principle that can be detected in the decision-making of the individual responsive modules and thereby has an impact on the overall module interaction. The selfishness resides in the modules' egoistic pursuit of their personal contentment: an action is only performed by the respective module if it improves that module's situation. This is definitely true for actions that consists of a single transformation and for *Hustling* actions. Only in rare cases where the module performes a *Swapping* or a *Pairing* action it may happen that the module concedes certain losses for the greater benefit of its counterpart.

Law of Requisite Variety: Last but not least, Ashby's *law of requisite variety* (Section 6.4.8) is reflected in the layout variability of SWARM's responsive modules. The greater the variety of a system is, the better are its chances to cope with changes in its environmental conditions. In analog layout design, these conditions are represented by the constraints that define each specific design problem. So, following the law of requisite variety, encouraging a module to cover as many layout variants as possible, and exploiting that variability during a module's action exploration enlarges the system's room for maneuver and increases the probability of effectuating a layout arrangement wherein all constraints are really satisfied.

As can be seen, paying respect to the principles of self-organization is an ambition that has been embraced throughout all three core concepts of the developed SWARM methodology. From the perspective of complex adaptive systems, these principles are meant to equip SWARM with the ability to "survive" situations marked by unforeseen disturbances, i.e., to produce layout solutions in full-custom quality for various design problems with settings of design constraints that have not been completely anticipated in advance.

Beyond this aim, the conception of a layout methodology such as SWARM –implementing a system of self-organizing layout modules– can be considered *perfect* if the success of the self-organization is largely independent of the initial module constellation. In terms of chaos theory, this means that the system is engineered in a way such that the desired solutions represent *attractors*, i.e., stable states towards which the system tends to evolve for a wide range of starting conditions. Part III of this thesis is about to show inhowfar the current implementation of SWARM can already be seen to hit that mark.

Part III

The Implementation

Chapter 8

Implementation and Results

Any sufficiently advanced technology
is indistinguishable from magic.
Arthur C. Clarke (British writer)

The SWARM methodology, as described in Chapter 7, has been implemented in the programming language SKILL [109] to be used in the Cadence *Virtuoso* IC design environment. Although SWARM is still in its infancy, its current implementation is already advanced enough to achieve various remarkable results – as this chapter is about to show. First, Section 8.1 provides a couple of examples to illustrate the evocation of *emergent behavior* in SWARM. Next, Section 8.2 demonstrates how SWARM can be successfully employed for the practical purpose of *floorplanning*. Finally, Section 8.3 targets the main application area of SWARM, displaying how it can be utilized to tackle analog *place-and-route* problems. As will be seen, SWARM manages to coalesce the chief advantages of both *top-down automation* and *bottom-up automation*: the versatility of an automatism and the quality of its resulting layouts.

8.1 Examples of Emergence in SWARM

As has been explained in Chapter 6, the intention of the SWARM methodology is to let an optimal layout outcome *emerge* from the interaction of responsive layout modules, similar to the way captivating patterns can evolve in a *cellular automaton*. To substantiate this idea, the following examples illustrate inhowfar the phenomenon of emergence can really be observed in SWARM, distinguishing between the emergence of a collective motion (Section 8.1.1), the emergence of an optimal layout outcome (Section 8.1.2), and also the –unfortunate– emergence of interaction cycles that never terminate (Section 8.1.3).

8.1.1 Example of an Emerging Collective Motion

Figure 8.1 presents a simple SWARM example –also given in [355]– that shows the step-by-step interaction of two participants P_1 and P_2 inside a rectangular layout zone. For the sake of exemplification, these participants are only symbolic modules, which is to say that they are not imposed on real layout devices. Additionally, neither rotations nor deformations are taken into consideration during the action exploration in this example, i.e., the participants can only perform translational moves.

The individual actions of the two participants can be traced throughout images (a) to (f) in the upper portion of Figure 8.1. First, image (a) displays the initial constellation with the two participants lying completely inside the layout zone. Next, the layout zone is tightened as in (b) whereby both participants become *prone* and strive to take an action. Participant P_1, with the desire to become *safe* again, executes an *Evasion* move by going downwards and to the left, thus aligning itself with the north-western corner of the layout zone (c). In that location, there is only little overlap with P_2 compared to the *interference* that would be sustained if P_1 moved straightforwardly down. For the same reason, P_2 then also performs

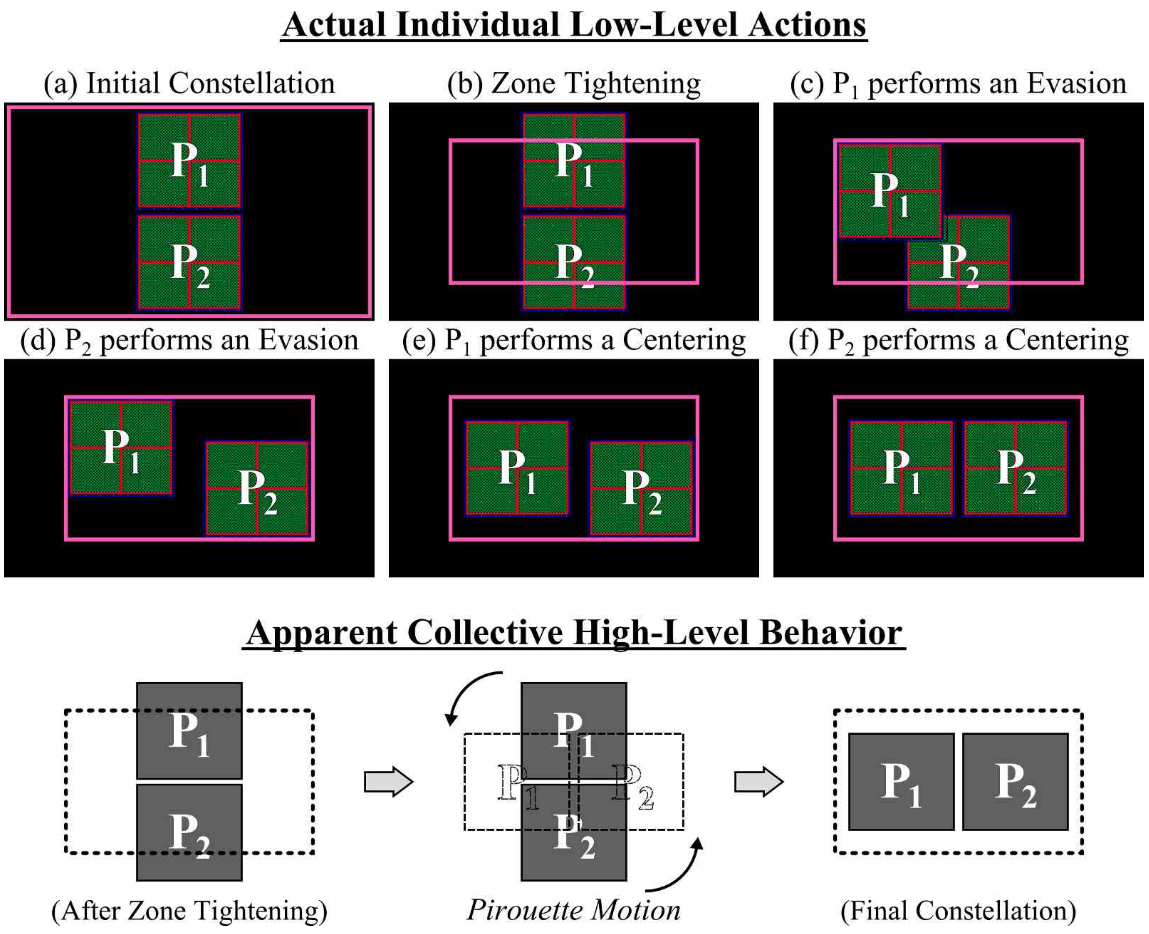

Actual Individual Low-Level Actions
(a) Initial Constellation
(b) Zone Tightening
(c) P1 performs an Evasion
(d) P2 performs an Evasion
(e) P1 performs a Centering
(f) P2 performs a Centering
P1
P2
Apparent Collective High-Level Behavior
(After Zone Tightening)
Pirouette Motion
(Final Constellation)

Figure 8.1: Example of how high-level behavior can emerge from SWARM's low-level interactions.

and *Evasion*, wandering upwards and right into the layout zone's south-eastern corner (d). Since both participants have become *safe* and *clear* (and thus *contented* in this case), first P_1 begins *Centering* itself within its *free peripheral space* (e) just like P_2 also performs a *Centering* until both of them reach a stable settlement in their final locations (f).

Comparing the initial constellation (a) with the final constellation (f) makes it clear that P_1 and P_2 have been forced to re-organize themselves by turning from an above-below arrangement into a side-by-side arrangement. As the six discussed images in Figure 8.1 put on record, the two participants actually achieved this through a successive series of individual actions (utilizing two native types of actions: *Evasion* and *Centering*). But, looking at this low-level interaction from a higher-level perspective (as done in the lower part of Figure 8.1), the participants' movements between the incipient situation (after the zone tightening) and the resulting arrangement (the final constellation) give the impression as if the modules had virtually performed a *pirouette motion* around their common center point, keeping their orientations like the gondolas of a Ferris wheel.

This overall motion represents a fine example of how an apparently solidary, collective behavior can be seen to emerge from the basic actions of the self-interested participants. Just like the cells in Conway's Game of Life (Section 6.5.1) do not have any knowledge of the patterns that evolve from their aggregate state transitions, the emergence of the seemingly harmonic pirouette motion in Figure 8.1 is beyond the grasp of the involved participants and can only be perceived by an outside observer – as is typical for emergent phenomena.

8.1.2 Examples of an Emerging Optimal Layout Outcome

A complete placement exercise is depicted in Figure 8.2, with ten selected snapshots giving an account of the entire SWARM run that is witnessed to solve the problem. As in the example of Figure 8.1, the given layout zone is rectangular and again the seven participants are implemented as symbolic modules that may solely perform translational movements. Each participant has *volatile comfort padding* –but no *solid comfort padding*– (see Section 7.4.3), as visualized by the white rectangles that are drawn around the modules.

The initial constellation is not *viable* because no participant is clear and two of them (P_1 and P_3) are even prone. With several rounds of interaction, the participants manage to struggle themselves free from each other and occupy locations where all of them are contented – this viable constellation represents the 1st settlement (b). After the subsequent tightening of the layout zone, the overall arrangement changes only slightly, with participant P_4 moving from its location below P_1 to the right of P_1 (c). The relative placement in this 2nd settlement is maintained until the 5th settlement inclusively (d). Following another zone tightening, the subsequent round of interaction leads to the intermediate, unviable constellation of (e), where P_4 now overlaps P_1. This arrangement persists until the interference causes the jammed participant P_1 to perform a *Swapping* with P_7 (f). After several more rounds of interaction, P_4 has built up much aversion against P_7 (in addition to the aversion against P_1), making it jump next to –and partially on top of– P_2 (g). A westwards *Budging* of P_2 can diminish but not eliminate the interference, which prompts P_4 to trade places with P_7 after some time (h). This results in an overlap between P_4 and P_6, which induces a *Swapping* of P_6 with P_3, at last leading to a viable constellation that marks the 6th settlement (i). Thus, watching the interaction throughout (e) to (i), it seems as if the problematic interference gets passed around different pairs of participants until it is eventually resolved. The remaining interaction comprises only *Centering* moves, so the overall arrangement does not change apart from becoming more and more compact until the final constellation is reached with the 10th settlement (j).

As should be visible in the depiction, this outcome of the SWARM run represents the *optimal* solution to the placement problem (from which the example was originally constructed). It should also be noted, that this arrangement is completely *nonslicing*: it is not even possible to cut out any rectangular part of the constellation unless at least one participant is torn in half. Therefore, the placement problem in this example truly has just *one* globally optimal solution without any alternative permutations (except for a horizontal and/or vertical flipping of the entire layout, to be sure).

In essence, this example demonstrates the basic ability of SWARM's interacting participants to obtain the *best* possible arrangement by themselves, even if there is only one single optimum available. Since every participant is selfishly focused on its own personal well-being, none of them is aware of the remarkable feat being accomplished at large. Again, only an outside observer recognizes that the participants' collective contentment in the end reflects their global achievement of solving the overall design problem. Hence, the optimal outcome truly *emerges* from the aggregate interaction – quasi as a "byproduct" of the participants' egoistic pursuit of happiness.

Table 8.1: Number of settlements, rounds, and actions in the SWARM run of Figure 8.2.

Settle-ment	**Num. of Rounds**	**Actions per Round of Interaction** 1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	**Num. of Actions**
1st	5	7	5	1	1	0	-	-	-	-	-	-	-	-	14
2nd	4	4	7	1	0	-	-	-	-	-	-	-	-	-	12
3rd	3	6	4	0	-	-	-	-	-	-	-	-	-	-	10
4th	3	5	3	0	-	-	-	-	-	-	-	-	-	-	8
5th	4	6	7	1	0	-	-	-	-	-	-	-	-	-	14
6th	13	6	7	7	7	7	7	7	7	7	7	7	4	0	80
7th	3	5	1	0	-	-	-	-	-	-	-	-	-	-	6
8th	3	5	1	0	-	-	-	-	-	-	-	-	-	-	6
9th	4	5	7	1	0	-	-	-	-	-	-	-	-	-	13
10th	4	5	7	1	0	-	-	-	-	-	-	-	-	-	13

To analyze the presented SWARM run more deeply, Table 8.1 provides a so-called *interaction record*, i.e., a matrix listing the number of settlements, the number of interaction rounds per settlement, the

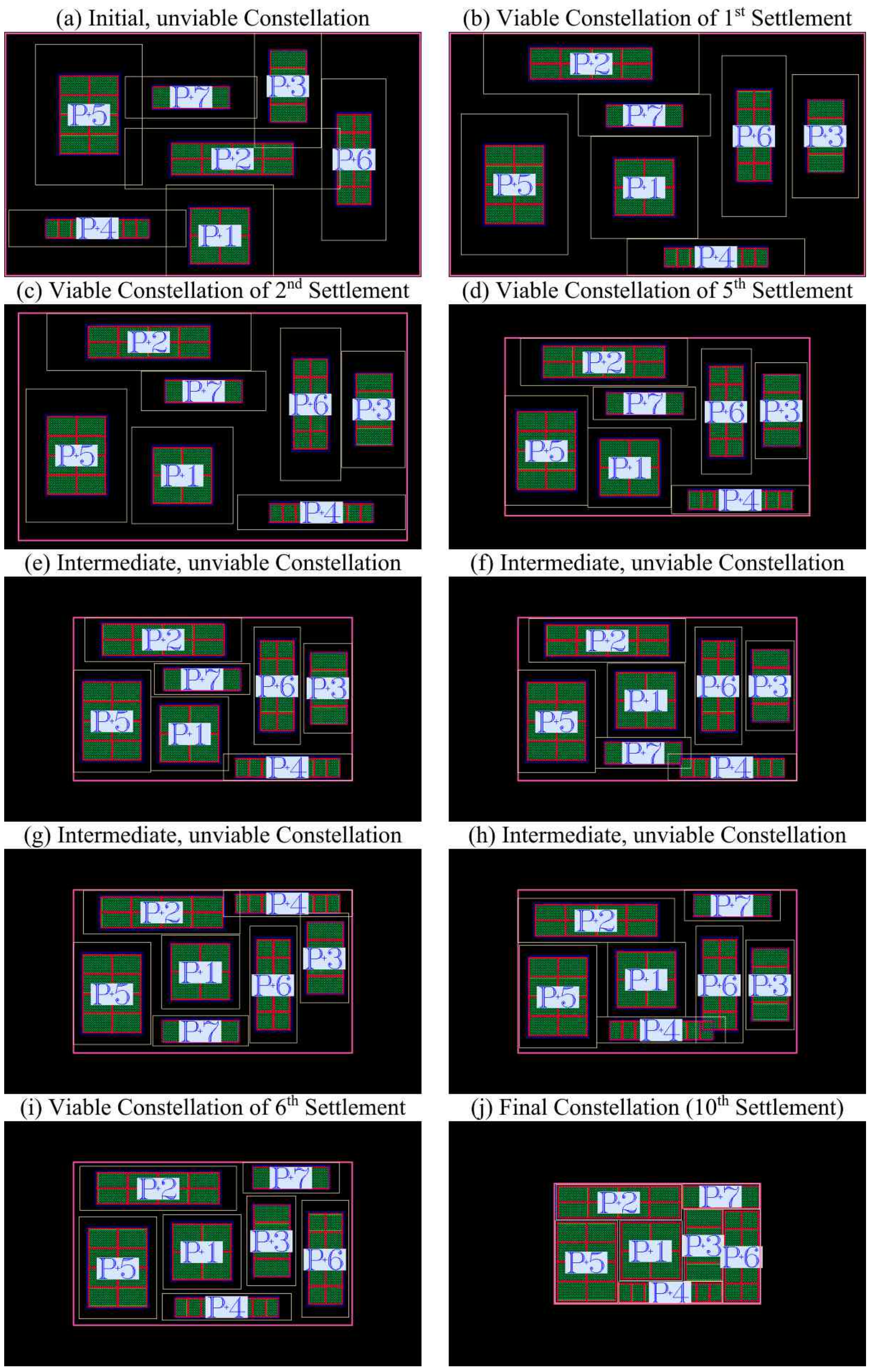

(a) Initial, unviable Constellation
(b) Viable Constellation of 1st Settlement
(c) Viable Constellation of 2nd Settlement
(d) Viable Constellation of 5th Settlement
(e) Intermediate, unviable Constellation
(f) Intermediate, unviable Constellation
(g) Intermediate, unviable Constellation
(h) Intermediate, unviable Constellation
(i) Viable Constellation of 6th Settlement
(j) Final Constellation (10th Settlement)
P1
P2
P3
P4
P5
P6
P7

Figure 8.2: Example of a placement problem with the emergence of the globally optimal outcome.

number of actions per round of interaction, and the total number of actions per settlement. Peering at the facts and figures in this interaction record, four particular observations can be made:

- It may be that during the first round of interaction in a tightening-settlement cycle less participants take an action than in the second round. This is because participants that do not lie near an edge of the layout zone do not become prone from the tightening and therefore do not need to act (until coming in conflict with other participants when these move in to become safe again). From then on, the number of acting participants in general abates until no participant at all performs an action.
- The interaction record exhibits the three characteristic *episodes* that can often be discerned in the *self-organization phase* of a SWARM run (as mentioned in the context of Figure 7.70 on page 182). Here, one can roughly say that the first episode encompasses the first four settlements (with a maximum of 14 actions per settlement), while the second episode begins with the fifth settlement (where the declining number of actions starts to rise again) and peaks out in the sixth settlement (with a total of 80 actions), so the third episode spans the remaining four settlements (seeing the number of actions drop again).
- Although the three characteristic episodes of a SWARM run's self-organization phase can be identified in this example, the flow of self-organization is relatively fluent. This is reflected in the number of interaction rounds per tightening-settlement cycle since the maximum value (13 rounds for the sixth settlement) is less than thrice as large as the arithmetic average (which amounts to 4.6 rounds per settlement). This is a remarkably good achievement among the test cases put into effect so far, as a comparison with the results of the subsequent example is about to show (see Table 8.3).
- The interaction adds up to a total of only 176 performed actions. Of course, the number of explored actions is much greater (with a total of 3368 actions) but still smaller –by several orders of magnitude– than the number of iterations in a conventional optimization algorithm (which usually amounts to several millions). This reinforces the idea of decentralization and local decision-making in SWARM, where the fund of available actions is considerably richer and each individual choice of action is far more intelligent than the savage perturbations and probabilistic search techniques found in existing top-down approaches. The computational effort per action may be larger in SWARM, but nonetheless only 28 seconds of runtime were required to obtain the optimal solution in this case.

To underline SWARM's emergent problem-solving abilities, another placement task is given in Figure 8.3, similar to the previous example but with ten instead of seven layout modules this time. As before, the initial constellation is again unviable (a), yet the participants manage to eliminate all conflicts once again and attain a viable constellation with the 1st settlement (b). The 2nd settlement involves only a minor revision of the overall arrangement (c), which –from then on– remains basically the same until reaching the 12th settlement (d). With the subsequent tightening, the situation gets difficult and asks for relatively many rounds of interaction to settle. Traversing numerous intermediate, unviable constellations –a couple of which being displayed throughout images (e) to (h)– the participants succeed in resolving all overlaps, thus concluding the 13th settlement with the viable constellation depicted in (i). The last tightening incites a further compaction of the arrangement and produces the final constellation, which ends the SWARM run with the 14th settlement (j).

In this example, the participants' interaction has again lead to the optimal outcome, which is once more a nonslicing arrangement as in the previous example. However, there are multiple global optima in this case since some modules can be permuted without increasing the overall area occupation (such as the two participants in the north-eastern corner of the layout zone). A look at the interaction record for this SWARM run –given in Table 8.2– substantiates that the self-organization did not proceed as smoothly as in the previous example: the highest number of rounds amounts to 29 (for the 13th settlement) and is more than five times as large as the arithmetic average (5.5 rounds per settlement).

Another insight from the interaction record here is that the first episode of this SWARM run lasts extremely long, such that the second episode occurs quite late (in the penultimate settlement) and more or less merges with the third episode (i.e., the last settlement). Paradoxically, this shift is rooted in the participants' achievement of obtaining a nearly optimal arrangement already with the 2nd settlement. For that reason, the remainder of the problem-solving is entirely postponed to the very end. Yet, although

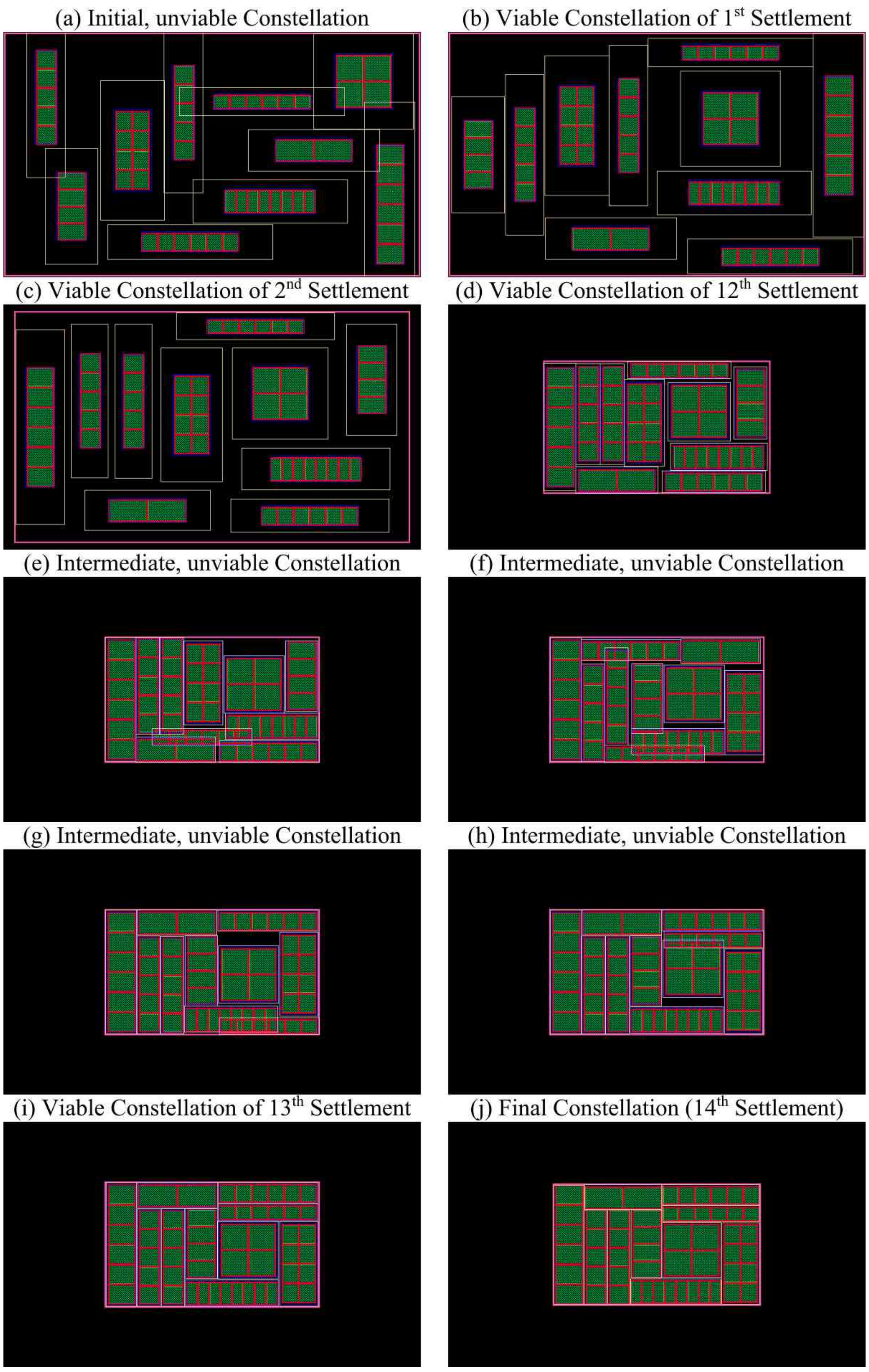

(a) Initial, unviable Constellation
(b) Viable Constellation of 1st Settlement
(c) Viable Constellation of 2nd Settlement
(d) Viable Constellation of 12th Settlement
(e) Intermediate, unviable Constellation
(f) Intermediate, unviable Constellation
(g) Intermediate, unviable Constellation
(h) Intermediate, unviable Constellation
(i) Viable Constellation of 13th Settlement
(j) Final Constellation (14th Settlement)

Figure 8.3: Second example of a placement problem with the emergence of an optimal outcome.

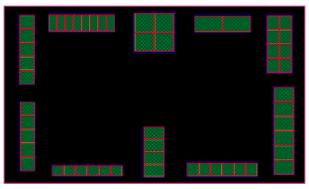
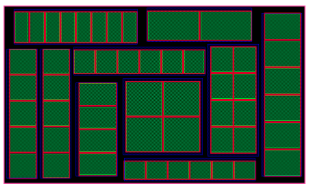
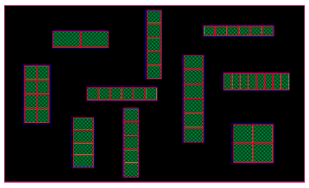
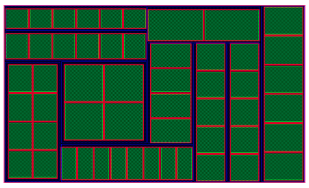
Table 8.2: Number of settlements, rounds, and actions in the SWARM run of Figure 8.3.

Settle-ment	Num. of Rounds	Actions per Round of Interaction													Num. of Actions
		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	...	27th	28th	29th	
1st	3	10	10	0	-	-	-	-	-	-	...	-	-	-	20
2nd	6	5	10	10	3	1	0	-	-	-	...	-	-	-	29
3rd	2	3	0	-	-	-	-	-	-	-	...	-	-	-	3
4th	3	6	2	0	-	-	-	-	-	-	...	-	-	-	8
5th	3	6	3	0	-	-	-	-	-	-	...	-	-	-	9
6th	5	7	10	4	1	0	-	-	-	-	...	-	-	-	22
7th	3	7	3	0	-	-	-	-	-	-	...	-	-	-	10
8th	3	6	10	0	-	-	-	-	-	-	...	-	-	-	16
9th	3	6	10	0	-	-	-	-	-	-	...	-	-	-	16
10th	3	6	10	0	-	-	-	-	-	-	...	-	-	-	16
11th	3	7	10	0	-	-	-	-	-	-	...	-	-	-	17
12th	3	8	10	0	-	-	-	-	-	-	...	-	-	-	18
13th	29	9	10	10	10	10	10	10	10	10	...	10	10	0	279
14th	8	10	9	10	7	8	4	3	0	-	...	-	-	-	51

this imbalance is detrimental for the fluency of the self-organization, this example demonstrates that the participants are still able to abandon such a local optimum –as good as it may be– in favor of an even better (here: the best) solution. The total number of performed actions adds up to 514 (out of 15476 explored actions), demanding a runtime of 145 seconds for the entire SWARM run.

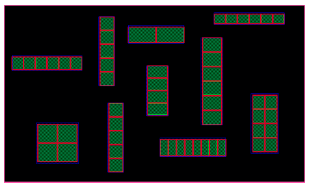
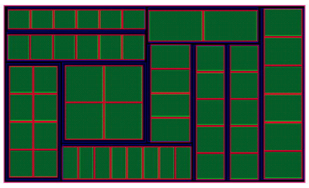
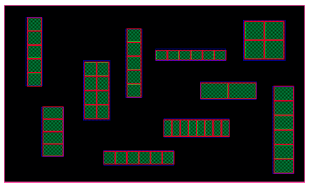
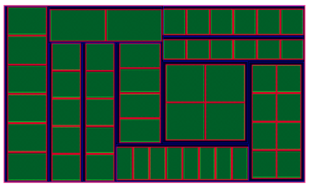
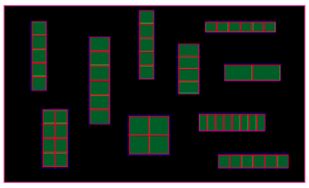
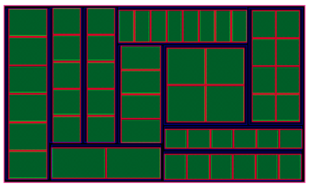
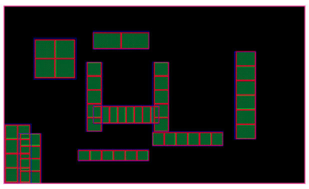
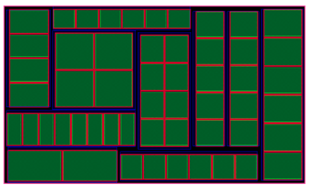
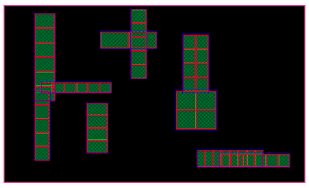
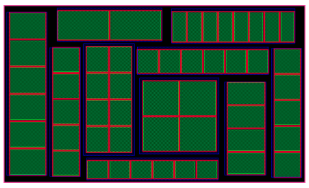
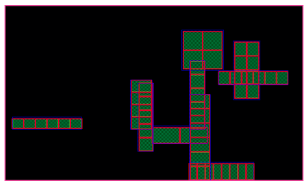
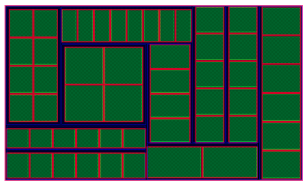
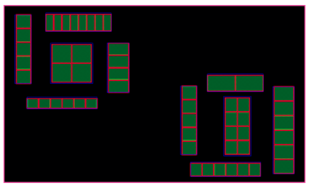
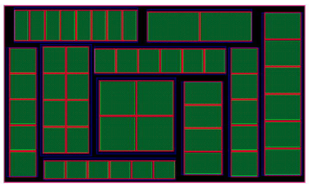
To fortify that the successful outcome in this example is not just a lucky strike, SWARM has been put to the test with ten different initial constellations (six of which were handcrafted while four were randomly created). For each SWARM run, the initial constellation and the final constellation, as well as certain statistics are listed in Table 8.3, sorted by the totally required runtime (with the SWARM run of Figure 8.3 appearing as #4). In six cases (covering runs with handcrafted and randomly created initial constellations), the participants manage to achieve the optimal solution while in the remaining four runs only a nearly optimal placement emerges from the interaction.

Table 8.3: Comparison of different SWARM runs for the same placement problem.

Run	Initial Constellation	Final Constellation	SWARM Run Statistics
#1	 (handcrafted)	 (nearly optimal)	Rounds per Settlement: ($\varnothing = \frac{54}{15} = 3.6$) 4, 4, 4, 2, 3, 3, 2, 3, 3, 4, 3, 5, 5, 3, 6 Number of Actions: 2089 (e) = 1549 (r) + 254 (p) + 286 (d) Total Runtime: 49 seconds
#2	 (handcrafted)	 (optimal)	Rounds per Settlement: ($\varnothing = \frac{77}{14} = 5.5$) 4, 3, 3, 3, 3, 3, 3, 3, 5, 27, 3, 3, 4, 10 Number of Actions: 10302 (e) = 9532 (r) + 531 (p) + 239 (d) Total Runtime: 106 seconds

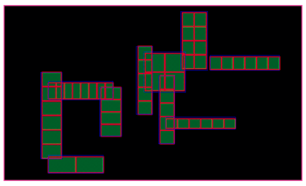
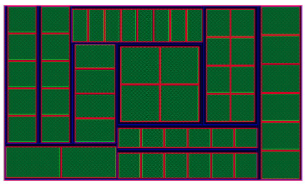
to be continued on next page

Table 8.3: Comparison of different SWARM runs for the same placement problem (continued).

#3	 (handcrafted)	 (optimal)	Rounds per Settlement: ($\varnothing = \frac{79}{13} = 6.1$) 37, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 9 Number of Actions: 15894 (e) = 15104 (r) + 598 (p) + 192 (d) Total Runtime: 137 seconds
#4	 (handcrafted)	 (optimal)	Rounds per Settlement: ($\varnothing = \frac{77}{14} = 5.5$) 3, 6, 2, 3, 3, 5, 3, 3, 3, 3, 3, 3, 29, 8 Number of Actions: 15476 (e) = 14706 (r) + 514 (p) + 256 (d) Total Runtime: 145 seconds
#5	 (handcrafted)	 (optimal)	Rounds per Settlement: ($\varnothing = \frac{95}{13} = 7.3$) 40, 3, 3, 3, 3, 3, 3, 4, 4, 10, 3, 4, 12 Number of Actions: 19529 (e) = 18579 (r) + 740 (p) + 210 (d) Total Runtime: 172 seconds
#6	 (randomized)	 (nearly optimal)	Rounds per Settlement: ($\varnothing = \frac{104}{13} = 8.0$) 4, 3, 3, 5, 3, 8, 4, 5, 7, 32, 3, 12, 15 Number of Actions: 21702 (e) = 20662 (r) + 779 (p) + 261 (d) Total Runtime: 198 seconds
#7	 (randomized)	 (nearly optimal)	Rounds per Settlement: ($\varnothing = \frac{100}{14} = 7.1$) 4, 3, 3, 3, 3, 3, 9, 9, 5, 24, 4, 3, 21, 6 Number of Actions: 23098 (e) = 22098 (r) + 719 (p) + 281 (d) Total Runtime: 208 seconds
#8	 (randomized)	 (optimal)	Rounds per Settlement: ($\varnothing = \frac{112}{13} = 8.6$) 11, 3, 4, 4, 4, 9, 3, 52, 4, 3, 3, 4, 8 Number of Actions: 26597 (e) = 25477 (r) + 879 (p) + 241 (d) Total Runtime: 226 seconds
#9	 (handcrafted)	 (nearly optimal)	Rounds per Settlement: ($\varnothing = \frac{145}{13} = 11.2$) 4, 5, 2, 4, 4, 4, 5, 5, 17, 81, 4, 4, 6 Number of Actions: 34528 (e) = 33078 (r) + 1177 (p) + 273 (d) Total Runtime: 296 seconds

to be continued on next page

Table 8.3: Comparison of different SWARM runs for the same placement problem (continued).

#10			Rounds per Settlement: ($\varnothing = \frac{267}{16} = 16.7$) 5, 3, 3, 4, 4, 28, 3, 177, 3, 3, 3, 4, 15, 3, 3, 6 Number of Actions: 83948 (e) = 81278 (r) + 2408 (p) + 262 (d) Total Runtime: 648 seconds
	(randomized)	(optimal)	

As can be seen, the required runtime is closely correlated with the number of actions involved. This primarily refers to the number of *explored actions* –marked (e)– although Table 8.3 also indicates inhowfar these explored actions divide into

- *rejected actions* (r):
  actions that are either deemed invalid or inferior to other actions,
- *performed actions* (p):
  the actions which are indeed executed by the participant(s), and
- *dismissed actions* (d):
  chosen actions that are not executed since they are *trivial* (primarily because they fall below the minimal movement distance).

A correlation between the runtime and the fluency of the self-organization can also be recognized in Table 8.3. The smoothest flow of self-organization is therefore found in example #1, where the number of rounds per settlement only varies between 2 and 6 (conceding that the final constellation is not the global optimum). A counterexample is given by the SWARM run of #10, climaxing in a total of 177 interaction rounds for the eigth settlement (not in vain since the best possible solution is finally obtained).

To survey the number of actions in greater detail, Table 8.4 presents a so-called *action palette* covering selected SWARM runs of Table 8.3. For each of these runs, the number of explored, rejected, performed, and dismissed actions are listed – distinguishing between the different kinds of native actions.

An attentive reader will probably notice that in the listed SWARM runs, no explored *Lingering* or *Yielding* action is ever rejected. Considering the former kind of action, this is generally always the case (because the participant is definitely contented when the *Lingering* is being explored). In contrast, a *Yielding* move can be rejected for the reason of being invalid. Furthermore, it can be seen that all dismissed actions in the table are *Centering* actions. Although this is usually the case indeed, it does not necessarily have to be like this because actions of other kinds can also be trivial.

More importantly, the following three basic insights can be gained from the given action palette:

- In each SWARM run, actions of almost every kind have been performed (in run #10 even all kinds of actions). This finding fleshes out that the general spectrum and the concrete implementation of SWARM's catalog of native actions is quite feasible.
- By far, the largest exploration effort (and highest rejection rate) naturally pertains to *Budging*, *Swapping*, and *Pairing* actions (to some extent rooted in the exhaustiveness of the respective *exploration plan*). This awareness presumably provides a major starting point for potential performance improvements in further developments of the SWARM methodology.
- The majority of performed actions usually represent *Centering* actions. To reduce runtime, it might be worthwhile to investigate the impact of eliding a *Centering* move in case the participant is contented anyway (especially in the early tightening-settlement cycles of a SWARM run, when there is still much space available).

Notwithstanding these performance aspects, one has to acknowledge that –to a certain degree– the required runtime is deliberately condoned in favor of visualizing a SWARM run since it allows the user to track the module interaction in real-time. This represents a valuable asset of SWARM that would be

Table 8.4: Detailed number of actions for selected SWARM runs of Table 8.3. The abbreviated kinds of actions are: Re-entering (R), Centering (C), Lingering (L), Budging (B), Hustling (H), Swapping (S), Pairing (P), Evasion (E), and Yielding (Y).

		Number of Actions (Distinguished by Kind of Action)									
Run	**Qualifier**	R	C	L	B	H	S	P	E	Y	**Total**
#2	Explored:	106	770	1	3020	47	3224	3060	74	0	10302
	Rejected:	99	102	0	2967	41	3212	3051	60	0	9532
	Performed:	7	429	1	53	6	12	9	14	0	531
	Dismissed:	0	239	0	0	0	0	0	0	0	239
#4	Explored:	106	770	0	4621	69	5435	4400	75	0	15476
	Rejected:	106	137	0	4562	55	5413	4378	55	0	14706
	Performed:	0	377	0	59	14	22	22	20	0	514
	Dismissed:	0	256	0	0	0	0	0	0	0	256
#6	Explored:	100	1040	0	6637	89	7406	6322	107	1	21702
	Rejected:	94	171	0	6545	77	7386	6303	86	0	20662
	Performed:	6	608	0	92	12	20	19	21	1	779
	Dismissed:	0	261	0	0	0	0	0	0	0	261
#8	Explored:	118	1120	5	8430	110	8901	7810	102	1	26597
	Rejected:	118	208	0	8328	97	8869	7778	79	0	25477
	Performed:	0	671	5	102	13	32	32	23	1	879
	Dismissed:	0	241	0	0	0	0	0	0	0	241
#10	Explored:	148	2670	1	26757	364	29714	24150	143	1	83948
	Rejected:	141	642	0	26492	331	29524	24042	106	0	81278
	Performed:	7	1766	1	265	33	190	108	37	1	2408
	Dismissed:	0	262	0	0	0	0	0	0	0	262

impossible and impractical to realize for a stochastic optimization algorithm which executes millions of perturbations to find a solution. Another concession is that SKILL is just a scripting language, whereas a potential implementation in a compilation-based programming language might reduce the runtime significantly. Nevertheless, this of course does not belittle the merit of the overall methodology.

8.1.3 Examples of Nonterminating Interaction Cycles

While the previous examples illustrated positive effects of emergent behavior in SWARM, there can unfortunately also be futile occurrences of emergence. Most intriguing, under certain circumstances it can happen that the interaction runs into a sequence of periodically recurring situations, which –unbeknownst to the interacting participants– represents a fatal infinity loop that will never terminate without intervention from outside.

An example for such a *nonterminating interaction cycle* is given in Figure 8.4, beginning with an initial constellation of nine participants as depicted in image (a). All participants are centered within their free peripheral space – except P_1, which perceives its free peripheral space S_{P_1} (limited by P_3, P_4, P_5 and P_8) and determines the center of S_{P_1} as its new location. After performing its *Centering* move (b), P_1 is still clear even though P_2 had been in a *blind spot* of P_1's free peripheral space. But due to that move, the free peripheral spaces of P_2, P_3 and P_4 change (whereas the other participants remain unaffected). Therefore, P_2, P_3 and P_4 also perform *Centering* moves, leading to the new situation shown in (c). By now, the free peripheral space of P_1 has changed –in particular because its western corridor is not limited by P_8 anymore but by P_2– and P_1 once again centers itself (d), which in turn entails small *Centering* movements of P_2, P_3, and P_4 (e). Because of P_1's move, the obstacle limiting its free peripheral space to the north is now P_6 instead of P_5, causing P_1 to make a *Centering* leap downwards and slightly to the right (f). While this does not alter the free peripheral space of P_4, it provokes *Centering* actions of P_2 and P_3 (g). By then, the westwards obstacle for P_1's free peripheral space is P_8 once again, making P_1 walk to the left by performing yet another *Centering* (h). This is followed by *Centering* moves of P_3 and P_4 (i), which results in a situation that is identical to the initial constellation (a). Since moving from a clear location to another clear location –as was the case with all *Centering* actions above– does not consider aversions and wounds (thus eluding a memory effect in examples like this), the participants will definitely repeat the described succession of movements once again, and again and again without ever coming to an end.

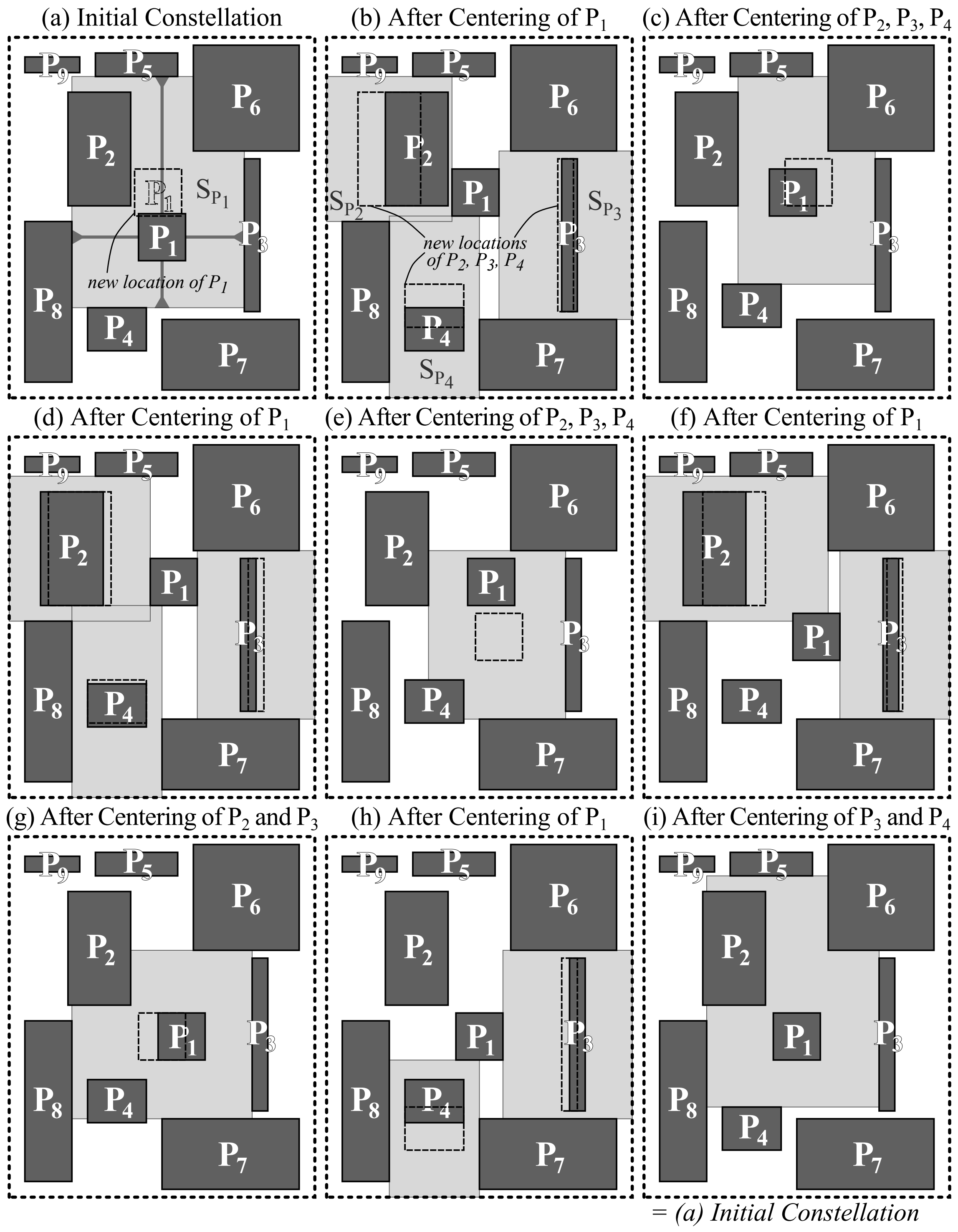

(a) Initial Constellation
(b) After Centering of P_1
(c) After Centering of P_2, P_3, P_4
(d) After Centering of P_1
(e) After Centering of P_2, P_3, P_4
(f) After Centering of P_1
(g) After Centering of P_2 and P_3
(h) After Centering of P_1
(i) After Centering of P_3 and P_4
P_1
P_2
P_3
P_4
P_5
P_6
P_7
P_8
P_9
S_{P_1}
S_{P_2}
S_{P_3}
S_{P_4}
new location of P_1
new locations of P_2, P_3, P_4
= (a) Initial Constellation

Figure 8.4: Example of a nonterminating interaction cycle: a rotatory circulation sequence.

Such an outcome of perpetual interaction (which is somehow reminiscent of a draw in chess due to *repetition of position*) cannot be detected from the limited viewpoint of a single participant but only by an outside observer who surveys the entire problem as a whole. The possibility to get lost in an infinity loop like this represents a tragic but natural weakness of decentralized systems that can also be encountered in reality. An enthralling example is given by a so-called *ant mill* as first reported in [356]. Although the pheromone-guided trailing habit of social insects helps an ant colony in accessing food resources (see Section 6.3.2), a self-intersection of the pheromone trail can delude the leading ants into tracing the tail of the foraging party. This leads to the formation of a rotating circular column (illustrated in Figure 8.5 [357]) which may go on forever, i.e., until the ants die of exhaustion.

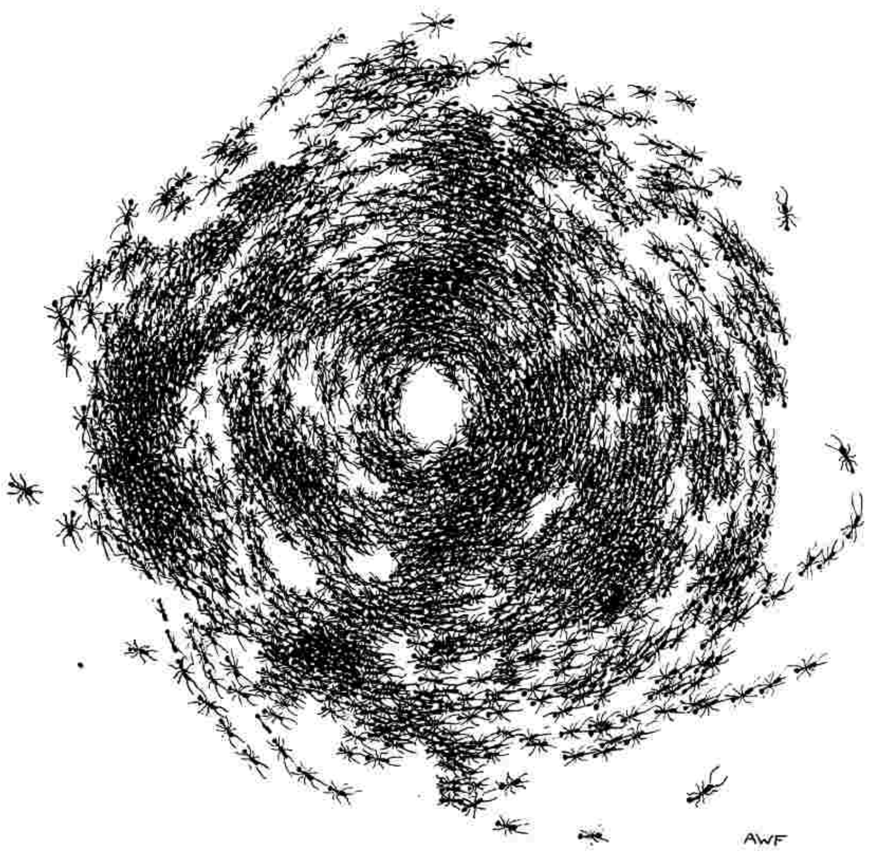


Figure 8.5: Circular column of an ant mill, as drawn according to a photograph (source: [357]).

Equivalent to an ant mill, the periodic interaction sequence of Figure 8.4 can be regarded as a cycle of rotatory *circulation* (with participant P_1 truly going round and round in endless circles). Another kind of nonterminating interaction cycle that can also be observed to emerge in SWARM is an *oscillation*. The simplest form is a two-period oscillation, where the movements in one round of interaction are entirely reversed in the subsequent round.

To visualize such an oscillation sequence, Figure 8.6 provides a plain example with seven participants. In the initial constellation (a), all participants except P_1 are centered within their free peripheral space. Naturally, P_1 also performs a *Centering* move leading to the situation shown in image (b). Thereby, the free peripheral space of P_2 gets trimmed, making the participant wander to the right via another *Centering* (c). In this fashion, the free peripheral space of P_3 is also cut, provoking a *Centering* move downwards (d). The *domino effect* continues to propagate through P_4 (e) and P_5 (f) while participants P_6 and P_7 are not bothered. So, the situation shown in (f) represents the initial constellation for the next round of interaction (g). Since P_5 moved out of P_1's southern corridor with its previous move, the free peripheral space of P_1 now stretches down to the zone outline and lets it step back into its initial location (h). Thereby, the free peripheral space of P_2 analogously gets reverted to its original size, sucking P_2 back into its previous location (i). The same thing happens to P_3 (j) and this chain reaction proceeds through P_4 (k) as well as P_5 (l). Like before, P_6 and P_7 stay where they are so in the end, the participants' final arrangement –as can be seen in (l)– is equal to the initial constellation (a).

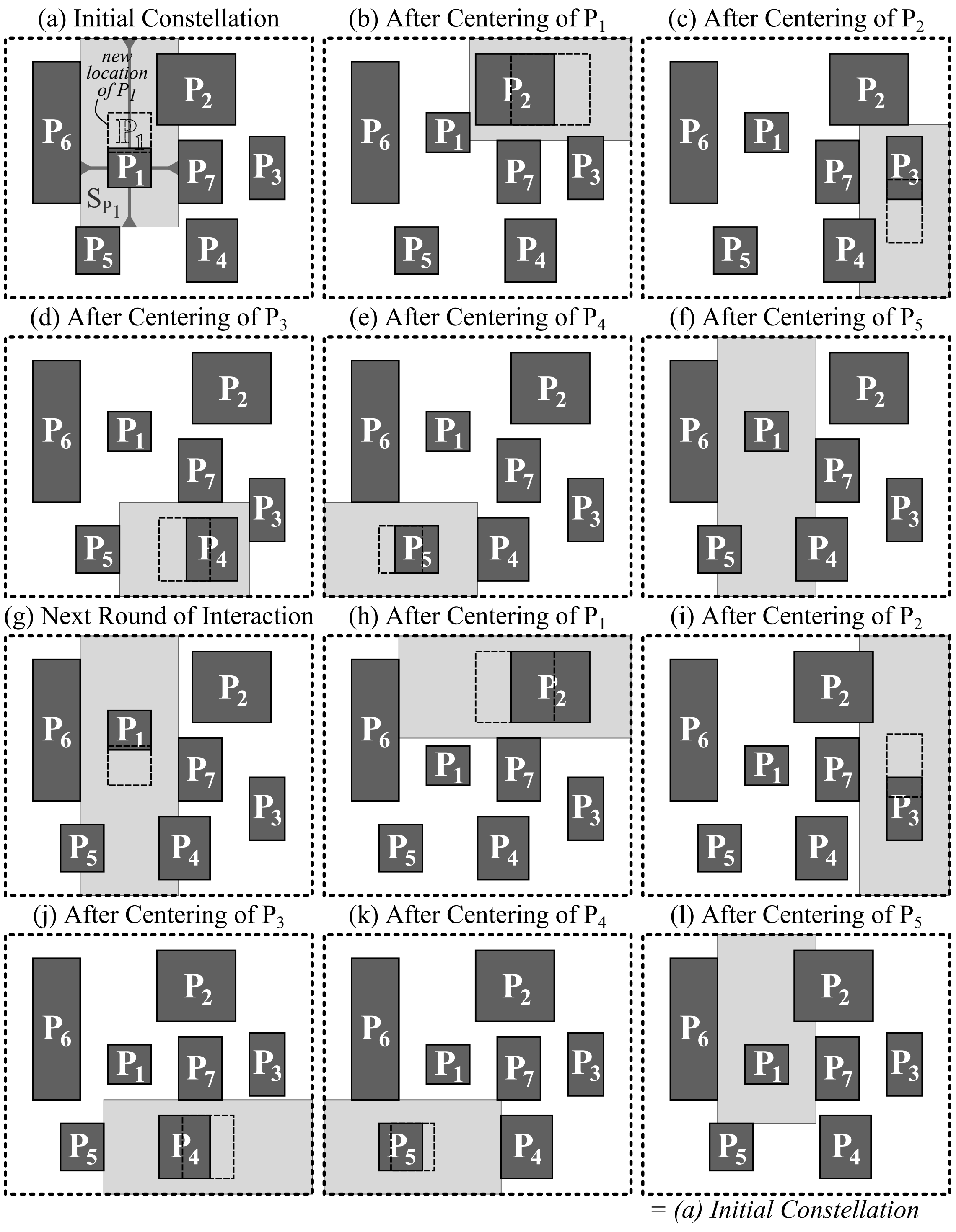

(a) Initial Constellation
(b) After Centering of P_1
(c) After Centering of P_2
(d) After Centering of P_3
(e) After Centering of P_4
(f) After Centering of P_5
(g) Next Round of Interaction
(h) After Centering of P_1
(i) After Centering of P_2
(j) After Centering of P_3
(k) After Centering of P_4
(l) After Centering of P_5
new location of P_1
S_{P_1}
P_1
P_2
P_3
P_4
P_5
P_6
P_7
= (a) Initial Constellation

Figure 8.6: Example of a nonterminating interaction cycle: a two-period oscillation sequence.

From then on, similar to the circulation example of Figure 8.4, the participants in Figure 8.6 will repeat the illustrated sequence of actions without ever settling in a definite location. But in contrast to the circular rotation in the former example, the interaction here rather represents an oscillation wave, consisting of a round with "forward" movements followed by a round of countermanding "backward" movements. Figuratively, the participants swing forth and back like electrical energy oscillates between the inductor and the capacitor in an LC resonant circuit.

Another parallel can be drawn between SWARM and cellular automata once again, since Conway's Game of Life also knows oscillators, i.e., periodic patterns which repeat themselves on the spot ad infinitum like the oscillation cycle in Figure 8.6 can be seen to do. Furthermore, moving patterns such as gliders and spaceships can be understood as the automaton's counterpart to the perpetual interaction in Figure 8.4 (although the pattern's trajectory is a straight line instead of a circular path). Completing the picture, a settlement in SWARM corresponds to a still life in the cellular automaton. This whole analogy fits snugly, but it also points to an important difference as follows.

A settlement in SWARM is stable in a static sense. A nonterminating interaction cycle is obviously not stable in that sense, no more than it is chaotic by any means. Yet, it would also be wrong to call such a cycle *metastable* in the sense of the *edge of chaos* (Section 6.4.3) since it is definitely not short-lived – the opposite is the case. From that perspective, one can say that a steady circulation or oscillation as exemplified above is also stable, not in a static sense but in a dynamic fashion. Of course, certain interaction dynamics are required in SWARM to reach a state of static stability after each tightening: a settlement which is tighter (i.e., of higher value) than the previous one. These dynamics ask for metastability – neither lazing in lower-level order, nor drifting into chaos and instability, nor (never-)ending in a periodic sequence of dynamic stability.

Now, to explicate the mentioned difference in the established analogy, one can contend that in Conway's Game of Life, the greatest fascination –at least concerning the visual spectacle– probably emanates from dynamically stable structures (such as gliders and spaceships) as well as the so-called methuselahs (i.e., highly chaotic patterns which require a large amount of transitions to reach a stable state or dissolve into nothing). In contrast, the success of SWARM depends on producing still lifes (above all: the ultimate settlement) whereas dynamically stable infinity loops and completely instable devolutions into utter chaos need to be avoided.

8.2 Practical Floorplanning Examples

Even though SWARM has not been primarily developed for targeting floorplanning problems, it can be readily employed for that purpose. This is demonstrated by the following two examples, the first one (Section 8.2.1) again featuring a rectangular outline like the examples before, the second one (Section 8.2.2) showcasing SWARM's ability to consider nonrectangular –i.e., rectilinear– outlines. An assessment of SWARM with respect to floorplanning tasks will then be given in Section 8.2.3.

8.2.1 Floorplanning Example with Rectangular Outline

Figure 8.7 depicts a rectangular floorplanning problem with 15 circuit blocks of various sizes. As is customary in floorplanning (see Table 2.2 on page 21), these blocks are treated as black boxes (here displayed as plain rectangles) whose aspect ratios have yet to be determined. In line with the explication of Section 7.3.1.2, the distance between two blocks is modeled as a straight *connection* between the centerpoints of the blocks. In this example, all connections are equally emphasized (i.e., each connection has an *emphasis* of $e = 1$). In the initial constellation, only 2 out of the 21 connections are *relaxed* (drawn as solid lines) while 19 connections are *unrelaxed* (drawn as patterned lines), as illustrated in image (a).

The distance between a block and an external component (e.g., a bondpad) is represented as a line which ends on the outline of the layout zone. It is handled like a block-to-block connection except that this endpoint automatically moves along the zone outline during the interaction, such that the distance is always kept as short as possible. The blocks themselves are implemented as *elastic* participants, exploiting the entire pool of elastic deformation behaviors covered in Section 7.3.3.3, to consider *full variability* during their action exploration.

The final constellation (b) reveals which aspect ratios the 15 floorplan blocks have assumed at the end of their self-organization. The resulting outcome is highly compact as the amount of dead space only makes up 2.9% of the layout zone area. Furthermore, all connections are relaxed at last: as can be seen in the final settlement, each pair of connected blocks is located in direct vicinity of each other and

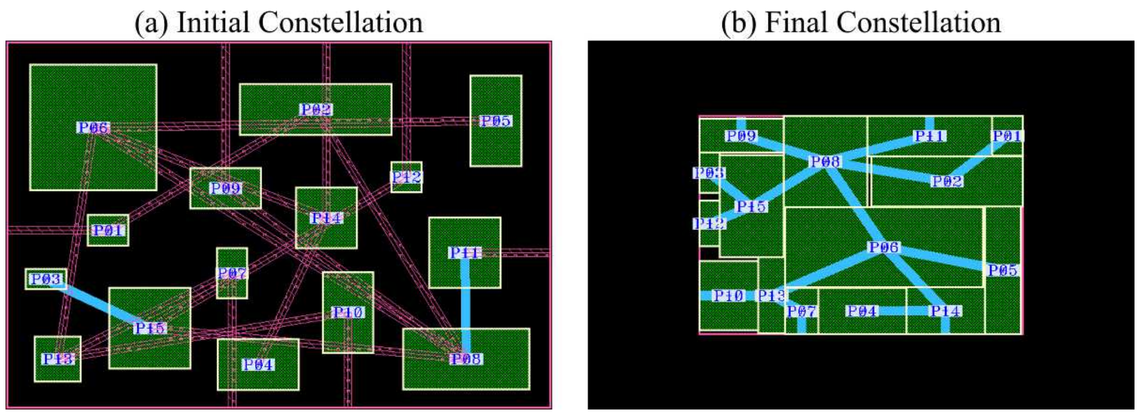


Figure 8.7: Example of applying SWARM to a floorplanning problem with a rectangular outline.

every block that has a connection to an external component lies next to the zone boundary. The figurative icing on the cake in this example is that the self-organization has led to the emergence of an arrangement wherein no connection lines cross one another.

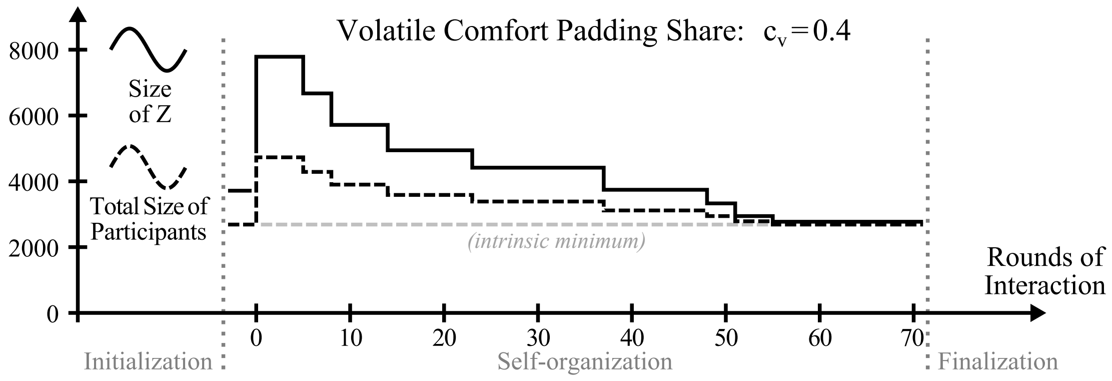


Figure 8.8: Tightening profile for the SWARM run of the floorplanning example from Figure 8.7.

To obtain this solution, the participants required a total of 71 interaction rounds, spread across 9 tightening-settlement cycles (interaction rounds per cycle: 5, 3, 6, 9, 14, 11, 3, 4, 16). An *abrasive tightening* policy and *volatile comfort padding* (with a volatile comfort padding share of $c_v = 0.4$) were used in this example. This can also be recognized in the tightening profile of the SWARM run, given in Figure 8.8, which shows how the size of the layout zone as well as the total area of the padded participants approach the *intrinsic minimum* (detailed numbers being listed in Table 8.5). Furthermore, the graph exhibits the three characteristic episodes of a typical SWARM run, except that the last settlement peaks out with a large number of interaction rounds (which is not unusual in SWARM runs where the layout zone becomes extremely tight in the end, as it is the case in this example).

Table 8.5: Area decline in the SWARM run of the floorplanning example from Figure 8.7.

	Area per Tightening-Settlement Cycle								
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Layout Zone:	7786.4	6674.3	5710.0	4939.6	4420.3	3749.8	3329.8	2945.3	2770.3
Participants:	4728.1	4284.0	3897.8	3589.5	3383.6	3114.0	2946.5	2791.8	2689.4
Dead Space:	39.3%	35.8%	31.7%	27.3%	23.5%	17.0%	11.5%	5.2%	2.9%

8.2.2 Floorplanning Example with Nonrectangular Outline

In Figure 8.9, SWARM is applied to a floorplanning problem where the layout zone has a nonrectangular outline. Such a contour can be encountered in cases where the desired floorplan only represents a part of the entire chip, using the notch to preserve layout space for other circuit components or IC sections. The 16 blocks feature 30 connections, and –in contrast to the previous example– the connections have been assigned different emphasis values (between $e = 0.4$ and $e = 1.0$), in the images indicated via the widths of the respective connection lines. As can be seen in image (a), 9 connections in the initial constellation are relaxed and the remaining 21 connections are unrelaxed.

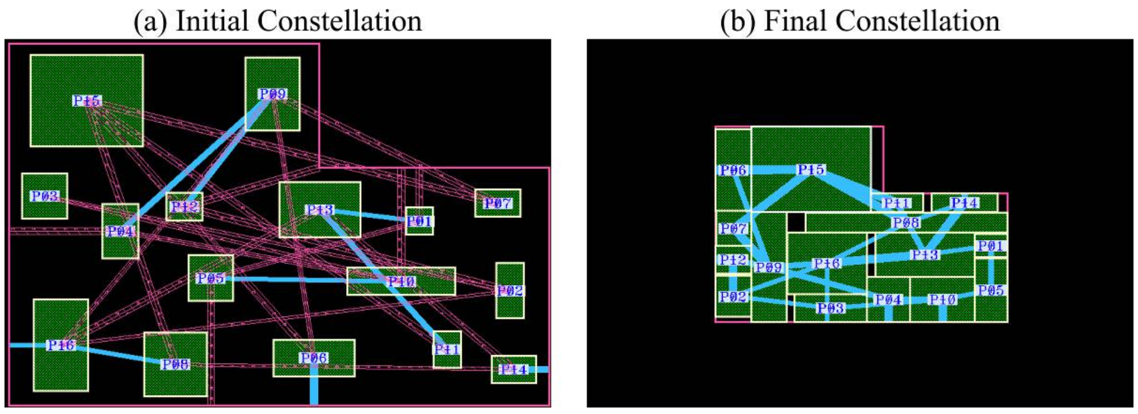


Figure 8.9: Example of applying SWARM to a floorplanning problem with a nonrectangular outline.

With a total of 95 interaction rounds throughout 8 tightening-settlement cycles, the participants are able to obtain the final constellation shown in (b) where the floorplan blocks can again be seen to have changed their aspect ratios. In this outcome, the dead space amounts to 5.7% of the layout zone and –as in the example of Figure 8.7– all connections are finally relaxed. And again, also reminiscent of the example before, tracking the number of interaction rounds per tightening-settlement cycle (5, 3, 10, 5, 9, 19, 12, 32) bares two maxima: an intermediate peak for the 6th settlement and a final peak for the 8th settlement. With an abrasive tightening policy and a volatile comfort padding share of $c_\text{v} = 0.4$, Table 8.6 displays the area decline observed during the SWARM run.

Table 8.6: Area decline in the SWARM run of the floorplanning example from Figure 8.9.

	Area per Tightening-Settlement Cycle							
	1st	2nd	3rd	4th	5th	6th	7th	8th
Layout Zone:	8183.1	6509.5	5130.4	3892.4	3272.3	2823.3	2497.6	2387.7
Participants:	4624.0	3955.7	3403.4	2908.2	2659.5	2479.7	2351.0	2252.2
Dead Space:	43.5%	39.2%	33.7%	25.3%	18.7%	12.2%	5.9%	5.7%

8.2.3 Assessment

Although SWARM has not been conceived with the task of floorplanning in mind initially, the examples of Section 8.2.1 and Section 8.2.2 demonstrate that the self-organization approach is quite adequate for addressing this application. Moreover, a look at Table 3.1 on page 39 reveals that the SWARM methodology even surpasses existing floorplanners in terms of five decisive aspects, as SWARM apparently is the first floorplanning approach that

(1) takes both area and block distances into account,
(2) pays respect to a rectilinear layout outline,
(3) considers fully variable block dimensions,

(4) supports nonslicing floorplan structures,
(5) and works completely deterministic.

The first three items are visibly evidenced by the examples above while the fifth issue is a logical consequence of SWARM's credo not to involve any kind of randomization. Regarding aspect (4), it is interesting to see that both the outcome in Figure 8.7 (b) as well as the final constellation of Figure 8.9 (b) represent slicing floorplans. Yet, it is important to realize that this predicate has not been explicitly imposed as a design-methodical restriction, but that it simply emerged from the interaction as an "energetically favorable state", so to speak. In other words, such slicing structures seem to be natural *attractors* of the system – but as the two examples published in [358] underline, it is generally possible that nonslicing solutions can result from a SWARM run.

Some particular floorplanning issues are not yet handled by SWARM. For instance, SWARM does not set pin positions on the floorplan blocks and is not intent on optimizing the power supply or the current flow. Inhowfar it is possible to incorporate such aspects into SWARM, e.g., via dedicated design constraints, may be subject to future works. Another common demand is to separate certain blocks from each other such that sensitive circuitry is not disturbed by noisy IC sections. In further developments of SWARM, this could be achieved by modeling compressive stress (instead of *tension*) into a connection, thereby reversing its effect on the interaction.

8.3 Practical Place-and-Route Examples

To demonstrate SWARM being employed for practical place-and-route purposes, the following Section 8.3.1 first elaborates on how the methodology has been integrated into the SDL design flow from the perspective of usage. Then, SWARM is shown creating layouts of various aspect ratios for a Symmetric P-Input OTA (Section 8.3.2) and a Folded Cascode P-Input OTA (Section 8.3.3). Finally, Section 8.3.4 assesses the SWARM methodology with regard to such place-and-route problems.

8.3.1 Usage of SWARM in the Design Flow

To facilitate the usage of SWARM in the design flow, it has been worked into the so-called *Constraint-Administered PCell-Applying Blocklevel Layout Engine* (CAPABLE) presented in [359]. CAPABLE is a designer-oriented framework that can be used to combine various automatisms into an executable layout automation script. While this concept is not restricted to including either algorithmic or procedural automatisms, CAPABLE is predominantly designated to realize a generator approach. In particular, the engine aims at applying *context aware* PCells to the layout in a hierarchical fashion (as it is intended for the *governing modules* in SWARM). Hence, a CAPABLE script represents a command sequence that operates similar to a procedural generator, but on a higher level of abstraction.

CAPABLE is constraint-administered insofar as it provides a special *constraint interpreter* by which certain parts of the layout automation script can be automatically created from formally expressed design constraints that have been assigned to the input schematic by the designer. For example, a custom CurrMirr constraint –assigned to a group of transistors that are meant to constitute a Current Mirror circuit– is turned into a series of script commands that consecutively impose a Positioning module, a Wiring module, and an Info module on the devices (thus establishing a Current Mirror module association as depicted in Figure 7.8 on page 105). In this manner, high-level design requirements can be *explicitly* formulated as constraints, while the respective low-level implications are *implicitly* taken into account by the automatisms that implement these constraints.

Taking the point of critique to heart that analog layout automatisms often find only little acceptance due to a lack of insight into their inner workings, CAPABLE comes with a convenient GUI that allows to execute the developed layout scripts in a transparently traceable way. A graphical *execution cockpit* provides the user with different pacing modes – from running the entire script all at once to performing each command call in a fine-grained fashion, even separating different steps that the command may be comprised of. For a governing module, each individual operation of the *adoption process* described in Section 7.2.2.1 can be separately carried out as a single step. Utilizing this facility, the events occurring in the self-organization of a SWARM run can be recorded as a script and later be replayed step by step via CAPABLE.

The interface between a CAPABLE script and the design environment Virtuoso is made up by a small set of API commands referred to as *Constraint-Derived Procedural Commands* (CDPC). It is written in SKILL and can be considered as a domain-specific language for layout automation. Since CDPC is primarily focused on the application of PCells, the central CDPC command is `PCell` and serves the instantiation of a context aware PCell according to the adoption process mentioned above. Another CDPC command is named `Group` and allows to store a set of layout components in a single internal group (which may be helpful if multiple PCells are to be imposed on the same set of components, as in a Current Mirror module association). To give an example for a non-PCell automatism, the `Modgen` command can be used to invoke the interdigitation feature of Virtuoso's own ModGen tool [146]. Currently, the following commands are supported:

`Cellview`
: Opens the specified target layout (or creates an empty one if none yet exists).

`Revert`
: Discards any modifications that have been made to the layout since its last save.

`Clear`
: Removes all objects (instances, shapes, labels, markers, etc.) from the layout.

`GenFromSource`
: Invokes the design environment's native *Generate from Source* functionality.

`SetVar`
: Stores a given value in a specified local variable.

`GetVar`
: Retrieves the value of a specified local variable.

`Group`
: Assigns a set of layout components to an internal group.

`PCell`
: Instantiates a context aware PCell following the adoption process of Section 7.2.2.1.

`Modgen`
: Executes the interdigitation function of Virtuoso's proprietary ModGen tool.

`Rotate`
: Rotates a layout component (plus its adopted children and associated modules).

`Move`
: Moves a layout component (plus its adopted children and associated modules).

`PlacePin`
: Puts a generated layout pin to a given location and optionally sets its layer.

`Flatten`
: Flattens a layout component (as well as its adopted children) hierarchically.

`Geo`
: Uses the PCell Designer's *geometry query language* to determine layout geometries.

`Replay`
: Performs an event that was recorded during the self-organization of a SWARM run.

Calling the `PCell` command, a dedicated Wiring PCell can be instantiated in the layout. Such a PCell instance draws a single wire, comprised of different wire segments, between two component terminals. Together with the `Geo` command for retrieving the location of a component terminal, this enables the implementation of procedural routing scripts. Referring to the explanation in [360], a PCell parameter allows to specify the wire as a sequence of points with the possibility to set the metal layer and wire width for each individual segment. Although the points may be given in absolute coordinates, they are rather meant to be defined in relation to the bounding boxes of existing components in the layout so the wire can smoothly run along the edges of obstructive modules.

Such a routing approach can be quite appropriate if the overall module arrangement in the layout is already known in advance. This will be the case in the subsequent two examples and thus motivates the preconception of a procedural routing scheme for connecting the governing modules with each other (while the connectivity inside each module is still taken care of by the respective PCells). Therein, the need to preserve sufficient layout space between the modules is where the idea of *solid comfort padding* comes into play. On that basis, going through the three phases of a SWARM run proceeds as follows:

(1) The *initialization phase* is carried out by a sequence of CDPC commands in the execution cockpit of CAPABLE. The corpus of this command sequence can be automatically derived from the constraints assigned to the input schematic via Virtuoso's constraint management system [24]. Custom edits and extensions to the command sequence are possible through manual scripting in a plain text editor.

(2) Switching to the SWARM tab of CAPABLE's user-interface, the user gains control over the *self-organization phase*. Similar to the different pacing modes in CAPABLE, the events may be driven by (a) stimulating a single participant, (b) initiating an entire round of interaction, (c) conducting a whole tightening-settlement cycle, or (d) making SWARM perform the complete self-organization until reaching the desired layout zone size.

(3) For the *finalization phase*, the user's focus returns to the execution cockpit of CAPABLE. Here, the preconceived routing scheme can be applied to the module arrangement of the self-organization's final settlement as a CDPC script. This complements the already existing *intra*-module wiring with the still missing *inter*-module wiring and finishes the layout.

8.3.2 Symmetric P-Input Operational Transconductance Amplifier

Figure 8.10 shows the schematic diagram of a Symmetric P-Input OTA circuit consisting of six functional units: a Differential Pair (1), three Current Mirrors (2 to 4), a Symmetric Current Mirror Pair (5) which in turn consists of two Current Mirrors (5a and 5b), as well as a Blocking Cap (6) to stabilize the supply voltage. By explicitly imposing a custom "Module" constraint (i.e., a DiffPair constraint, a CurrMirr constraint, etc.) on each of these functional units, SWARM is told to manage the corresponding layout devices with a governing module, module association, or hierarchical module association. Thereat, a user-definable interdigitation pattern can be provided for each Current Mirror (to be put into effect by the respective modules) for the benefit of a good device matching.

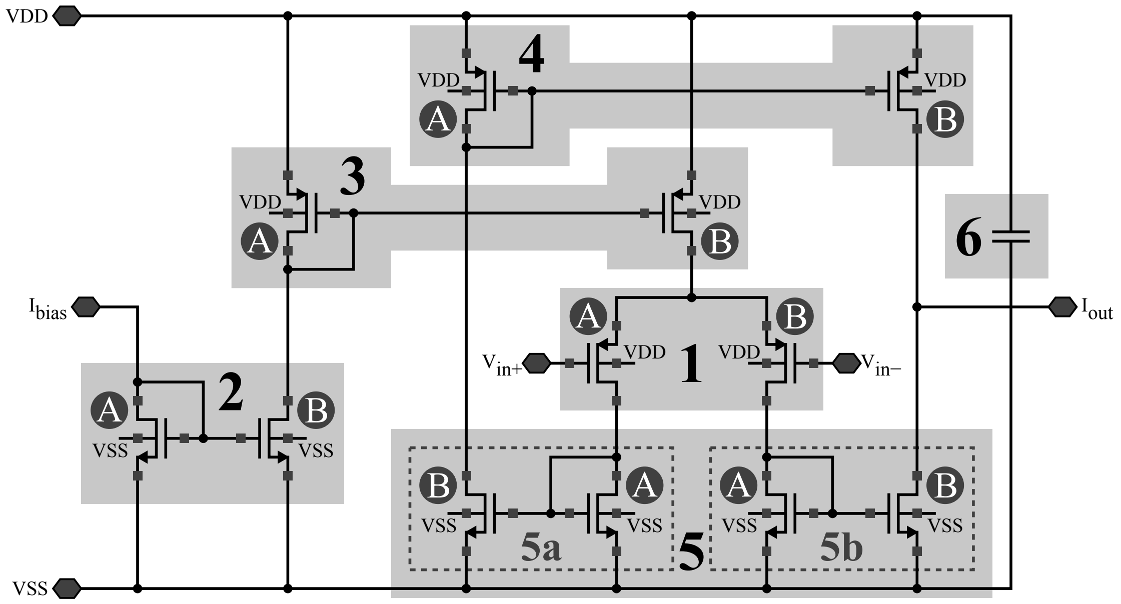


Figure 8.10: Schematic diagram of the Symmetric P-Input OTA circuit addressed with SWARM.

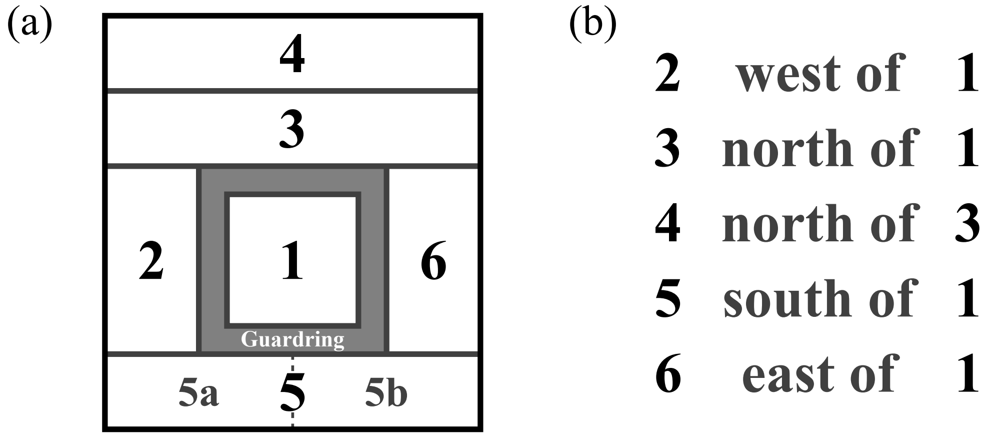
Based on the interdigitation pattern, each module implicitly takes care of its inherent matching requirements on device level. To achieve overall symmetry, the modules then are to be arranged according to a predefined *placement template* as visualized in Figure 8.11 (a). For the module interaction in

SWARM, this sought-after arrangement is explicitly articulated via Relative Placement constraints (b). A constraint of this type –also appearing in [63]– forces a component to lie north, east, south, or west of another component. By implementing a *verification function* covering these four cases, the Relative Placement constraint is checked during the self-organization phase to avoid *noncompliance* whenever one of the involved modules explores its potential actions.

As an example for the circuit at hand, the input Current Mirror (2) is supposed to be located west of the Differential Pair (1). In this case, the verification function $v(P, P')$ –with the Current Mirror and the Differential Pair being assigned to P and P' respectively– looks like this:

$$v(P,P') \;=\; \begin{cases} 1 & \Leftrightarrow \;\; \frac{1}{2}\big(\vdash(\square P) + \dashv(\square P)\big) < \frac{1}{2}\big(\vdash(\square P') + \dashv(\square P')\big) \\ 0 & \Leftrightarrow \;\; \text{otherwise.} \end{cases} \tag{8.1}$$

The condition in this formula is less stringent than the one in [63] as it does not forbid an overlap of P and P'. This is deliberate: on the one hand, the condition is as lax as possible to temporarily permit situations of interference (which can be vital after a zone tightening to let prone modules become safe again); on the other hand, the condition is as strict as neccessary to effectively influence the modules in their attempt to become overlap-free at the end of each settlement. In other words, it is the combined power of two innate *desires* (formulated in Section 7.3.1.1 and Section 7.3.1.5 respectively) making sure that the modules are finally clear and constraint-compliant.



2 west of 1
3 north of 1
4 north of 3
5 south of 1
6 east of 1

Figure 8.11: (a) Placement template and (b) respective constraints for the Symmetric P-Input OTA.

Table 8.7 gives an overview of the circuit's devices according to the module by which they are managed. Also listed are the respective interdigitation patterns and the *ductility* (number of possible deformation variants, as introduced in Section 7.2.4.4) of the module. As can be seen, the Blocking Cap (6) is handled with full variability (managed by a *variator module* as mentioned in Section 7.2.4.3). This is done to achieve that the Differential Pair is neatly placed in the middle. Paying heed to an Imitation constraint to enforce the action of the same name (see Section 7.3.3.2), the capacitor imitates the transformations of the input Current Mirror (2) on the other side of the Differential Pair (1), thereby permanently fitting its aspect ratio to that of the Current Mirror.

To protect the Differential Pair from signal disturbances, it is to be enwrapped by a guardring (also indicated in Figure 8.11). This is accomplished with a Guardring constraint, telling SWARM to place a dedicated Guardring PCell aroung the Differential Pair module. Whenever the Differential Pair deforms itself during the module interaction, the Guardring automatically re-*absorbs* the Differential Pair and *adapts* itself to the new dimensions. Remember that *absorption* and *adaptation* represent the first two operations of SWARM's *adoption process* (Section 7.2.2.1).

Now, the initialization phase starts with a *Generate from Source* (called via the `GenFromSource` command), in this case leading to the picture of Figure 8.12 (a) by producing a total of 43 primitive devices. Next, (based on the `PCell` command) the governing modules –implemented with Cadence PCell Designer [126]– are imposed on the respective groups of devices and then (using the `Modgen` command) put into the initial constellation shown in image (b). This arrangement, which already follows

Table 8.7: Overview of the functional units constituting the Symmetric P-Input OTA circuit.

In-dex	Governing Module / Module Association	Device Type	Num. of Devices	Interdigitation Pattern	Ductility (Variants)
1	Differential Pair	PMOS	4	A B / B A (fix)	40
2	Current Mirror	NMOS	4	A B B A	56
3	Current Mirror	PMOS	6	B A B B A B	56
4	Current Mirror	PMOS	12	A B A B A B B A B A B A	56
5	Symm. Current Mirror Pair	-	-	-	112
└**5a**	Current Mirror	NMOS	8	A B A B B A B A	-
└**5b**	Current Mirror	NMOS	8	A B A B B A B A	-
6	Variator Module	Capacitor	1	-	full variability

the template of Figure 8.11, ends the initialization phase (in line with the depiction of Figure 7.3 on page 99) and represents the starting point for the self-organization – irrespective of the layout zone's aspect ratio of which various ones are covered in the subsequent examples of this Section 8.3.2.

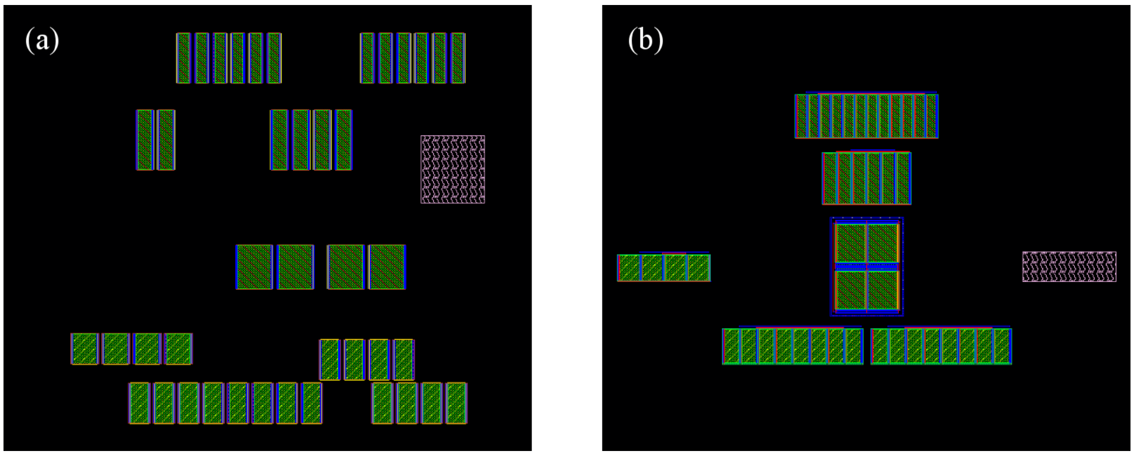


Figure 8.12: OTA layout (a) after device generation and (b) at the end of the initialization phase.

For a width-to-height aspect ratio of 2:3, the layout obtained by SWARM after going through the self-organization and the finalization phase is displayed in Figure 8.13. As can be seen, every module except the Differential Pair has deformed itself into a layout variant different from the one in the initial constellation of Figure 8.12 (b). A very fortunate advantage of a template-based arrangement as pursued here is that the locations of the modules in relation to each other do not change throughout the entire interaction. This allows to omit the exploration of *Swapping* and *Pairing* actions, which reduces the runtime tremendously (since these kinds of actions are computationally the most expensive ones, as mentioned in the context of Table 8.4).

For that reason, the SWARM run in this case just required a total of 22 interaction rounds throughout 5 tightening-settlement cycles. All in all, only 77 out of 1389 explored actions had to be performed to obtain the illustrated outcome, taking a runtime of 60 seconds for the mere self-organization phase. The observation that the quotient between this runtime and the number of performed or explored actions is larger than in the SWARM runs of Table 8.3 is simply rooted in the fact that –in contrast to the symbolic modules in those examples– a module transformation here is more elaborate because it has to include the corresponding co-transformations of all adopted children and associated modules (as explained in Section 7.2.2.2 and Section 7.2.3.3).

To demonstrate the immense layout flexibility achievable with SWARM, it been applied to the Symmetric P-Input OTA circuit of Figure 8.10 for a total of seven different aspect ratios. Table 8.8 displays the cardinal statistics for the respective SWARM runs, including the rounds of interaction per settlement, the number of explored, rejected, performed, and dismissed actions, as well as the number of performed

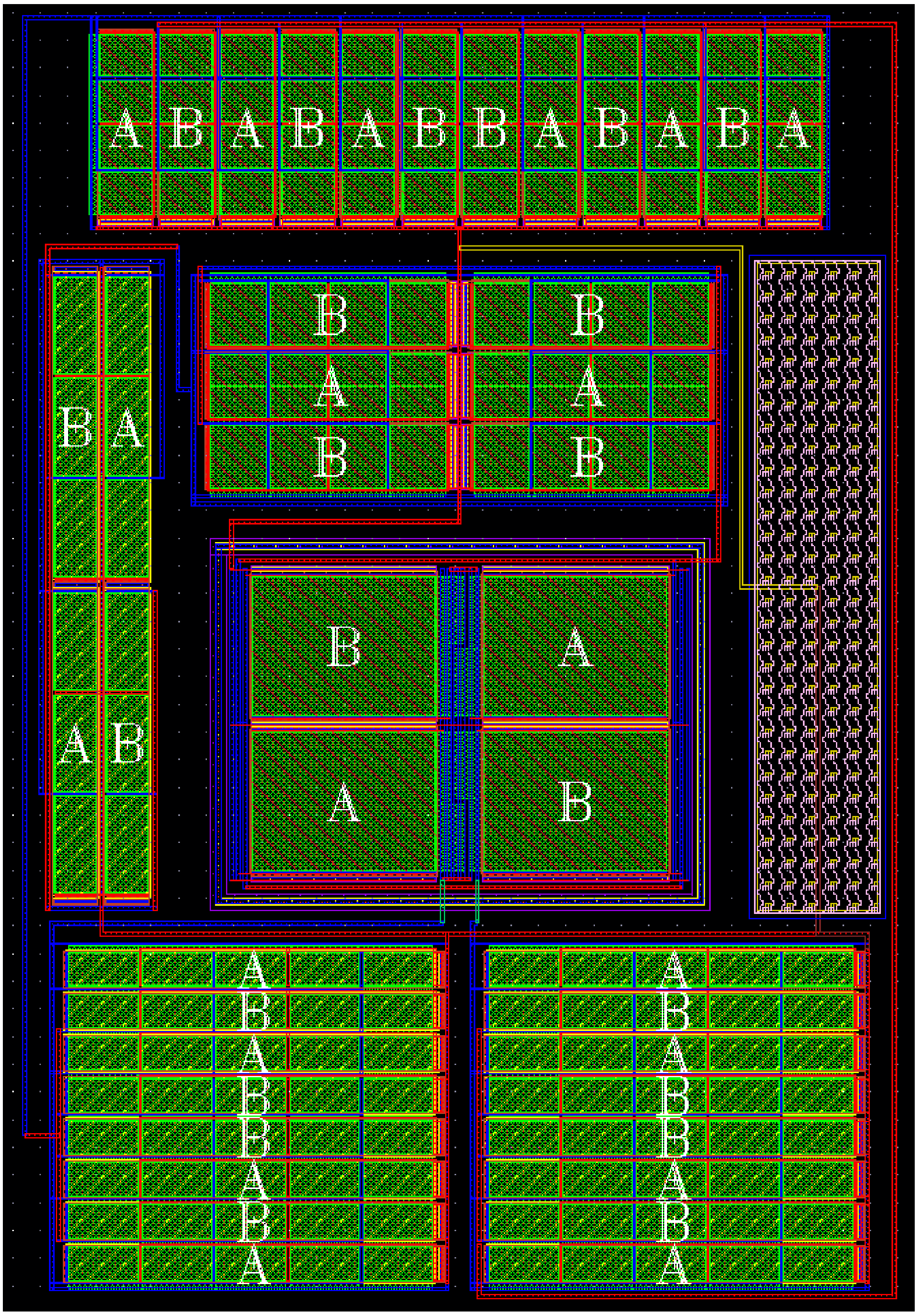


Figure 8.13: Finalized layout of the Symmetric OTA, obtained by SWARM for a 2:3 aspect ratio.

actions distinguished by the kind of action, and the runtime required for the entire run (covering all three phases, i.e., initialization, self-organization, and finalization).

Table 8.8: Statistics of the different SWARM runs applied to the Symmetric OTA. The abbreviated kinds of actions are: Re-entering (R), Centering (C), Lingering (L), Budging (B), Hustling (H), Evasion (E), Yielding (Y), and Imitation (I).

Asp.	**Rounds per**	**Total**	**Number of Actions**				**Perf. Actions (By Kind)**								**Total**
Ratio	**Settlement**	**Rounds**	expl.	reje.	perf.	dism.	R	C	L	B	H	E	Y	I	**Runtime**
1:1	4, 5, 3, 8	20	1392	1272	71	49	1	47	0	7	1	1	1	13	108 sec.
1:2	4, 4, 3, 4, 10	25	364	214	97	53	0	77	1	0	0	1	0	18	113 sec.
1:3	5, 4, 4, 4, 5, 7	29	791	617	109	65	0	83	1	2	0	1	0	22	93 sec.
2:1	4, 16, 4, 10	34	1803	1599	109	95	2	62	0	6	10	1	1	27	233 sec.
2:3	4, 3, 4, 5, 6	22	1389	1257	77	55	1	55	0	5	0	0	0	16	113 sec.
3:2	4, 4, 6	14	935	851	46	38	1	32	0	3	0	2	0	8	95 sec.
5:2	4, 4, 5, 20	33	2158	1960	140	58	1	82	0	17	1	1	11	27	177 sec.

Taking a look at the rounds of interaction per settlement in Table 8.8 reveals that the self-organization in these runs proceeded quite smoothly on the whole. Even in the least fluent case (the SWARM run with a 5:2 aspect ratio, peaking out with 20 interaction rounds for the final settlement), only 140 actions out of 2158 explored ones had to be performed to obtain a compact and constraint-compliant layout solution in less than three minutes. Apart from this, it is again interesting to examine the number of performed actions per kind of action (understandably, *Swapping* and *Pairing* have been omitted here while *Imitation* was added to the action palette). As in the examples of Table 8.4, the most popular kind of action is *Centering*. But more importantly, it can be seen that every kind of action has been performed in at least two of the seven listed SWARM runs. This finding once more affirms the feasibility of the action catalog implemented by SWARM.

The finalized layouts for the 1:1 aspect ratio and the 5:2 aspect ratio are shown in Figure 8.14 and Figure 8.15 respectively (while the outcomes for the remaining aspect ratios have been moved to the appendix). As becomes evident from these images, the basic arrangement is the same in every case, exploiting the *cumulative variability* of each module to make the overall arrangement fit inside the outline of the layout zone.

Considering this template-based approach, one might argue that the problem at hand could have been easier solved by modeling it as a *slicing tree* and tackling it with a so-called "floorplan sizing" algorithm (such as [29]) based on *shape functions*.[1] However, this would not just have simplified but trivialized the problem. Instead, translating the template (Figure 8.11) into a set of Relative Placement constraints –as explained above– leaves more degrees of freedom: the final constellation may look like Figure 8.16 (a) –i.e., precisely as in the depiction of Figure 8.11– or could alternatively be structured as in any of the other arrangements from (b) to (j).[2]

On the one hand, this is fine since it does not impair the desired overall symmetry. Moreover, on the other hand it is even serviceable because it keeps up a greater solution space. And rightly so, as certain solutions that would have been unattainable otherwise do indeed emerge in SWARM: the layouts of Figure 8.13 and Figure 8.14 correspond to that of Figure 8.16 (b) while the outcome of Figure 8.15 reflects the placement of Figure 8.16 (c).

So, although the overall module arrangement is comparably strictly predefined by the given placement template, SWARM concedes quite some flexibility here. Apart from this, the solution space is above all opened up by the combined variability of the governing modules. Concentrating only on these deformations, the number of mathematically possible layout solutions based on the multiplied module

[1]A shape function describes the dependency between the width and the height of a layout component.

[2]The amount of potential layout arrangements here is given by the number of modules encountered when traversing the template in vertical direction through its center. Then, the amount of potential arrangements is $\frac{1}{2} \cdot n \cdot (n+1)$: the Gaussian sum formula, which defines the sequence of triangular numbers. In this case, there are $\frac{1}{2} \cdot 4 \cdot (4+1) = 10$ arrangement alternatives.

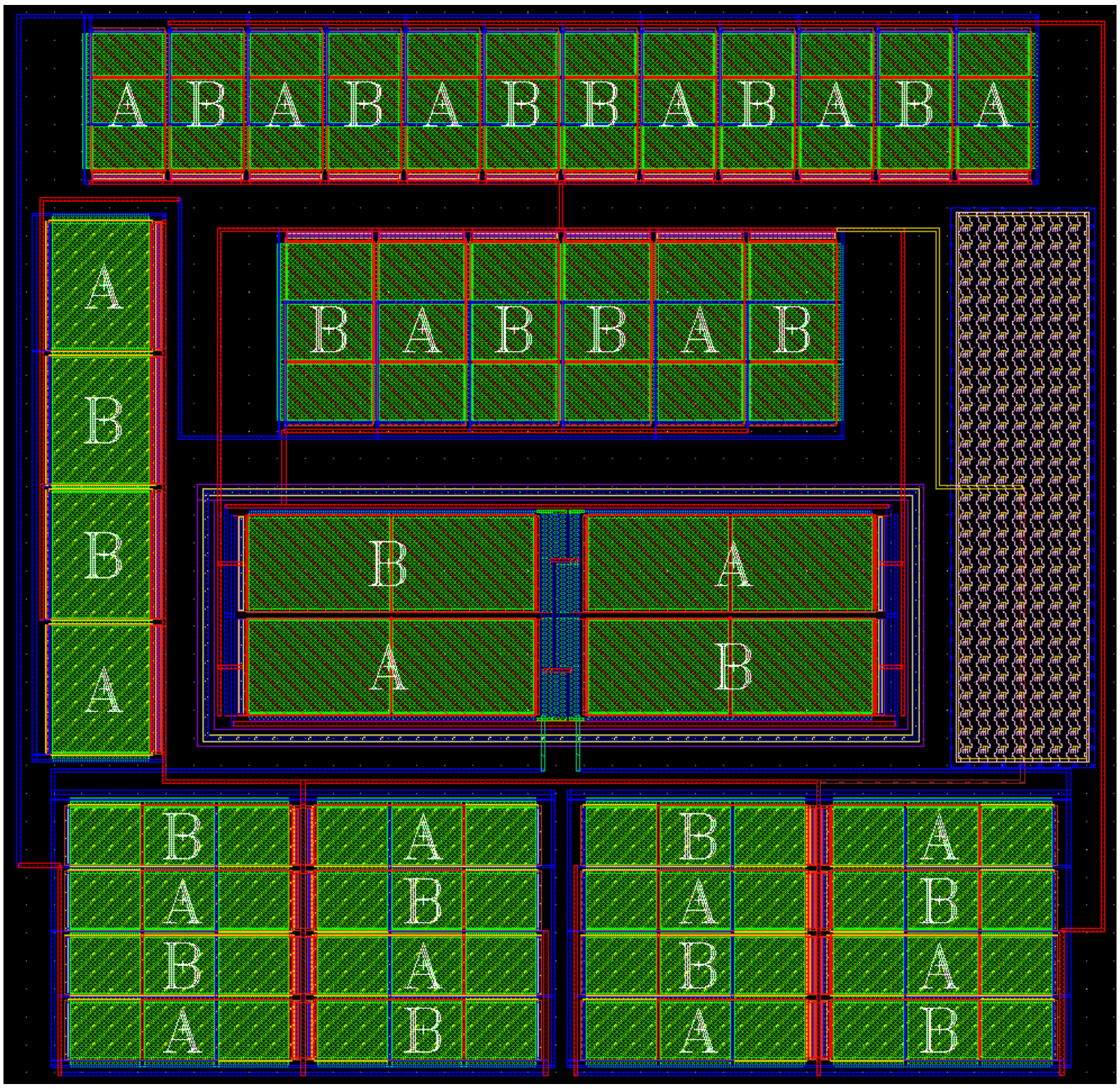


Figure 8.14: Finalized layout of the Symmetric OTA, obtained by SWARM for a 1:1 aspect ratio.

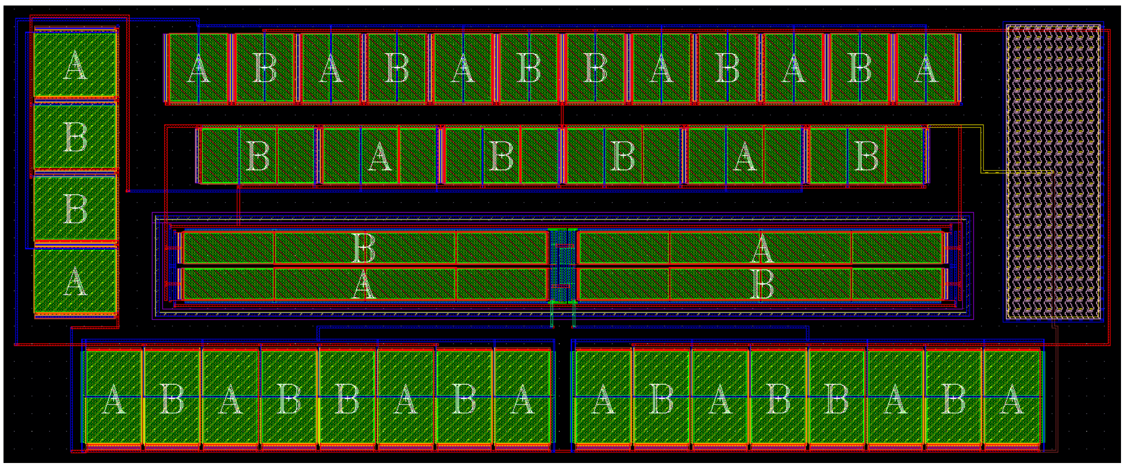


Figure 8.15: Finalized layout of the Symmetric OTA, obtained by SWARM for a 5:2 aspect ratio.

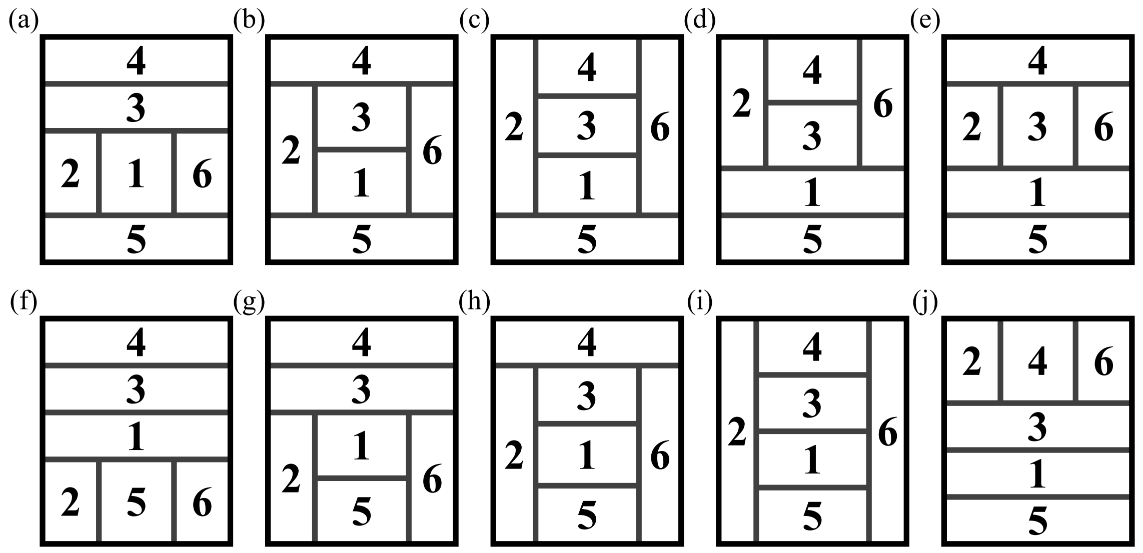


Figure 8.16: Potential layout arrangement alternatives for the Symmetric P-Input OTA circuit.

ductiliy *alone* amounts to a total of

$$40 \cdot 56 \cdot 56 \cdot 56 \cdot 112 = 786\ 759\ 680$$

variants in this example, thus drawing near to a billion potential combinations. Yet (in contrast to a conventional optimization algorithm, for which a larger solution space metaphorically corresponds to a bigger haystack making it more difficult to look for the proverbial needle), the many degrees of freedom are downright helpful for a decentralized approach as pursued here. A valuable conclusion for the topic of this thesis is that formalized design constraints are not necessitated by SWARM to diminish the solution space, but are specifically taken into consideration for their actual purpose: to guarantee the functionality of the layout.

8.3.3 Folded Cascode P-Input Operational Transconductance Amplifier

Applying SWARM to another place-and-route problem, Figure 8.17 presents the schematic diagram of a Folded Cascode P-Input OTA. The six functional units constituting this circuit are: a Differential Pair (1), two simple Current Mirrors (2 and 3, with two and three stages respectively), a Wide-swing Current Mirror (4) consisting of a Bank (4a) and a Cascode (4b), then a three-stage Cascode Current Mirror (5) with a Bank (5a) and a Cascode (5b), and finally also a Blocking Cap (6). As in the example of Section 8.3.2, customizable patterns can by given by the user to control the device interdigitation in the respective governing modules.

Once again, a placement template –see Figure 8.18 (a)– is turned into a set of Relative Placement constraints (b) to achieve overall symmetry, and (also like in the previous example) an Imitation constraint forces the Blocking Cap (6) to mimic the input Current Mirror (2). Just as well, a Guardring constraint is assigned to the Differential Pair (1) here to make SWARM employ a corresponding Guardring PCell. An overview of the circuit's devices and modules is provided in Table 8.9.

The initialization phase is depicted in Figure 8.19: image (a) displays the 61 primitive devices instantiated by calling *Generate from Source* while image (b) shows the situation after imposing appropriate governing modules on these devices and using the `Modgen` command to obtain an arrangement which reflects the given placement template. As in the example of Figure 8.12, this initial constellation serves as the starting point for the self-organization in seven different SWARM runs covering seven different layout zone aspect ratios.

SWARM's layout for an aspect ratio of 2:3 can be seen in Figure 8.20. Altogether, a total of 26 interaction rounds spanning 6 tightening-settlement cycles were required for the emergence of this outcome. Although the number of rounds per settlement (4, 3, 3, 3, 4, 9) could have been distributed even

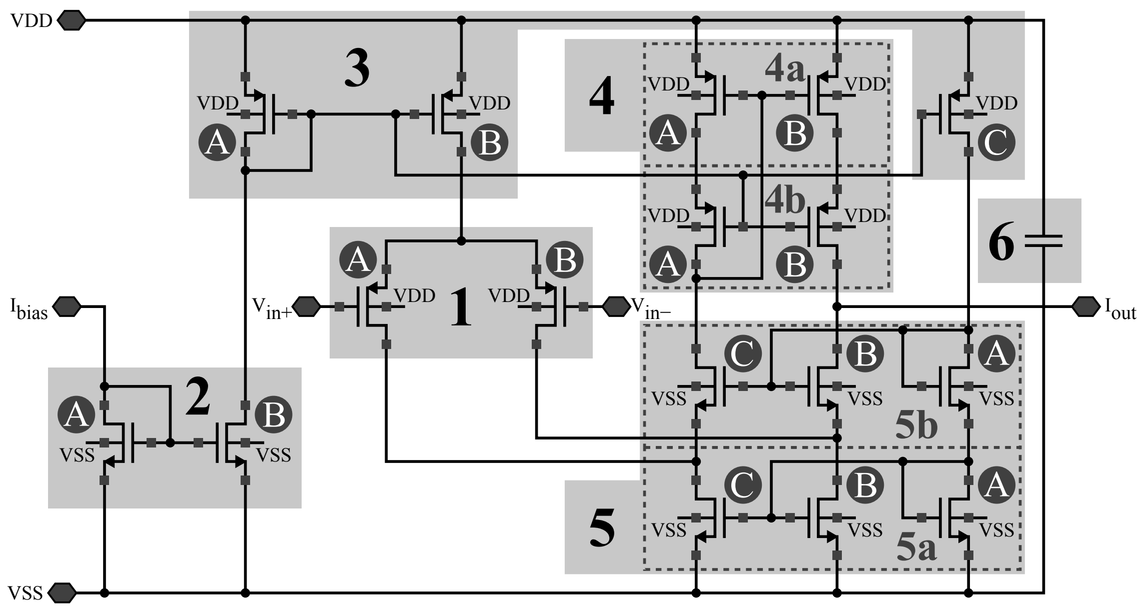
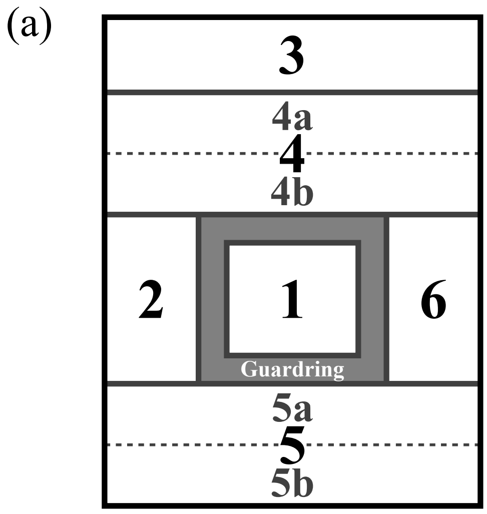


Figure 8.17: Schematic diagram of the Folded Cascode P-Input OTA circuit addressed with SWARM.



(b)

2	west of	1
3	north of	4
4	north of	1
5	south of	1
6	east of	1

Figure 8.18: (a) Placement template and (b) respective constraints for the Folded Cascode P-Input OTA.

Table 8.9: Overview of the functional units constituting the Folded Cascode P-Input OTA circuit.

Index	**Governing Module / Module Association**	**Device Type**	**Num. of Devices**	**Interdigitation Pattern**	**Ductility (Variants)**
1	Differential Pair	PMOS	4	A B / B A (fix)	40
2	Current Mirror	NMOS	6	B A B B A B	64
3	Current Mirror	PMOS	10	B C A C B B C A C B	64
4	Wide-swing Current Mirror	-	-	-	32
└**4a**	Bank	PMOS	8	A B B A A B B A	-
└**4b**	Cascode	PMOS	8	A B B A A B B A	-
5	Cascode Current Mirror	-	-	-	32
└**5a**	Bank	NMOS	12	B A C B A C C A B C A B	-
└**5b**	Cascode	NMOS	12	B A C B A C C A B C A B	-
6	Variator Module	Capacitor	1	-	full variability

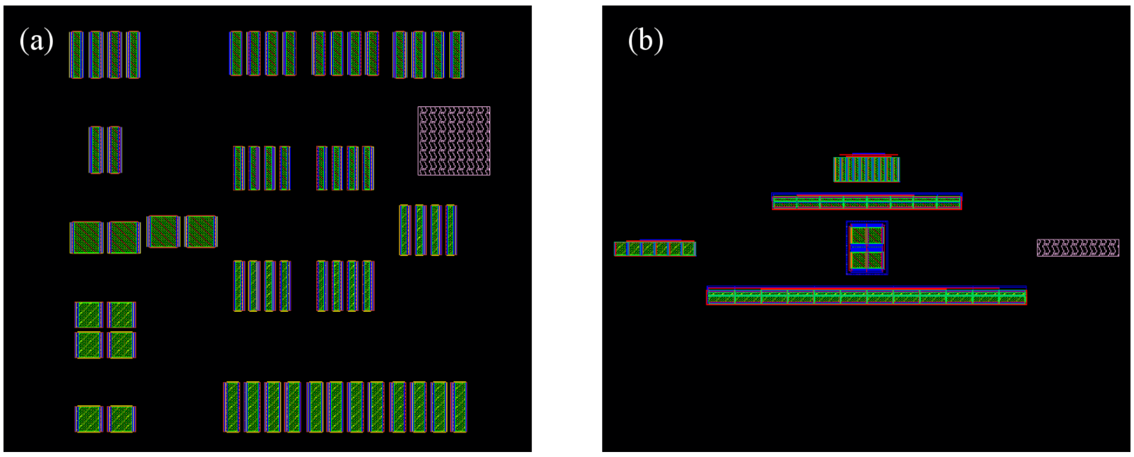


Figure 8.19: OTA layout (a) after device generation and (b) at the end of the initialization phase.

smoother, the module interaction was exceptionally efficient with respect to the observation that only 388 actions had to be explored for the 105 actions that were eventually performed (the best rate among all SWARM runs carried out for either the Symmetric OTA or the Folded Cascode OTA). Thus, no more than 28 seconds of runtime were measured for the self-organization phase in this example, which is less than the time needed for the initialization plus the finalization (adding up to 41 seconds, which accounts for 69 seconds in total).

The cardinal statistics of this SWARM run, as well as those obtained for the other six runs, are listed in Table 8.10. At large, the self-organization was again quite fluent as the rounds per settlement indicate. And like in the examples of Table 8.8, every kind of action has played its part here in one run or another. Regarding performance, the total runtimes of the listed SWARM runs (including the initialization, self-organization, and finalization phase) are actually shorter than those of Section 8.3.2: even in the worst case (encountered for an aspect ratio of 3:2), SWARM does not take much more than two minutes to produce a compact, constraint-compliant, completely placed and routed layout solution.

Table 8.10: Statistics of the different SWARM runs applied to the Folded Cascode OTA. The abbreviated kinds of actions are: Re-entering (R), Centering (C), Lingering (L), Budging (B), Hustling (H), Evasion (E), Yielding (Y), and Imitation (I).

Asp.	**Rounds per**	**Total**	**Number of Actions**				**Perf. Actions (By Kind)**								**Total**
Rat.	**Settlement**	**Rounds**	expl.	reje.	perf.	dism.	R	C	L	B	H	E	Y	I	**Runtime**
1:1	3, 3, 4, 4, 6	20	1283	1163	75	45	1	51	0	7	0	2	0	14	79 sec.
1:2	4, 3, 4, 3, 2, 5, 4, 10	35	1307	1097	135	75	2	102	0	4	0	2	0	25	91 sec.
1:3	6, 3, 3, 3, 3, 3, 3, 4, 4, 8	40	2742	2502	133	107	0	92	0	6	4	5	0	26	117 sec.
2:1	3, 3, 3, 4, 4	17	534	432	49	53	2	30	0	3	0	3	0	11	74 sec.
2:3	4, 3, 3, 3, 4, 9	26	388	232	105	51	1	81	1	1	0	2	0	19	69 sec.
3:2	3, 2, 2, 3, 3, 10, 4	27	2246	2084	97	65	3	58	0	12	1	3	2	18	129 sec.
5:2	3, 3, 3, 4	13	982	904	40	38	1	22	0	7	0	2	0	8	81 sec.

As illustrated in Figure 8.20, SWARM's outcome for a 2:3 aspect ratio precisely realizes the overall module arrangement specified by the placement template (Figure 8.18). The same is true for the 1:1 layout depicted in Figure 8.21, again relying on the cumulative variability of the modules to satisfy the contour of the square layout zone. In the result of Figure 8.22, achieved by SWARM for a 5:2 aspect ratio, the final module constellation represents a slightly different alternate arrangement (corresponding to that of Figure 8.16 (g) with modules 3 and 4 permuted) which is alright because it still attains the overall symmetry enforced by the set of Relative Placement constraints. This is also the case with the SWARM solutions for the other four aspect ratios (as confirmed by the respective screenshots that can be found in the appendix).

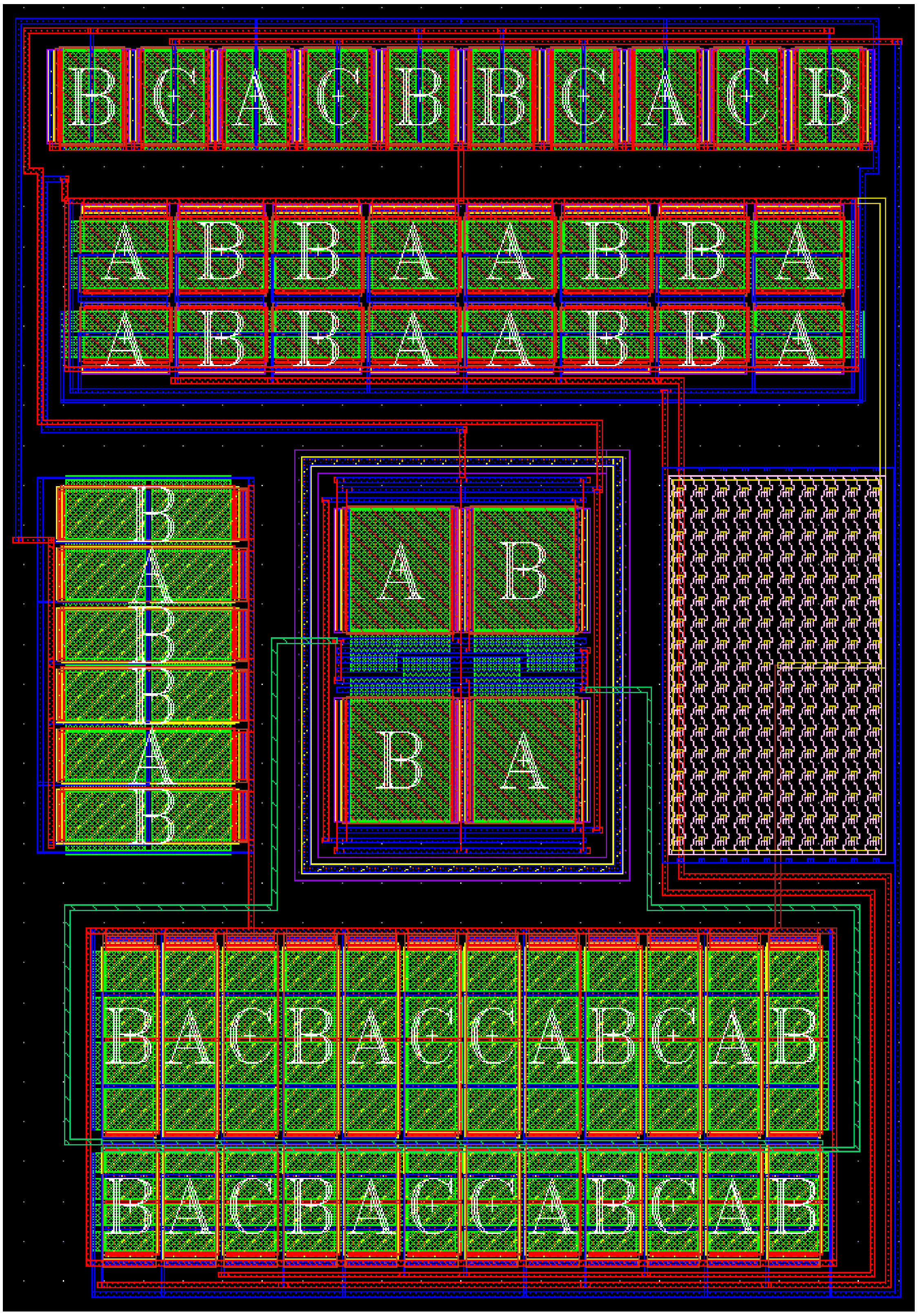


Figure 8.20: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 2:3 aspect ratio.

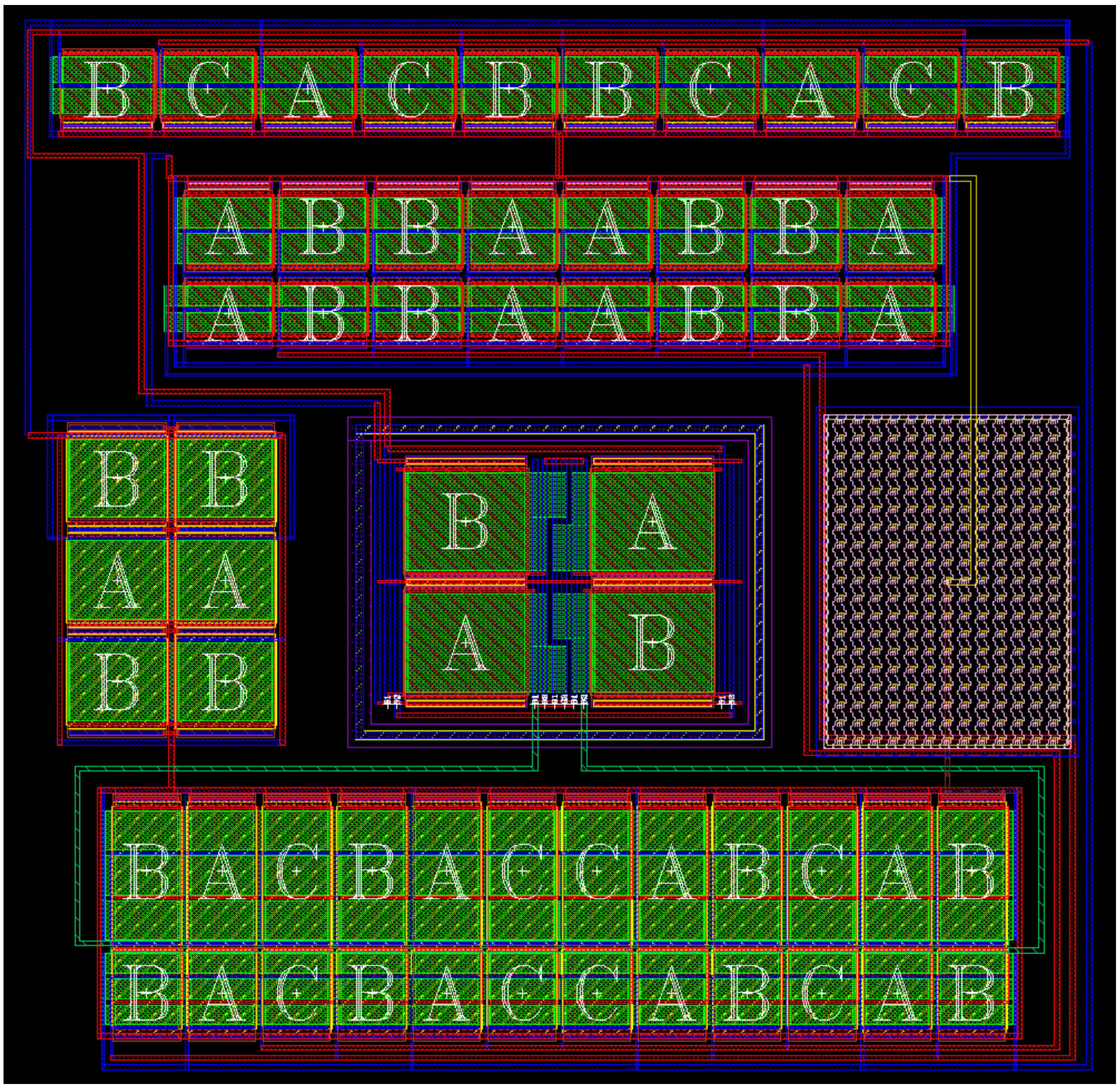


Figure 8.21: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 1:1 aspect ratio.

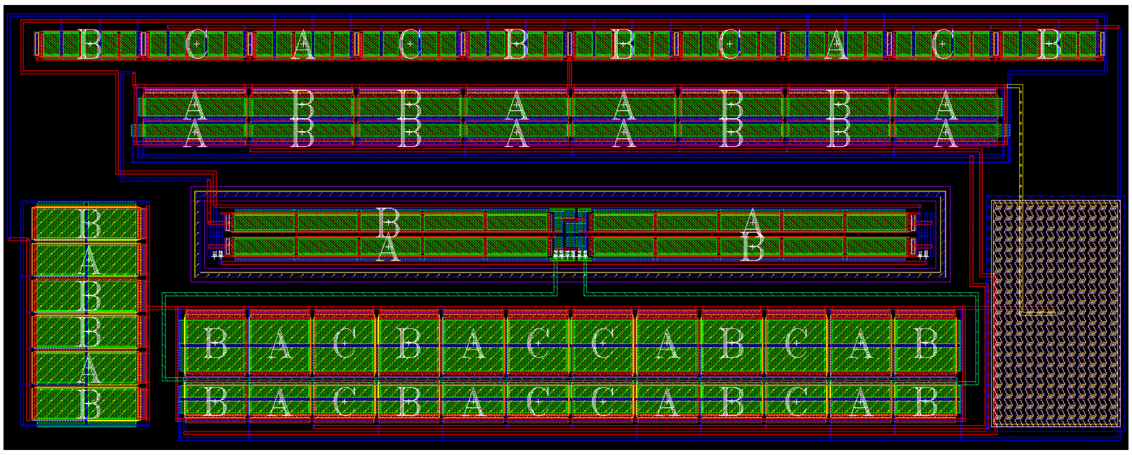


Figure 8.22: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 5:2 aspect ratio.

8.3.4 Assessment

As discussed in Chapter 4, assessing the practical worth of the SWARM methodology for real place-and-route problems should distinguish between its impact on *design productivity* (Section 4.2.1) and the degree of achievable *layout quality* (Section 4.2.2). The latter is investigated in the following Section 8.3.4.1 while the former will be subject to Section 8.3.4.2.

8.3.4.1 Assessment Regarding Layout Quality

Above all, a fundamental asset of SWARM is its ability to produce layout results which are completely placed and routed – a merit that already sets the methodology apart from placement-only approaches (Table 3.2 on page 41) and mere routing algorithms (Table 3.3 on page 42). Thereto, SWARM closely follows the common procedure of manual layout design, tackling the positioning of a module's devices and its intra-module wiring simultaneously to create the inter-module connectivity after the modules have been placed in the desired arrangement. Based on this modular approach, the layouts achieved by SWARM are qualitatively reminiscent of handcrafted solutions, both in terms of *functionality* and *consistency*.

Concerning functionality (see Section 4.2.2.1), the decisive strength of SWARM is to satisfy all relevant design constraints that would have been taken into account if the layout was created by a human expert. This involves an explicit *and* implicit constraint consideration, thus coming up to the technical aim of this thesis to include both formalized and nonformalized expert knowledge. Regarding the two exemplified OTA circuits, the following constraints have been taken into consideration by SWARM explicitly:[3]

- "Module" constraints imposed on each functional unit (e.g., a DiffPair constraint or a CurrMirr constraint) tell SWARM to manage the respective devices with the designated governing modules and module associations.
- If a governing module provides a dedicated parameter for specifying a custom device interdigitation pattern, then the desired pattern can by formally entered by the user as a string to be explicitly put into effect by the module.
- A Guardring constraint assigned to the Differential Pair effectuates the instantiation of a Guardring PCell around the four transistors, able to incessantly re-adapt itself during the module interaction.
- Using Relative Placement constraints derived from the predefined placement template, the interacting modules are confined in their action exploration via *noncompliance* – one of SWARM's five influencing factors.
- An Imitation constraint forces the Blocking Cap to mimic the transformations of the input Current Mirror, thus supporting the Relative Placement constraints in their aim to achieve overall symmetry.
- With the influencing factor *protrusion*, all modules are obliged to get into (and stay within) the contour of the user-defined layout zone. This represents the consideration of a Fixed Outline* constraint.
- Based on SWARM's general idea of a successively tightened layout zone, the modules are explicitly driven towards an increasingly compact arrangement (which effectively implements the optimization goal of area minimization).

If the modules were not bound to a template-based arrangement, some additional constraints can be added to the picture:

- The influencing factor *turmoil* may be used to pursue the optimization goal of wirelength minimization, and –by replacing a connection's *tension* with compressive stress– to separate certain modules from each other.
- Prompting a module to locate itself near the zone border can be achieved with a connection between the module and a (hypothetical) external component, as described in Section 8.2.1). This corresponds to a Boundary* constraint.

[3]Constraints marked with an asterisk (*) belong to those addressed in literature (see Section 3.1.1.2).

- To realize a Distance* constraint, a connection's *relaxation threshold* may be overridden with a maximum or minimum value acting as a hard restriction on the proximity or separation between two modules.

Turning to the internal layout of the governing modules, it goes without saying that their introversive behavior in general can, should, and in practice usually does indeed profit from emulating the best practice design procedures cultivated throughout a design team's history (and informally available as their legacy know-how). In the presented examples, constraints implicitly taken into consideration by the governing modules are:[3]

- If a module does not allow for custom device interdigitation, then the module is inherently bound to a specific interdigitation pattern. For example, the Differential Pair module always sticks to a crosswise AB/BA layout (thus implicitly satisfying a Common Centroid* constraint).
- The devices of a module are cohesively positioned according to a single-row or dual-row Alignment*, also taking care of consistent Orientation* (across or upright). This fulfills the demands of a Symmetry Island* constraint.
- Whenever a module deforms itself into a different layout variant by changing the *number of fingers* in its devices, this is uniformly done for all of these devices to secure that they still have Equal Parameters*.
- In each module, the respective devices are positioned in a highly compact fashion without lacking sufficient space for the wiring. The modules automatically see to an Abutment* that makes use of diffusion sharing.
- By means of the PCell Designer's geometry query language, the modules are able to geometrically determine whether and where their adopted devices feature bulk contacts. These are then automatically connected by the modules (which may even foster the application of bulk contacts and thus reduce substrate debiasing).
- No wires are routed on top of the devices but exclusively between them. This fundamental maxim reflects an implicit consideration of the constraint type introduced as Blockage*.
- Answering the Layer Limitation* constraint, only metal1 and metal2 wires are employed for every intra-module wiring. Apart from a few exceptions which additionally ask for metal3, this is also true for the inter-module wiring.
- In regard to current-carrying capacity, dedicated module parameters allow to make each *trunk* (lateral wire to connect a row of devices) wider than the perpendicular wires coming from the devices (also known as *pin-to-trunk* routes). This possibility pertains to the Wire Width* constraint.
- Without exemption, only double-cut vias (in remembrance of the Double-Cut Vias* constraint) are used throughout the entire layouts, not just for the connections inside each governing module but also by the Wiring PCells that create the connections between the modules.
- The connections between the modules are not tied to any layer preferences. For that reason, unnecessary layer changes between horizontal and vertical wire segments are avoided, which in turn minimizes the number of required vias.
- To avoid the antenna effect, all Current Mirror modules draw a metal1-only wire to connect their transistor gates with the drain of the reference transistor. On the same account, the Differential Pair module connects its transistor gates as depicted in Figure 3.14 (b), also using merely metal1.
- The modules generate their internal connections in a well-balanced fashion, evenly distributing the wires around (and –if appropriate– in between) the respective devices. This attains a high degree of symmetry, as well as homogeneity in the wire density.

Comparing this itemization with the state of the art (as surveyed in Chapter 3) underlines that the overall amount of constraints taken into consideration in SWARM by far exceeds the constraint coverage achieved with purely optimization-based or purely generator-based automation approaches. Living up to the technical aim of the thesis (as articulated in Chapter 5), this accomplishment –of considering constraints both explicitly and implicitly– plays to the facet of circuit functionality. Beyond that, the SWARM methodology also goes strong in terms of consistency (see Section 4.2.2.2):

- Since SWARM relies on well-established module PCells to automate the analog basic circuits that constitute the design as functional units, it effectuates an intuitive approach of functional modularization that the design team is supposed to be familiar with.
- The modularization eases a visual inspection of the layout as it allows to pick a single PCell (e.g., a certain Wiring module) and examine it in isolation. Furthermore, signing off the layout is disburdened by inspiring trust on the designer's side, rooted in the confidence and experience that many design constraints are implicitly satisfied by the PCells.
- SWARM works completely deterministic, which means that any of its layout solutions can be exactly reproduced if necessary. In conjunction with the observation that a SWARM run typically comprises only a modest number of actions (rather than millions of incomprehensible perturbations), this offers the great opportunity that the run itself can be surveyed by the user and thus visually inspected in real-time.
- The final layouts are neither entirely flat nor utterly secluded. Instead, the procedural powers of the modules are still available such that parametrical adjustments can be made at module level. Even if these adjustments affect the inter-module connections, the Wiring modules can be deleted in order to repeat the finalization phase of the SWARM run.
- A pivotal advantage about context awareness is that even though the given schematic may be flat (i.e., built from primitive devices), modular facilities can be utilized in the layout via governing modules – without sacrificing SDL-conformity, because a distinct one-to-one device correspondence is maintained.[4]
- SWARM has been seamlessly integrated into the design environment (using native Virtuoso features such as *Generate from Source*, the ModGen tool, and –of course– PCells). This implementation merely enhances the design flow instead of replacing it, so it does not introduce any artifacts incompatible with the existing tool chain. If the layout produced by a SWARM run is indistinguishable from a genuine manual solution, further processibility is definitely guaranteed.
- Design iterations due to a re-sizing of the circuit or a PDK update (that involves a revision of the governing modules' master PCells) should be no problem as long as the circuit topology and the design constraints don't change. A new SWARM run, based on the same arguments as before, is supposed to spawn an adequate layout once again. To that end, the arguments of a SWARM run can be stored in a separate file and reloaded the next time.
- As already stated in Section 3.1.2.2, employing PCells to automate simple modules facilitates re-use of expert knowledge in a way much smarter than through rough-and-ready copy-pasting. Naturally, the same inventory of governing modules can be utilized in different SWARM runs of the same project and in other projects of the same semiconductor technology.
- To address projects in different technologies, the governing modules have been implemented in a way by which technology-specific data (e.g., layer names) is separated from the actual PCell code and instead passed to the PCell via parameters. In the best case, migrating a PCell to another technology is achieved solely by adjusting the technology-specific parameter values without having to touch the PCell's command sequence.

8.3.4.2 Assessment Regarding Design Productivity

Due to the large amount of assessment criteria (see Figure 4.1 on page 69), calculations on design productivity are no exact science, but certain statements can be made when distinguishing between different orders of magnitude. For giving estimations on the involved design effort, Table 8.11 defines a vocabulary of temporal quantities from 1' (one minute) to 1Y (one year). For each of these quantities, the table lists the respective time span in minutes since the subsequent calculations also utilize minutes as the basic unit of time. Every quantity represents the labor costs ascribed to one human expert (i.e., a month –1M– in fact denotes a person-month, for example). Where it is appropriate, intermediate values such as 5' (five minutes), 2D (two days), or 2W (two weeks) can be found.

[4]Difficulties can arise concerning net correspondence because symbolic LVS checkers may be unable to descend into a Wiring PCell and discern its internal connectivity. However, modern SDL flows are beginning to offer suchlike hierarchical support to some degree.

Table 8.11: Temporal quantities used to estimate the design effort. Please note that a day is understood as a working day, a week is understood as a working week, and so on.

Quantity	Meaning	Calculated As	In Minutes
1’	1 minute	-	1
1h	1 hour	60 minutes	60
1D	1 day	8 hours	480
1W	1 week	5 days	2400
1M	1 month	4 weeks	9600
1Q	1 quarter	3 months	28800
1H	1 half-year	2 quarters	57600
1Y	1 year	2 half-years	115200

The subsequent assessment provides a general comparison of the four fundamental layout strategies discussed in this thesis: manual layout design (without any kind of automation), generator-based automation, optimization-based automation, and SWARM. The intention of this comparison is to gauge every strategy’s suitability with respect to the *component magnitude* (reflecting the functional complexity of the component being layouted), let it be a *primitive device*, a *simple module*, an *advanced module*, or a high-level *block* (as introduced in Table 2.1 on page 20). For each of these component magnitudes, the respective number of occurrences n_C per chip is reckoned in decades from 1000 instances down to 1 instance.

For every layout strategy and component magnitude, regarding one type of component (which may cover an entire circuit class if it does not represent a primitive device), Table 8.12 makes a best-case and a worst-case educated guess on the respective layout design effort. In both cases, the estimation covers the fixed preliminary effort Eff_P (which incurs prior to the actual design work) and the effort Eff_O per occurrence (i.e., the labor required to layout each individual instance of the component in the design project). Based on these estimations, the total average effort Eff_C per chip in each case is calculated via

$$\mathit{Eff}_\mathrm{C} = \frac{\mathit{Eff}_\mathrm{P} + \mathit{Eff}_\mathrm{O} \cdot n_\mathrm{C} \cdot n_\mathrm{T}}{n_\mathrm{T}} = \frac{\mathit{Eff}_\mathrm{P}}{n_\mathrm{T}} + \mathit{Eff}_\mathrm{O} \cdot n_\mathrm{C} \tag{8.2}$$

where n_T represents the total number of chips under consideration for a particular semiconductor technology. Concerning the estimation at hand, a value of $n_\mathrm{T} = 1$ is assumed first. To give further insight into the figures of Table 8.12, a couple of remarks should be made:

- Financial expenses whatsoever are omitted from this assessment – only effort in the sense of labor costs is taken into account.
- For simple modules, advanced modules and blocks, it is assumed that the primitive devices are readily given as procedural generators.
- As discussed in the context of Figure 3.16 (page 55), automating advanced modules –or even blocks– as genuinely procedural generators is virtually impossible since their variability covers a continuous and infinitely huge parameter space. For the sake of comparison, Table 8.12 supposes that it *is* possible – but only if a correspondingly large amount of preliminary effort is spent on their development.
- By the same token, Table 8.12 hypothesizes the availability of optimization algorithms that are able to perform both placement and routing, that can explicitly consider all kinds of placement and routing constraints, and that indeed find constraint-compliant solutions in expert quality. The difficulty with (or nonexistence of) such algorithms is reflected by the exorbitant amount of constraining effort necessary to express all design constraints in a formal fashion.
- It goes without saying that optimization algorithms are deemed to be *not applicable* (n/a) for automating primitive devices, and the same is of course also true for the SWARM methodology. Just as well, SWARM is not applicable to produce layouts of simple modules because such modules already represent the interacting entities in a SWARM run.

- When employing the SWARM methodology to target an advanced module, the preliminary effort for implementing the simple modules that are to be impelled into the interaction may seem quite low. However, much substance can be obtained from the module exemplifications of this thesis (and –much more– from generators that might already be in use within the design team). Furthermore, a lot of effort can be saved if the module implementations are cleverly derived from generic modules for the positioning and the wiring – as done in this thesis (see Table 7.5 on page 114).
- Regardless of whether SWARM is used for an advanced module or an entire block, the preliminary effort is expected to be roughly the same because the cadre of simple modules is basically always the same. In the latter case, although not realized in the scope of this thesis, the –discretely variable– simple modules are meant to interact with each other and thus form advanced modules (as in the examples of Section 8.3.2 and Section 8.3.3) while these –fully variable– advanced modules interact with each other on the next-higher hierarchy level (like the floorplan blocks in the examples of Section 8.2). To that end, only the utilization effort during design is expected to become greater (as indicated in Table 8.12).

Table 8.12: Estimated layout design effort for different layout strategies and component magnitudes.

	Primitive Device		**Simple Module**		**Advanced Module**		**Block**	
Occurrences per Chip	1000		100		10		1	
Best Case or Worst Case?	Best Case:	Worst Case:	Best Case:	Worst Case:	Best Case:	Worst Case:	Best Case:	Worst Case:
Effort with Manual Layout Design								
Preliminary Effort	0’	0’	0’	0’	0’	0’	0’	0’
Effort per Occurrence	10’	1h	1h	1D	1D	1W	1W	1M
Total Effort per Chip	10000’	60000’	6000’	48000’	4800’	24000’	2400’	9600’
Effort with Generator-based Automation								
Preliminary Effort	1h	1W	1W	1M	2M	1H	1Q	1Y
Effort per Occurrence	1’	5’	5’	20’	1h	1D	1D	1W
Total Effort per Chip	1060’	7400’	2900’	11600’	19800’	62400’	29280’	117600’
Effort with Optimization-based Automation								
Preliminary Effort	n/a	n/a	1h	1D	1h	1D	1h	1D
Effort per Occurrence	n/a	n/a	1D	1W	2D	1M	2W	1Q
Total Effort per Chip	n/a	n/a	48060’	240480’	9660’	96480’	4860’	29280’
Effort with SWARM Methodology								
Preliminary Effort	n/a	n/a	n/a	n/a	2W	2M	2W	2M
Effort per Occurrence	n/a	n/a	n/a	n/a	1h	1D	1h	1W
Total Effort per Chip	n/a	n/a	n/a	n/a	5400’	24000’	4860’	21600’

Grouped by the respective component magnitude, the total layout design effort per chip (covering both the best-case and the worst-case estimation) for every layout strategy is displayed in the chart of Figure 8.23 on a logarithmic scale. From this depiction, the following conclusions can be drawn:

- Considering a primitive device, it is generally advisable to implement it as a procedural generator. In regard to the estimation made here, even the worst-case effort on the generator's side is lower than the best-case effort in manual design. This reckoning underlines why it is common practice to utilize PCells at device level. Only in cases where a primitive device is assumed to be used quite rarely, it may be favorable to put up with manual polygon pushing.
- For a simple module, generator-based automation can basically be expected to outrival manual layout design. However, the situation is not as palpable as with primitive devices. Whether automating a simple module as a procedural generator will presumably pay off in the end, needs to be predetermined on a case-by-case basis within the respective design team.

- Optimization-based automation is the least appealing layout strategy on the level of simple modules where satisfying the relevant design constraints is a duty that expert layout engineers have off pat. This suggests to pass over the huge constraining effort for an optimization algorithm in favor of manual layout design or –maybe even better– a procedural generator.
- When the component magnitude reaches that of an advanced module, the effort for generator-based optimization already escalates into uneconomical regions. In contrast, optimization algorithms become more rewarding, but still they can beat manual layout design only in singular cases. That is why advanced modules are mostly done in a manual fashion – so far.
- Advanced modules are where the SWARM methodology may achieve substantial benefit for design productivity. If only one circuit class is targeted, the effort is –according to the experiences from this thesis– at least comparable to manual layout design. The true benefit unfolds when more than one circuit class is automated via SWARM because the set of governing modules –and thus the preliminary effort– is basically the same. Figure 8.23 illustrates the best-case estimation supposing that five circuit classes are covered: the preliminary effort of two weeks (ten working days) is virtually cut down to two days per circuit class, resulting in a total effort of $\mathit{Eff}_\mathrm{P} + \mathit{Eff}_\mathrm{O} \cdot n_\mathrm{C} = 2 \cdot 480' + 60' \cdot 10$ which amounts to 1560 minutes.
- Regarding a high-level layout block, optimization-based automation is thought to overtrump procedural generators as these become increasingly inappropriate when ascending towards larger component magnitudes. However, manual layout design is still unrivaled here (again except certain isolated cases, if at all). Inhowfar SWARM can compete for design productivity here, is to be investigated in future works. Following the projected numbers of Table 8.12, Figure 8.23 shows that SWARM is probably on a par with optimization-based automation if only one circuit class is concerned. As in the case of advanced modules, SWARM's benefit rises with the number of circuit classes being covered and may thereby noticeably outplay all of the other layout strategies. The illustrated best-case estimation for five circuit classes is $2 \cdot 480' + 60' \cdot 1$ (= 1020 minutes).

While the estimations of Table 8.12 contemplated only one single chip, it is also interesting to analyze how the layout design effort does change when multiple chips in the same semiconductor technology can profit from a certain layout strategy. For values of $n_\mathrm{T} > 1$, a look at equation 8.2 easily reveals what happens to the total average effort per chip: the greater the preliminary effort Eff_P is in relation to the effort Eff_O per occurrence, the larger the benefit is for design productivity when more than one chip is taken into account. This fundamental law is reflected by the differently oriented arrows in Figure 8.23, indicating the respective change in effort per chip.

With respect to manual layout design, there is no change in effort per chip for $n_\mathrm{T} > 1$ since there is no preliminary effort involved. Concerning generator-based automation, Eff_P is much greater than Eff_O, which redounds to a significant decrease in design effort when the procedural generators are utilized throughout multiple chips. By contrast, the decrease in design effort is only marginal in the case of optimization-based automation due to the large constraining effort during design, which towers over the preliminary installation effort. With SWARM, the labor again incurs much more on the predevelopment side rather than during an IC project, so the design effort per chip significantly decreases with every chip that is being targeted.

Regarding the degree of achievable layout quality (discussed in Section 8.3.4.1) as well as the good performance in terms of design productivity (examined in this Section 8.3.4.2), it is fair to say that the accomplished work on SWARM so far pressed the right buttons to successfully master the practical ambition of this thesis (as formulated in Chapter 5). To illuminate the scientific achievement of the presented work, the subsequent Chapter 9 puts the SWARM methodology in the greater context of a "higher-level" design flow addressing both schematic design and layout design.

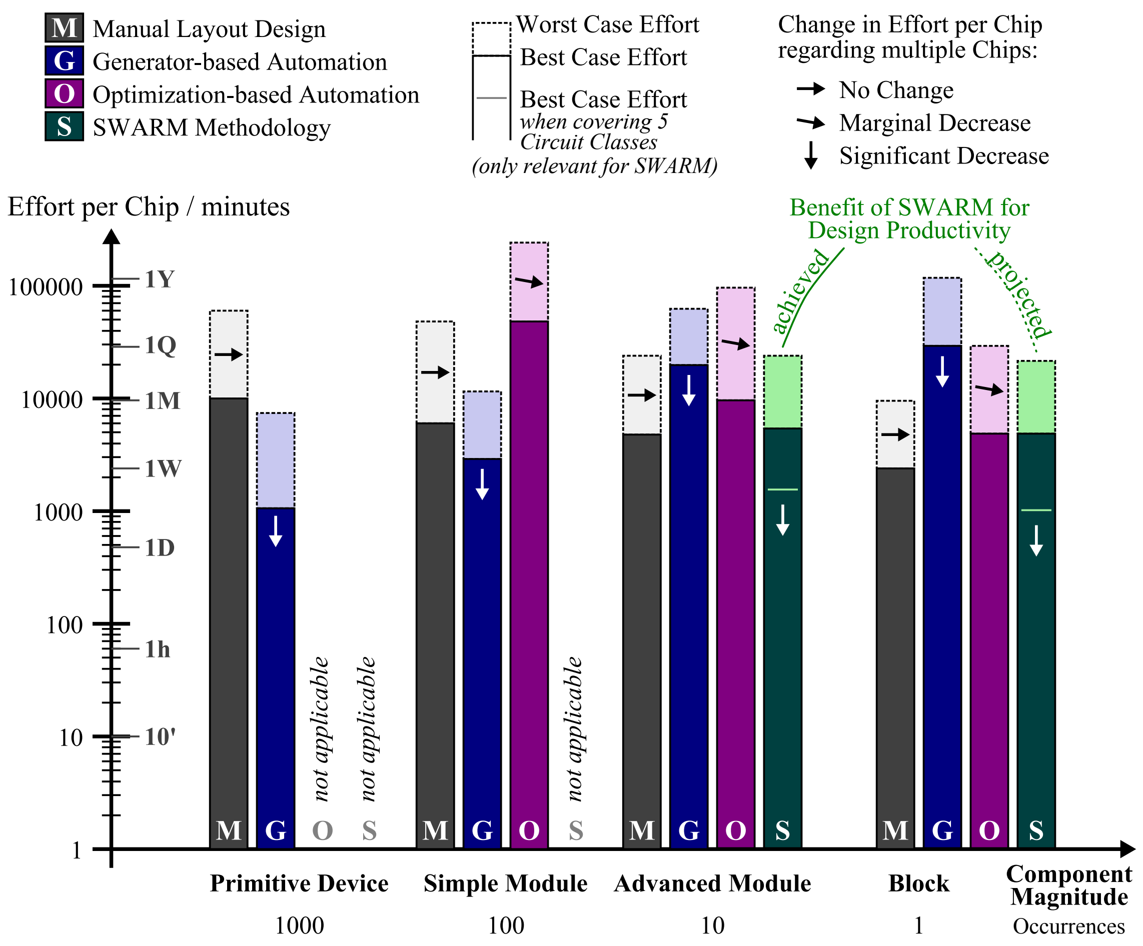


Figure 8.23: Depiction of the various layout design efforts estimated in Table 8.12.

Chapter 9

Towards a Holistic Design Flow on Module Level

From time to time you should step back from
yourself like a painter from his painting.
Christian Morgenstern (German poet)

This chapter takes a step back from the developed SWARM methodology and frames its role inside the bigger picture of an improved design flow which seamlessly covers both of the two major design steps in IC design: schematic design and layout design. For this purpose, Section 9.1 touches on a couple of cognate topics accompanying SWARM in the pursuit of that holistic long-term vision. Then, Section 9.2 throws a glance at the scientific value of SWARM, considering the sketched-out EDA roadmap towards a novel *bottom-up meets top-down* automation paradigm.

9.1 Cognate Topics Across the Three Different Design Domains

Referring to the well-known *Y diagram* [361], the IC design flow can be said to traverse three different *design domains* denoted as the *behavioral domain* (which deals with the general function of the circuit), the *structural domain* (that the schematic representation of the circuit pertains to), and the *physical domain* (wherein the detailed layout geometries of the circuit are to be described). As illustrated in Figure 9.1, the step of schematic design is thus set between the behavioral domain and the structural domain while the step of layout design accounts for the transition from the structural domain to the physical domain.

In former times, when procedural generators did not reach beyond the level of primitive devices, the design flow exhibited a hierarchical break between the behavioral and the structural domain because creating flat schematics and flat layouts does not match the designers' way of thinking in functional units (i.e., modules) when the behavior of a circuit part is concerned. This inconsistency involves a loss of information about the purpose of a module and its respective design requirements. To prevent such discontinuities, this thesis has been conceived as part of an envisioned design flow on module level that holistically proceeds throughout all of the three design domains. Tackling both of the two intermediary design steps, Figure 9.1 displays related works that have been (or are being) developed within (or beyond) the scope of this thesis.

For historical reasons, these works are now described in backwards order (i.e., converse to the design flow), from the physical domain (Section 9.1.1) through the structural domain (Section 9.1.2) to the behavioral domain (Section 9.1.3).

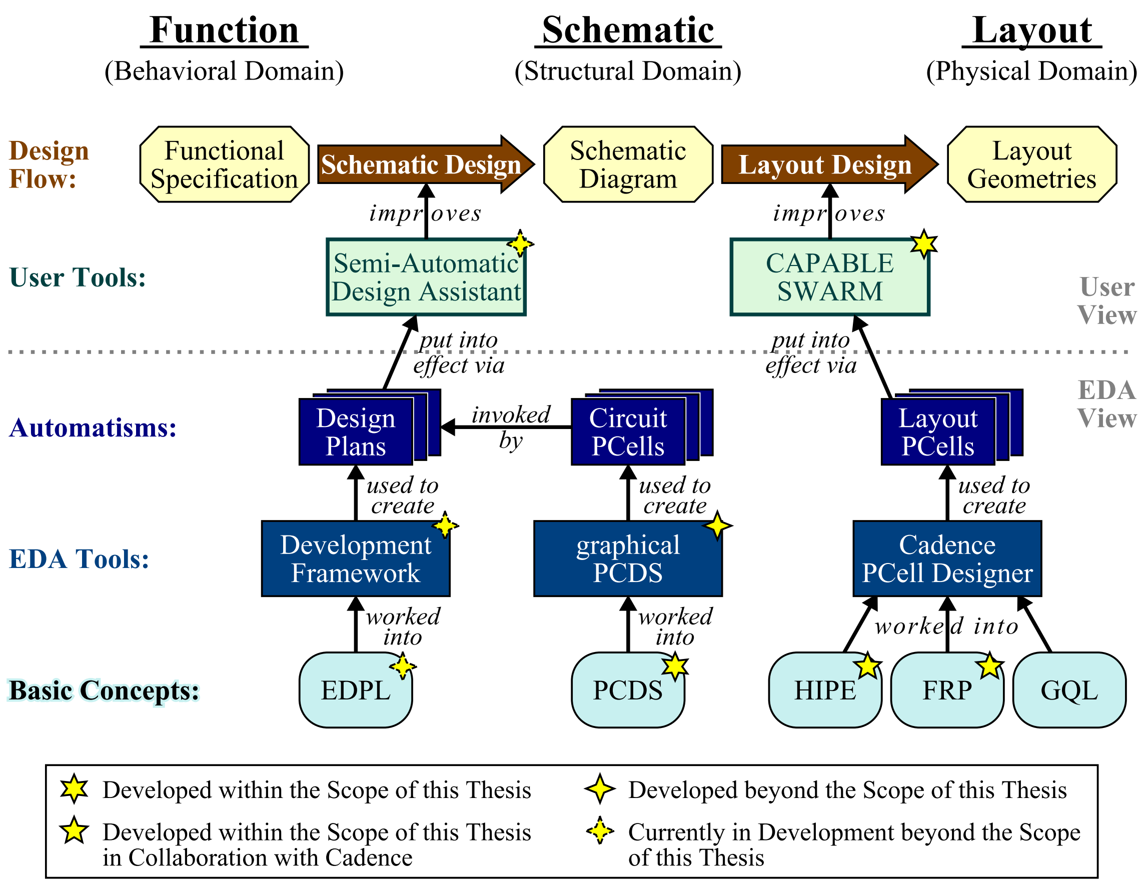


Figure 9.1: SWARM and its partner subjects in pursuit of a holistic design flow on module level.

9.1.1 Works Concerning the Physical Domain

The SWARM methodology (and its host tool CAPABLE) were initiated upon the success story of layout PCells by which generator-based automation began to pioneer in the physical domain. Especially with the advent of the Cadence PCell Designer tool, the potential of procedural generators –and likewise the need for further concepts to keep up with the new possibilities and challenges– became increasingly obvious. One fundamental pillar that the PCell Designer has been built upon from the beginning is the already-mentioned *geometry query language* (GQL).

Along with a couple of other enhancements, the idea of *FR PCells* (FRP) –which served as the conceptual trailblazer for SWARM (see Section 7.2.1)– was invented in collaboration with Cadence during the work on this thesis. Another basic concept added to PCell Designer is *Hierarchical Instance Parameter Editing* (HIPE) [362], which enables a parametrical customization of module PCells throughout arbitrary levels of subhierarchy (as demonstrated in [363]). The scientific earning of HIPE is that the PCell parameters need not be entirely predefined by the master PCell, but are dynamically determined depending on the contents of the specific PCell instance.

9.1.2 Works Concerning the Structural Domain

To accommodate the ascent of layout PCells in the physical domain, attention has come to the development of corresponding circuit PCells in the structural domain. As already explained in Section 3.1.2.5, a circuit PCell in fact consists of two PCells: a schematic PCell and a symbol PCell. With the intention of facilitating the implementation of such PCells, a domain-specific language named *Parameterized Circuit Description Scheme* (PCDS) has been worked out in the scope of this thesis. Apart from providing high-level *shape commands* and a *relative placement* notation, PCDS realizes a *dual interpretation* technique by which both the schematic PCell and the symbol PCell can be compiled from one single PCDS description. As outlined in [364], PCDS reduces the amount of necessary code lines by a factor of 50 on average.

Despite all these merits, PCDS still relies on a textual implementation of circuit PCells which is far from the mentality of circuit design experts accustomed to visually drawing schematic diagrams. Hence, the winning pitch here is to offer a graphical PCell programming approach in the fashion of PCell Designer. This has been prototypically achieved through *graphical PCDS* (gPCDS) [365], a circuit PCell development tool realized in a contemporaneous research project. Meanwhile, that approach has aroused so much interest that the basic concepts of PCDS are being integrated into PCell Designer to deliver a professionalized version of gPCDS. A first glimpse on these new features has been caught in [366].

9.1.3 Works Concerning the Behavioral Domain

To complete the route towards a holistic design flow on module level, the next logical measure is to transfer the idea of procedural automation over to the behavioral domain and thus address the step of schematic design. This step is about taking a functional circuit specification (mainly expressed via performance parameters) and producing a sized schematic (that meets the functional specification). For this purpose, procedural automation can be employed to handle the inherent tasks (finding an appropriate circuit topology, choosing the device types, and setting the device dimensions) through a so-called "expert design plan", which is a sequence of instructions mimicking a human expert's recipe-like design proceeding. In [367], the feasibility of doing so has been exemplarily demonstrated on the OTA circuit class.

As in the physical domain and the structural domain, the grand bargain for EDA is not to develop such design plans but rather to devise adequate tools for their development. In this spirit, an *Expert Design Plan Language* (EDPL) is currently being elaborated [368] as a companion concept of PCDS, so to speak. Based on EDPL, it is furthermore scheduled to realize a sophisticated development framework (serving as the logical counterpart to gPCDS and PCell Designer) for the creation of expert design plans. This approach becomes well-rounded if an expert design plan is enabled to invoke a corresponding circuit PCell (covering all topologies of the respective circuit class) that generates the schematics needed for simulation. On the user's side, a GUI-based tool is meant to bring an expert design plan to life (similar to a cassette deck playing a tape). Since such a tool relies on close interaction with the user, it is presently labeled as a *Semi-Automatic Design Assistant* (SADA).

9.2 The Scientific Value of SWARM: Meeting Bottom-up With Top-down

Beside the practical merits of SWARM, strategically more valuable is the academic achievement of illustrating how the two converse automation paradigms denoted as *bottom-up* and *top-down* can be made to work together. Interestingly, SWARM's journey towards this vision of a novel *bottom-up meets top-down* philosophy followed the five successive steps articulated in Section 3.2.3:

(1a) The research project behind SWARM was encouraged by the development (and positive reception) of the PCell Designer tool, for which a lot of technical input –stemming from the work on SWARM– has been conveyed to Cadence.
(1b) The PCell Designer facilitated a swift implementation of the procedural generators for the simple modules (analog basic circuits) as layout PCells, utilizing parameters to cover the module-specific degrees of freedom.
(1c) Building on the idea of context awareness, the CAPABLE framework has been realized to incorprate the implemented layout PCells into the design flow as governing modules that are applied to primitive devices in the layout.
(2) The algorithmic way of optimizing a layout has been worked into the conception of a control organ which exerts pressure by repeatedly tightening the available layout space to enforce increasingly compact layouts indirectly.
(3) Devising a decentralized system, where constraint-compliant layout outcomes emerge from the self-organization of interacting modules (allowing the modules to consider their innate design re-

quirements bottom-up and simultaneously satisfying the top-down demands of the control organ), represent the glue for uniting the two automation paradigms and let bottom-up meet top-down.

As already indicated, SWARM's ability to take both formalized and nonformalized expert knowledge into account –via explicit and implicit constraint consideration– is reflected in joining the cardinal strengths of optimization-based automation and generator-based automation: the versatility of the approach (i.e., not having to preconceive the entire solution space in advance) and the resulting layout quality (i.e., not having to specify every single detail through formal constraints). To provide a final overview, Table 9.1 summarizes the major characteristics of SWARM, as opposed to optimization algorithms and procedural generators.

Table 9.1: Major characteristics of optimization algorithms, procedural generators, and SWARM.

Automation Strategy	**Optimization Algorithm**	**Procedural Generator**	**SWARM Methodology**
Circum-scription	Problem-Solving Routine	Layout-Creation Script	Multi-Agent System
Working Principle	Loop of Exploration and Evaluation	Straight Sequence of Design Commands	Self-Organization of Interacting Modules
Expert Knowledge	Formalized	Nonformalized	Formalized and Nonformalized
Constraint Consideration	Explicit	Implicit	Explicit and Implicit
Designated Output	Layout Solution	Layout Result	Layout Outcome
Solution Policy	Solution Found by Optimization Loop	Solution Preconceived by Human Expert	Solution Emerges from the Interaction
Behavioral Determinacy	Unpredictable, often Randomized	Deterministic and even Predictable	Deterministic but not Predictable
Cardinal Strength	Versatility of the Automatism	Quality of the Resulting Layout	Autom. Versatility and Layout Quality
Automation Paradigm	Top-Down	Bottom-Up	Bottom-Up meets Top-Down

From today's perspective, SWARM pioneers an entirely new field for EDA expected to blossom with many more innovative approaches that will further contribute to *bottom-up meets top-down* automation. Many questions –like whether upcoming works will also facilitate an interplay of bottom-up procedures with genuine top-down algorithms in their traditional form– are subject to future research and shall therefore be delegated to upcoming generations of EDA enthusiasts. Nevertheless, some embryos of big ideas –such as an interaction of context-aware, "smart" Wire PCells that perform a self-intelligent algorithmic pathfinding– have already come to mind.

Chapter 10

Summary and Outlook

"Begin at the beginning," the King said very gravely,
"and go on till you come to the end: then stop."
Lewis Carroll: Alice in Wonderland

This thesis presents a new methodology for layout automation in the design of analog integrated circuits, referred to as *Self-organized Wiring and Arrangement of Responsive Modules* (SWARM).

While digital IC design has already been following highly automated flows for layout synthesis since the 1980s, many analog layout automation approaches do not find evident industrial acceptance so far, even though a couple of commercial tools –primarily focused on *placement* and *routing*– have arisen from the past three decades of EDA. The basic reason for this circumstance is that –in contrast to the *quantitative complexity* (More Moore) in the digital domain– the functional diversification of analog circuits and the continuous nature of its signals makes the analog design problem a matter of *qualitative complexity* (More than Moore). This involves many diverse and concurrent functional *constraints* (design restrictions and optimization goals) that need to be considered simultaneously.

Until today, thorough constraint consideration can only be satisfyingly achieved through a laborious and largely manual design flow, drawing heavily upon the *expert knowledge* of human layout engineers. For that reason, the presented methodology particularly concentrates on a comprehensive incorporation of expert knowledge into the automation. In real design flows, expert knowledge is communicated in *formalized* and *nonformalized* ways, which corresponds to an *explicit* and *implicit* consideration of constraints, respectively. Since both is required to solve the analog layout problem in its entirety, the technical aim of the SWARM methodology is to devise a new automation approach that supports both an explicit *and* implicit consideration of constraints.

For that purpose, this thesis attempts to combine two basic automation strategies: *optimization algorithms* (which can consider constraints only explicitly) and *procedural generators* (able to consider constraints only implicitly). As a scientific look reveals, these two automation strategies follow two fundamentally different *automation paradigms* denoted as *top-down automation* and *bottom-up automation*. The asset herein is that their respective strengths and weaknesses complement each other, which suggests that a coalescence of *bottom-up meets top-down* has much more potential than optimization-based or generator-based approaches alone. However, the scientific challenge is posed by the question if and how the two different automation paradigms can be brought together.

To address this challenge, SWARM pursues a very interdisciplinary approach building upon the idea of *decentralization* and touching various fields related to *multi-agent systems*. Simple *analog basic circuits* are realized as autonomous, mobile, self-interested, parameterized layout modules that interact with each other as agents inside an increasingly tightened layout zone. By provoking a flow of *self-organization*, a compact and constraint-compliant layout *outcome* is meant to emerge from the module interaction, reflecting the phenomenon of *emergence* as can be found in nature. SWARM pays respect to several principles of self-organization such as operating near the *edge of chaos* and promoting *synergy*, and draws parallels to several other types of decentralized systems subsumed under the term *artificial life* (e.g., cellular automata and agent-based models of collective motion). The SWARM methodology consists of three core concepts that are correlated with each other:

Responsive modules realize the simple analog basic circuits in the layout (e.g., Current Mirrors and Differential Pairs). They are implemented as procedural generators whose natural *introversive behavior* is enhanced with an *extroversive behavior*. In short, the major traits are:

- The responsivity is achieved through an *interface fabric* that equips the modules with the ability to read and modify their design context.
- Based on this *context awareness*, a responsive module can manage other components in the layout as a *governing module* (e.g., positioning the components and wiring them).
- Governing modules can be joined and/or imposed onto each other to build hierarchical *module associations*.
- Each module covers a certain amount of *intrinsic variability* (spanning the module layout's *degrees of freedom*) which –in the case of a hierarchical module association– evaluates into a *cumulative variability*. This variability is then exploited during the module interaction.

Module interaction refers to the concept behind the decision-making by which the responsive modules interact with each other as *participants* during the self-organization phase of a SWARM run. Every participant adheres to a common *action scheme* consisting of four *measures*:

- Assessing the participant's *condition*, which depends on five *influencing factors* (interference, turmoil, protrusion, wounds, and noncompliance).
- Perceiving its *free peripheral space* (i.e., the –presumably– vacant, rectangular area around the participant).
- Exploring and evaluating possible *actions*, which involves nine native actions (such as *Budging* and *Swapping*) but may also revert to custom actions (e.g., *Imitation*).
- Executing the *preferred* action (or staying idle) according to a distinct *comparison metric*. By executing an action, the participant can move, rotate, and deform itself (i.e., assume another layout variant as featured by its cumulative variability).

Interaction control is exerted by a control organ responsible for carefully steering the module interaction towards the desired outcome. This is done in an indirect way, by successively modifying the size of the layout zone's outline:

- Beginning with an initial constellation of *primitive devices* (upon which the governing modules are instantiated), the user-defined layout zone is set and *enlarged* to a significantly greater size (that easily accommodates all modules).
- Then, several *rounds* of interaction are performed until all participants have *settled* (i.e., no participant has executed an action within the same round).
- If all participants are in a valid location, the interaction control organ *tightens* the layout zone according to a certain *tightening policy* (in order to induce another *settlement*).
- Multiple *tightening-settlement cycles* are performed until the layout zone reaches the user-defined size (or the participants fail to settle in a viable constellation).
- Throughout the interaction, solid and volatile *comfort padding* can be wrapped around a participant. The latter changes with the zone size in favor of a fluent self-organization. The former preserves a fix amount of layout space that can be used by an inter-module routing step to complete the layout in a subsequent finalization phase.

In view of the current implementation, applying the SWARM methodology to a plain placement problem has shown that even *globally optimal* outcomes can emerge from the module interaction. Contemplating that the responsive modules do not cooperate (but compete) with each other, and that the modules do not survey the problem as a whole (but only have a limited viewpoint and selfishly pursue –above all– their personal desires), this observation is quite remarkable and substantiates the decentralized self-organization approach taken by SWARM.

With the power to support *full variability* (whereby the aspect ratio of a layout module –representing a black box– can change within a continuous range) as well as nonrectangular (but rectilinear) outlines of the layout zone, SWARM is also suitable for the task of *floorplanning*. Therein, SWARM not only minimizes the total layout area but also the distances of floorplan blocks connected with each other or to the periphery. Two floorplanning examples have been given.

The primary purpose of SWARM –i.e., tackling practical *place-and-route* problems– has been demonstrated on a Symmetric P-Input OTA and a Folded Cascode P-Input OTA circuit. The outline of the available layout space as well as the basic module arrangement (predefined via a placement template) are considered explicitly during the interaction while each module simultaneously takes care of its innate design requirements implicitly. Another mentionable attribute of SWARM is that it refrains from randomization and works completely *deterministic*.

Producing various OTA layouts with different aspect ratios depicts that SWARM combines the versatility of optimization algorithms with the layout quality of procedural generators. For circuits of such magnitude (denoted as *advanced modules*), it is of practical interest that SWARM not only goes strong with regard to layout quality but also surpasses generator-based and optimization-based automation in terms of *design productivity*. In particular, SWARM's benefit increases when targeting multiple different circuit classes, because the basic set of governing modules is largely the same and thus profits from *re-use*.

By teaming the *bottom-up* capabilities of procedural generators with the *top-down* perspective of algorithmic optimization, SWARM's multi-agent approach represents one veritable way of merging the two automation paradigms towards a *bottom-up meets top-down* design flow. With this philosophy in mind, the presented SWARM methodology paves the way to a novel branch of automation, inspiriting entirely new opportunities for future works in EDA research and development.

Research work on SWARM continues as it is meant to become part of a holistic IC design flow on module level, from the functional circuit specification across the generation of a sized schematic diagram to the creation of the physical layout (thus covering the three different *design domains* of the well-known *Y diagram*). Convinced by the results already obtained so far, SWARM itself is about to be integrated into an industrial design flow for ASICs in automotive applications. Further conceptual enhancements of SWARM that have been envisaged are:

- adaptive tightening policies (correlated with the fluency of the self-organization),
- multiple concurrent interaction control organs (handling different circuit parts),
- hierarchically nested interaction flows (for tackling higher-level layout blocks),
- governing modules with learning aptitude (to train expedient interaction maneuvers),
- improving performance and convergence (runtime reduction and independence of start conditions),
- parallelization of module activity via multi-threading (quite obvious for a multi-agent approach),
- user intervention to facilitate real-time human-machine collaboration (as pursued in Industry 4.0).

This continuation of the work on SWARM is strategically motivated by the belief that –in the long run– *bottom-up meets top-down* approaches may turn out to be one essential key for finally closing the persistent *automation gap* in analog layout design.

Vocabulary

One should use common words
to say uncommon things.
Arthur Schopenhauer (German philosopher)

Abbreviations

ADC	**A**nalog-**D**igital **C**onverter	(p. 54)
A-Life	**A**rtificial **Life**	(p. 81)
AMPLE	**A**dvanced **M**ulti-**P**urpose **L**anguag**E**	(p. 50)
API	**A**pplication **P**rogramming **I**nterface	(p. 46)
BAG	**B**erkeley **A**nalog **G**enerator	(p. 52)
CAPABLE	**C**onstraint-**A**dministered **P**Cell-**A**pplying **B**locklevel **L**ayout **E**ngine	(p. 213)
CDPC	**C**onstraint-**D**erived **P**rocedural **C**ommands	(p. 214)
DAC	**D**igital-**A**nalog **C**onverter	(p. 52)
DRC	**D**esign **R**ule **C**heck	(p. 26)
EDA	**E**lectronic **D**esign **A**utomation	(p. 12)
EDPL	**E**xpert **D**esign **P**lan **L**anguage	(p. 235)
ESD	**E**lectro**S**tatic **D**ischarge	(p. 73)
FIPA	**F**oundation for **I**ntelligent **P**hysical **A**gents	(p. 95)
FM	**F**iduccia-**M**attheyses	(p. 187)
FRP	**FR P**Cells	(p. 234)
GEM	**G**eneric **E**ngineering **M**odel	(p. 51)
GOLF	**G**eometric **O**bject **L**ayout **F**ormula	(p. 52)
gPCDS	**g**raphical **PCDS**	(p. 235)
GQL	**G**eometry **Q**uery **L**anguage	(p. 234)
GUI	**G**raphical **U**ser-**I**nterface	(p. 51)
HIPE	**H**ierarchical **I**nstance **P**arameter **E**diting	(p. 234)
IC	**I**ntegrated **C**ircuit	(p. 12)
IDE	**I**ntegrated **D**evelopment **E**nvironment	(p. 52)

IP	**I**ntellectual **P**roperty	(p. 51)
JADE	**J**ava **A**gent **DE**velopment Framework	(p. 95)
KL	**K**ernighan-**L**in	(p. 187)
LVS	**L**ayout **V**ersus **S**chematic	(p. 26)
MCNC	**M**icroelectronics **C**enter of **N**orth **C**arolina	(p. 66)
MOGLAN	**MO**dule **G**enerator **LAN**guage	(p. 51)
MOS	**M**etal-**O**xide-**S**emiconductor	(p. 20)
OTA	**O**perational **T**ransconductance **A**mplifier	(p. 71)
PCDS	**P**arameterized **C**ircuit **D**escription **S**cheme	(p. 234)
PCell	**P**arameterized **Cell**	(p. 48)
PDK	**P**rocess **D**esign **K**it	(p. 20)
PyCell	**Py**thon-based Parameterized **Cell**	(p. 50)
SADA	**S**emi-**A**utomatic **D**esign **A**ssistant	(p. 235)
SDL	**S**chematic-**D**riven **L**ayout	(p. 26)
SKILL	–is not an acronym but a name–	(p. 50)
SWARM	**S**elf-organized **W**iring and **A**rrangement of **R**esponsive **M**odules	(p. 96)
TCL	**T**ool **C**ommand **L**anguage	(p. 52)
UDD	**U**ser-**D**efined **D**evice	(p. 51)
via	**v**ertical **i**nterconnect **a**ccess	(p. 21)

Mathematical Operators

Calculation	
$\sum$	sum
$\prod$	product
$\lvert\ldots\rvert$	absolute value of a scalar
$\lfloor\ldots\rfloor$	floor (rounding down a scalar)
$\lceil\ldots\rceil$	ceiling (rounding up a scalar)
$\min$	smallest of the given values
$\max$	greatest of the given values
Set Theory	
$\{\ldots\}$	set brackets
$\emptyset$	empty set
$\in$	membership
$\notin$	non-membership
$\subseteq$	subset
$\subset$	proper subset
$\supseteq$	superset
$\supset$	proper superset
$\cap$	set intersection
$\cup$	set union
$\setminus$	set complement
$\triangle$	symmetric difference
$\times$	Cartesian product
$\lvert\ldots\rvert$	cardinality of a set
Logic	
$\forall$	universal quantification
$\exists$	existential quantification
$\nexists$	non-existential quantification
$\exists!$	uniqueness quantification
$\wedge$	logical conjunction (AND)
$\vee$	logical disjunction (OR)
$\oplus$	exclusive disjunction (XOR)
$\Leftrightarrow$	if and only if
$\square$	quod erat demonstrandum (end of proof)
Thesis-specific	
$\diamond$	commutative order of two morphisms
$\triangleright$	noncommutative order of two morphisms
$\succ$	succession of two morphisms
$\Leftarrow$	mapping from parameter domain to layout variability
⧾	set plus single element
⧿	set minus single element

Geometrical Operators

Domain	
$\mathbb{U}$	entire layout plane (geometrical *universe*)
$\varnothing$	geometrical void ($\varnothing = \overline{\mathbb{U}}$)
Membership	
$G_1 \sqsubseteq G_2$	G_1 is enclosed by G_2 (geometrical inclusion)
$G_1 \sqsubset G_2$	G_1 is properly enclosed by G_2 (strict geometrical inclusion)
$G_1 \sqsupseteq G_2$	G_1 encloses G_2 (geometrical containment)
$G_1 \sqsupset G_2$	G_1 properly encloses G_2 (strict geometrical containment)
Construction	
$G = ((\check{x}, \check{y}), (\hat{x}, \hat{y}))$	rectangle from south-western vertex $(\check{x}, \check{y})$ to north-eastern vertex $(\hat{x}, \hat{y})$
$G = (N_1, N_2, \dots)$	polygonal shape, given as a polygonal chain with sequential nodes $N_1, N_2, \dots$
$G_1 \sqcap G_2$	intersection of G_1 and G_2 (geometrical AND)
$G_1 \sqcup G_2$	union of G_1 and G_2 (geometrical OR)
$G_1 \mathbin{\lrcorner} G_2$	relative complement of G_2 in G_1 (geometrical difference)
$\overline{G}$	absolute complement of G ($\overline{G} = \mathbb{U} \mathbin{\lrcorner} G$)
Outline Functions	
$\vdash G$	horizontal coordinate of the leftmost point in G
$\dashv G$	horizontal coordinate of the rightmost point in G
$\top G$	vertical coordinate of the uppermost point in G
$\bot G$	vertical coordinate of the lowermost point in G
⬚G	rectangular bounding box of geometrical shape G
⬚⬚$\mathcal{G}$	rectangular bounding boxes of geometrical shapes $\mathcal{G}$
$\circledcirc_\varepsilon G$	enlargement of G's contour in the four cardinal directions by ε (*grow operator*)
$\circledast_\xi G$	relative shrinking of G's contour by a factor of ξ (*contraction operator*)
Measurement	
$\leftrightarrow G$	width of G
$\updownarrow G$	height of G
$\lceil G \rfloor$	area of G
$\lvert G \rvert$	number of vertices of G
$\overline{L_1 L_2}$	Euclidean distance between two points L_1 and L_2

Symbols

α	angle in the orientation of a design component (counterclockwise)	(p. 108)
a	general index	(p. 106)
A	adoption process of a governing module	(p. 101)
Ab	absorption operation of a module's adoption process	(p. 101)
Ad	adaptation operation of a module's adoption process	(p. 101)
Am	amendment operation of a module's adoption process	(p. 101)
As	assimilation operation of a module's adoption process	(p. 101)
β	exponentiation base in a logistic regressive tightening policy	(p. 175)
b	general index	(p. 106)
B	bounding box around the part of a participant inside the layout zone	(p. 129)
$\mathbb{B}$	Boolean domain $\{0, 1\}$	(p. 128)
γ	number of clashes between two participants	(p. 119)
c_s	solid comfort padding amount	(p. 183)
c_ξ	volatile comfort padding factor	(p. 184)
c_v	volatile comfort padding share	(p. 184)
C	connection between two participants	(p. 119)
$\mathcal{C}$	set of connections of a participant	(p. 121)
d	ductility of a design component	(p. 114)
Δ	uppercase delta (absolute difference)	(p. 132)
δ	lowercase delta (relative change)	(p. 148)
D	deformation morphism of a module transformation	(p. 106)
$\mathbb{D}_{P,I}$	parameter domain of an input parameter I for a module P	(p. 111)
ε	arbitrarily small positive number	(p. 130)
ϵ	absolute edge displacement due to a tightening or enlargement of the layout zone	(p. 157)
e	emphasis of a connection between two participants	(p. 119)
E	edge of a rectangle or polygon	(p. 129)
$\mathtt{f}$	number of fingers parameter of a procedural generator for a MOS transistor	(p. 111)
$\mathfrak{f}$	number of fingers value of a procedural generator instance for a MOS transistor	(p. 111)
F	constellation frame around all participants	(p. 157)
g	(introversive) behavior of a procedural generator	(p. 48)
G	geometrical shape in the layout design	(p. 118)
$\mathcal{G}$	set of geometrical shapes	(p. 118)
η	number of connections of a participant	(p. 120)
$\bar{h}$	horizontal flipping in the orientation of a design component	(p. 108)
h	height of an object	(p. 135)
H	hard constraint (strict confinement) imposed on a set of constraint members	(p. 128)
$\mathcal{H}$	set of hard constraints imposed on a set of constraint members	(p. 128)
ϑ	aversion between two participants	(p. 117)
θ	tension contributing to the turmoil of a participant	(p. 122)
Θ	turmoil of a participant	(p. 122)
i	general counter	(p. 114)
I	input parameter of a procedural generator	(p. 48)
$\mathcal{I}$	set of input parameters of a procedural generator	(p. 48)
j	general counter	(p. 114)
$\varkappa$	kickoff enlargement multiplier (desired zone size divided by intrinsic minimum)	(p. 158)
k	general counter	(p. 114)
K	corridor for a participant's perception of its free peripheral space	(p. 129)
$\mathcal{K}$	set of corridors for a participant's perception of its free peripheral space	(p. 129)
λ	leeway coefficient for the calculation of a connection's relaxation threshold	(p. 120)
ℓ	layout layer in a semiconductor technology	(p. 118)

Λ	set of layout layers in a semiconductor technology	(p. 118)
l	length (Euclidean distance between two points)	(p. 121)
L	location (x, y) of a design component / point in the layout plane	(p. 108)
$\mathcal{L}$	set of locations	(p. 135)
$\mathbb{L}$	the universe of layout designs	(p. 48)
m	minimal movement distance to prevent infinitesimal actions	(p. 136)
M	movement morphism of a module transformation	(p. 106)
$\mathcal{M}$	constraint members (set of components on which a certain constraint is imposed)	(p. 128)
n	general counter	(p. 114)
N	node (vertex) of a geometrical shape	(p. 154)
$\mathbb{N}^0$	set of natural numbers (including zero)	(p. 125)
$\mathbb{N}^+$	set of positive natural numbers (excluding zero)	(p. 111)
ξ	contraction amount for a relative tightening or enlargement of the layout zone	(p. 157)
Ξ	set of contraction amounts	(p. 165)
o	orientation parameter of a procedural generator	(p. 112)
O	orientation of a design component	(p. 108)
$O_{\bar{h}}$	orientation of a design component in horizontal notation	(p. 109)
O_{υ}	orientation of a design component in vertical notation	(p. 109)
$\mathbb{O}$	set of orientations	(p. 151)
ϖ	pressing rate (to define the rigorousness of the layout zone tightening policy)	(p. 161)
p	positioning parameter of a procedural generator	(p. 112)
P	parameterized design component / procedural generator / participant	(p. 112)
$\mathcal{P}$	set of parameterized design components / procedural generators / participants	(p. 114)
$\mathcal{P}_{\iota}$	participants that are involved in an action	(p. 132)
q	contraction quotient in an exponential progressive tightening policy	(p. 171)
Q	square geometrical shape (regular quadrilateral)	(p. 120)
ϱ	relaxation threshold of a connection between two participants	(p. 120)
ρ	region of a wound on a participant	(p. 125)
r	radius of a circle	(p. 120)
R	rotation morphism of a module transformation	(p. 106)
$\mathbb{R}$	set of real numbers	(p. 119)
ς	severity of a wound on a participant	(p. 125)
s	strength of a connection between two participants	(p. 119)
S	free peripheral space of a participant	(p. 129)
$\mathbb{S}$	the universe of schematic circuits	(p. 48)
τ	trouble contributing to the interference of a participant	(p. 117)
t_H	constraint type of a hard design constraint H	(p. 128)
T	transformation of a participant	(p. 132)
$\mathcal{T}$	set of transformations (i.e., an action involving multiple participants)	(p. 150)
$\mathbb{T}$	set of actions (where each action is given as a set of transformations)	(p. 151)
υ	vertical flipping in the orientation of a design component	(p. 109)
Υ	interference of a participant	(p. 117)
u	number of minor tightenings in a linear progressive tightening policy	(p. 169)
$\mathcal{U}$	set of obstacles	(p. 129)
φ	conciliation quota for the decrease of the aversion between two participants	(p. 119)
Φ	morphism (movement, rotation, or deformation) of a module transformation	(p. 106)
$v_t(\mathcal{M})$	verification function for a constraint of type t imposed on constraint members $\mathcal{M}$	(p. 128)
V	layout variant of a design component	(p. 132)
$\mathcal{V}$	variability of a design component	(p. 111)
$\acute{\mathcal{V}}$	intrinsic variability of a design component	(p. 111)
$\widetilde{\mathcal{V}}$	cumulative variability of a design component	(p. 112)
$\mathfrak{w}_{\mathrm{ch}}$	total channel width of a MOS transistor	(p. 113)

$\mathfrak{w}_{\text{min}}$	minimum channel width of a MOS transistor	(p. 113)
w	width of an object	(p. 135)
W	wound of a participant	(p. 125)
$\mathcal{W}$	set of wounds of a participant	(p. 127)
χ	projection divisor for tightening or enlarging a nonrectangular layout zone	(p. 166)
x	horizontal coordinate of a component's location or of a point in the layout plane	(p. 108)
X	exploration scenario in a participant's action exploration plan	(p. 154)
$\mathcal{X}$	exploration plan for a participant's action exploration	(p. 154)
ψ	protrusion extent of a participant's protrusion	(p. 124)
Ψ	protrusion of a participant	(p. 123)
ω	overlap between two participants	(p. 117)
y	vertical coordinate of a component's location or of a point in the layout plane	(p. 108)
Y	yielding region around a participant, used to determine a *Yielding* action	(p. 141)
ζ	number of a participant's unrelaxed connections	(p. 122)
z, z^*	zone size quotients in a logistic regressive tightening policy	(p. 174)
Z	outline of the user-defined layout zone	(p. 98)

Index

Meaning of the superscript signs:

*	indicates a term already established in electronic design automation
**	indicates a term already established in the field of complex systems
***	indicates a term already established in physics, math, or computer science
†	indicates a term coined in publications accompanying the work of this thesis
‡	indicates a term specific to the SWARM methodology as covered in this thesis.

References

A reader lives a thousand lives before he dies.
The man who never reads lives only one.
George R. R. Martin (US-American writer)

Bibliography

[1] Alan Hastings, "The Art of Analog Layout", Second Edition, Pearson Prentice Hall, New Jersey, 2006, ISBN: 978-0-13-146410-0.

[2] Jef Rijmenants / James B. Litsios / Thomas R. Schwarz / Marc G. R. Degrauwe, "ILAC: An Automated Layout Tool for Analog CMOS Circuits", *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 417–425, Apr. 1989, DOI: 10.1109/4.18603.

[3] Masato Mogaki / Naoki Kato / Youko Chikami / Naoyuki Yamada / Yasuhiro Kobayashi, "LADIES: An Automatic Layout System for Analog LSI's", *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 450–453, Nov. 1989, DOI: 10.1109/ICCAD.1989.76989.

[4] Volker Meyer zu Bexten / Claudio Moraga / Roland Klinke / Werner Brockherde / Klaus-Gunther Hess, "ALSYN: Flexible Rule-Based Layout Synthesis for Analog IC's", *IEEE Journal of Solid-State Circuits*, vol. 28, no. 3, pp. 261–268, Mar. 1993, DOI: 10.1109/4.209992.

[5] Alberto Luigi Sangiovanni-Vincentelli, "The Tides of EDA", *IEEE Design and Test of Computers*, vol. 20, no. 6, pp. 59–75, Nov./Dec. 2003, DOI: 10.1109/MDT.2003.1246165.

[6] R. Colin Johnson, "Analog EDA Finally Automated", *online*, Apr. 2015, quotation from Rob A. Rutenbar at the International Symposium on Physical Design, URL: `http://www.eetimes.com/document.asp?doc_id=1326192`.

[7] Jürgen Scheible, "Constraint-Driven Design – Eine Wegskizze zum Designflow der nächsten Generation", *Proc. of ANALOG 2008*, pp. 153–158, VDE Verlag, Berlin, Offenbach, Apr. 2008, ISBN: 978-3800730834.

[8] Jürgen Scheible, "Layoutentwurf integrierter Schaltkreise", *Skriptum zur Vorlesung*, Sep. 2015, Chapter 7.

[9] Franco Maloberti, "Layout of Analog CMOS Integrated Circuit", *online*, Part 2: Transistors and Basic Cells Layout, URL: `http://ims.unipv.it/Courses/download/AIC/Layout02.pdf`.

[10] Jens Lienig, "Layoutsynthese elektronischer Schaltungen – Grundlegende Algorithmen für die Entwurfsautomatisierung", Springer, Berlin Heidelberg New York, 2006, ISBN: 978-3-540-29627-0, DOI: 10.1007/3-540-29942-4.

[11] "Analog Layout Synthesis – A Survey of Topological Approaches" (edited by Helmut E. Gräb), Springer, New York Dordrecht Heidelberg London, 2011, ISBN: 978-1-4419-6931-6, DOI: 10.1007/978-1-4419-6932-3.

[12] Rob A. Rutenbar, "Analog Layout Synthesis: What's Missing?", *Proc. of 19th International Symposium on Physical Design*, p. 43, Mar. 2010, ISBN: 978-1-60558-920-6, DOI: 10.1145/1735023.1735037.

[13] Harald Bauer / Felix Grawert / Nadine Kammerlander / Ulrich Naeher / Florian Weig, "Getting Mo(o)re out of Semiconductor R&D", *McKinsey on Semiconductors*, vol. 1, no. 1, pp. 59–65, Sep. 2011, URL: `http://www.mckinsey.com/~/media/mckinsey/dotcom/client_service/semiconductors/pdfs/mosc1rd.ashx`.

[14] Andrew B. Kahng / Jens Lienig / Igor L. Markov / Jin Hu, "VLSI Physical Design: From Graph Partitioning to Timing Closure", Springer Netherlands, 2011, ISBN: 978-90-481-9590-9, DOI: 10.1007/978-90-481-9591-6.

[15] Gordon E. Moore, "Cramming More Components onto Integrated Circuits", *Proc. of Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998, reprinted from *Electronics*, pp. 114–117, April 19, 1965, DOI: 10.1109/JPROC.1998.658762.

[16] Wolfgang Arden / Michel Brillouët / Patrick Cogez / Mart Graef / Bert Huizing / Reinhard Mahnkopf, "'More-than-Moore' White Paper", *Proc. of International Technical Roadmap for Semiconductors*, pp. 1–31, 2010, URL: `http://www.itrs2.net/uploads/4/9/7/7/49775221/irc-itrs-mtm-v2_3.pdf`.

[17] "The Electronic Design Automation Handbook" (edited by Dirk Jansen *et al.*), Springer, New York, Feb. 2010, ISBN: 978-1-4419-5369-8, DOI: 10.1007/978-0-387-73543-6.

[18] Erich Barke / Markus Olbrich / Lars Hedrich *et al.*, "Electronic Design Automation (EDA)", *online*, Section 3.1.5, URL: `http://edascript.ims.uni-hannover.de/de/index.html`.

[19] Dan Clein, "What is Full Custom Layout Design", *online*, Jun. 2001, URL: `http://www.eetimes.com/document.asp?doc_id=1277368`.

[20] Thomas Schiex, "Possibilistic Constraint Satisfaction Problems or 'How to Handle Soft Constraints?'", *Proc. of 8th International Conference on Uncertainty in Artificial Intelligence*, pp. 268–275, 1992, URL: `https://arxiv.org/ftp/arxiv/papers/1303/1303.5427.pdf`, ISBN: 1-55860-258-5.

[21] C. Domshlak / S. Prestwich / F. Rossi / K. B. Venable / T. Walsh, "Hard and Soft Constraints for Reasoning About Qualitative Conditional Preferences", *Journal of Heuristics*, vol. 12, no. 4–5, pp. 263–285, 2006, DOI: 10.1007/s10732-006-7071-x.

[22] B. J. Hicks / A. J. Medland / G. Mullineux, "The Representation and Handling of Constraints for the Design, Analysis and Optimization of High Speed Machinery", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 20, no. 4, pp. 313–328, Nov. 2006, DOI: 10.1017/S0890060406060239.

[23] Göran Jerke / Jens Lienig, "Constraint-Driven Design: The Next Step Towards Analog Design Automation", *Proc. of 18th International Symposium on Physical Design*, pp. 75–82, Mar. 2009, ISBN: 978-1-60558-449-2, DOI: 10.1145/1514932.1514952.

[24] Cadence Design Systems, Inc., "Virtuoso Unified Custom Constraints User Guide, Chapter 1: The Constraint Manager Assistant", *Virtuoso Layout Suite*, Aug. 2015, Product Version 6.1.6.

[25] Rob A. Rutenbar, “Design Automation for Analog: The Next Generation of Tool Challenges”, *1st IBM Academy Conference on Analog Design, Technology, Modeling and Tools*, Sep. 2006, Slide 13, URL: `http://rutenbar.cs.illinois.edu/wp-content/uploads/2012/10/rutenbar-ibm06.pdf`.

[26] Rob A. Rutenbar, “Design Automation for Analog: The Next Generation of Tool Challenges”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 458–460, Nov. 2006, DOI: 10.1109/ICCAD.2006.320157.

[27] Rob A. Rutenbar, “Analog CAD: Not Done Yet”, *National Science Foundation (NSF) Workshop: Electronic Design Automation – Past, Present, and Future*, Jul. 2009, Slide 6, URL: `http://cadlab.cs.ucla.edu/nsf09/slides/Session3/rutenbar.pdf`.

[28] Jürgen Scheible / Daniel Marolt, “Der analoge Entwurfsfluss mit parametrisierten Schaltungsmodulen”, *Fachgruppe Layoutentwurf*, Sep. 2012, Fachgruppentreffen Dortmund.

[29] Larry Stockmeyer, “Optimal Orientations of Cells in Slicing Floorplan Designs”, *Information and Control*, vol. 57, no. 2–3, pp. 91–101, May 1983, DOI: 10.1016/S0019-9958(83)80038-2.

[30] C. K. Kim / E. Berkcan / B. Currin / M. d’Abreu, “A New Floorplanning Algorithm for Analog Circuits”, *Proc. of Custom Integrated Circuits Conference*, pp. 3.2.1–3.2.4, May 1989, DOI: 10.1109/CICC.1989.56679.

[31] L. Paris / G. Berbel / T. Osés, “Floorplanning Strategy for Mixed Analog-Digital VLSI Integrated Circuits”, *Proc. of European Conference on Design Automation*, pp. 346–350, Feb. 1991, DOI: 10.1109/EDAC.1991.206422.

[32] Nasir-ud-Din Gohar / Peter Y. K. Cheung, “A New Schematic-Driven Floorplanning Algorithm for Analog Cell Layout”, *Proc. of IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 1770–1773, May 1993, DOI: 10.1109/ISCAS.1993.394087.

[33] Saurabh N. Adya / Igor L. Markov, “Fixed-Outline Floorplanning: Enabling Hierarchical Design”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003, DOI: 10.1109/TVLSI.2003.817546.

[34] Saurabh N. Adya / Shubhyant Chaturvedi / Jarrod A. Roy / David A. Papa / Igor L. Markov, “Unification of Partitioning, Placement and Floorplanning”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 550–557, Nov. 2004, DOI: 10.1109/ICCAD.2004.1382639.

[35] Yan Feng / Dinesh P. Mehta / Hannah Yang, “Constrained Floorplanning Using Network Flows”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 572–580, Apr. 2004, DOI: 10.1109/TCAD.2004.825877.

[36] Tung-Chieh Chen / Yao-Wen Chang, “Modern Floorplanning Based on B*-Tree and Fast Simulated Annealing”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 4, pp. 637–650, Apr. 2006, DOI: 10.1109/TCAD.2006.870076.

[37] Peter G. Sassone / Sung Kyu Lim, “Traffic: A Novel Geometric Algorithm for Fast Wire-Optimized Floorplanning”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1075–1086, Jun. 2006, DOI: 10.1109/TCAD.2005.855921.

[38] Maolin Tang / Raymond Y. K. Lau, “A Parallel Genetic Algorithm for Floorplan Area Optimization”, *Proc. of 7th International Conference on Intelligent Systems Design and Applications*, pp. 801–806, Oct. 2007, DOI: 10.1109/ISDA.2007.47.

[39] Jianli Chen / Wenxing Zhu / M. M. Ali, “A Hybrid Simulated Annealing Algorithm for Nonslicing VLSI Floorplanning”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 41, no. 4, pp. 544–554, Jul. 2011, DOI: 10.1109/TSMCC.2010.2066560.

[40] Prabhjit Kaur, “An Enhanced Algorithm for Floorplan Design Using Hybrid Ant Colony and Particle Swarm Optimization”, *International Journal for Research in Applied Science and Engineering Technology*, vol. 2, no. 9, pp. 473–477, Sep. 2014, ISSN: 2321-9653.

[41] Hongxia Zhou / Chiu-Wing Sham / Hailong Yao, “Slicing Floorplans with Handling Symmetry and General Placement Constraints”, *Proc. of IEEE Computer Society Annual Symposium on VLSI*, pp. 112–117, Jul. 2014, DOI: 10.1109/ISVLSI.2014.62.

[42] Jai-Ming Lin / Chih-Yao Hu / Kai-Chung Chan, “Routability-Driven Floorplanning Algorithm for Mixed-Size Modules with Fixed-Outline Constraint”, *Proc. of International Symposium on VLSI Design, Automation and Test*, pp. 1–4, Apr. 2015, DOI: 10.1109/VLSI-DAT.2015.7114531.

[43] Ralph H. J. M. Otten, “Automatic Floorplan Design”, *Proc. of 19th Design Automation Conference*, pp. 261–267, Jun. 1982, DOI: 10.1109/DAC.1982.1585510.

[44] S. Kirkpatrick / C. D. Gelatt Jr. / M. P. Vecchi, “Optimization by Simulated Annealing”, *Science*, vol. 220, no. 4598, pp. 671–680, May 1983, DOI: 10.1126/science.220.4598.671.

[45] Neil R. Quinn Jr., “The Placement Problem as Viewed from the Physics of Classical Mechanics”, *Proc. of 12th Design Automation Conference*, pp. 173–178, 1975.

[46] Melvin A. Breuer, “A Class of Min-Cut Placement Algorithms”, *Proc. of 14th Design Automation Conference*, pp. 284–290, 1977.

[47] Sheqin Dong / Zhe Zhou / Xianlong Hong, “A New Constraint-Driven Placement Approach for Analog Circuits”, *Proc. of 8th International Conference on Solid-State and Integrated Circuit Technology*, pp. 1763–1765, Oct. 2006, DOI: 10.1109/ICSICT.2006.306419.

[48] Yiu-Cheong Tam / Evangeline F. Y. Young / Chris Chu, “Analog Placement with Symmetry and Other Placement Constraints”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 349–354, Nov. 2006, DOI: 10.1109/ICCAD.2006.320057.

[49] Shinichi Koda / Chikaaki Kodama / Kunihiro Fujiyoshi, “Linear Programming-Based Cell Placement with Symmetry Constraints for Analog IC Layout”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 4, pp. 659–668, Apr. 2007, DOI: 10.1109/TCAD.2007.891365.

[50] Po-Hung Lin / Shyh-Chang Lin, “Analog Placement Based on Novel Symmetry-Island Formulation”, *Proc. of 44th Design Automation Conference*, pp. 465–470, Jun. 2007, ISBN: 978-1-59593-627-1.

[51] Po-Hung Lin / Shyh-Chang Lin, “Analog Placement Based on Hierarchical Module Clustering”, *Proc. of 45th Design Automation Conference*, pp. 50–55, Jun. 2008, ISBN: 978-1-60558-115-6.

[52] Martin Strasser / Michael Eick / Helmut Gräb / Ulf Schlichtmann / Frank M. Johannes, “Deterministic Analog Circuit Placement Using Hierarchically Bounded Enumeration and Enhanced Shape Functions”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 306–313, Nov. 2008, DOI: 10.1109/ICCAD.2008.4681591.

[53] Rui He / Lihong Zhang, “Analog Placement Design with Constraints of Multiple Symmetry Groups”, *Proc. of Canadian Conference on Electrical and Computer Engineering*, pp. 1204–1207, May 2009, DOI: 10.1109/CCECE.2009.5090316.

[54] Po-Hung Lin / Yao-Wen Chang / Shyh-Chang Lin, “Analog Placement Based on Symmetry-Island Formulation”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 6, pp. 791–804, Jun. 2009, DOI: 10.1109/TCAD.2009.2017433.

[55] Linfu Xiao / Evangeline F. Y. Young, “Analog Placement with Common Centroid and 1-D Symmetry Constraints”, *Proc. of 14th Asia and South Pacific Design Automation Conference*, pp. 353–360, Jan. 2009, DOI: 10.1109/ASPDAC.2009.4796506.

[56] Cheng-Wu Lin / Jai-Ming Lin / Chun-Po Huang / Soon-Jyh Chang, “Performance-Driven Analog Placement Considering Boundary Constraint”, *Proc. of 47th Design Automation Conference*, pp. 292–297, Jun. 2010, ISBN: 978-1-4503-0002-5.

[57] Lingyi Zhang / Sheqin Dong / Yuchun Ma / Xianlong Hong, “Multi-Stage Analog Placement with Various Constraints”, *Proc. of International Conference on Communications, Circuits and Systems*, pp. 881–885, Jul. 2010, DOI: 10.1109/ICCCAS.2010.5581851.

[58] Cheng-Wu Lin / Cheng-Chung Lu / Chun-Po Huang / Soon-Jyh Chang / Jai-Ming Lin, “Routing-Aware Placement Algorithms for Modern Analog Integrated Circuits”, *Proc. of 54th IEEE International Midwest Symposium on Circuits and Systems*, pp. 1–4, Aug. 2011, DOI: 10.1109/MWSCAS.2011.6026537.

[59] Qiang Ma / Linfu Xiao / Yiu-Cheong Tam / Evangeline F. Y. Young, “Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 85–95, Jan. 2011, DOI: 10.1109/TCAD.2010.2064490.

[60] Cheng-Wu Lin / Cheng-Chung Lu / Jai-Ming Lin / Soon-Jyh Chang, “Routability-Driven Placement Algorithm for Analog Integrated Circuits”, *Proc. of International Symposium on Physical Design*, pp. 71–78, Mar. 2012, DOI: 10.1145/2160916.2160934.

[61] Hongxia Zhou / Chiu-Wing Sham / Hailong Yao, “Congestion-Oriented Approach in Placement for Analog and Mixed-Signal Circuits”, *Proc. of 5th Asia Symposium on Quality Electronic Design*, pp. 97–102, Aug. 2013, DOI: 10.1109/ASQED.2013.6643571.

[62] Ricardo Martins / Nuno Lourenço / Nuno Horta, “Analog IC Placement Using Absolute Coordinates and a Hierarchical Combination of Pareto Optimal Fronts”, *Proc. of 11th Conference on Ph.D. Research in Microelectronics and Electronics*, pp. 61–64, Jun./Jul. 2015, DOI: 10.1109/PRIME.2015.7251334.

[63] Sherif M. Saif / Mohamed Dessouky / M. Watheq El-Kharashi / Hazem Abbas / Salwa Nassar, “Pareto Front Analog Layout Placement Using Satisfiability Modulo Theories”, *Proc. of Design, Automation and Test in Europe Conference*, pp. 1411–1416, Mar. 2016, ISBN: 978-3-9815370-7-9.

[64] Po-Hung Lin / Ho-Che Yu / Tian-Hau Tsai / Shyh-Chang Lin, “A Matching-Based Placement and Routing System for Analog Design”, *Proc. of International Symposium on VLSI Design, Automation and Test*, pp. 1–4, Apr. 2007, DOI: 10.1109/VDAT.2007.373200.

[65] Ender Yilmaz / Günhan Dündar, “Analog Layout Generator for CMOS Circuits”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 32–45, Jan. 2009, DOI: 10.1109/TCAD.2008.2009137.

[66] Suchismita Pattanaik / Subhendu Prakash Bhoi / Rakesh Mohanty, “Simulated Annealing Based Placement Algorithms and Research Challenges: A Survey”, *Journal of Global Research in Computer Science*, vol. 3, no. 6, pp. 33–37, Jun. 2012, ISSN: 2229-371X.

[67] C. Y. Lee, “An Algorithm for Path Connections and Its Applications”, *IRE Transactions on Electronic Computers*, vol. EC–10, no. 3, pp. 346–365, Sep. 1961, DOI: 10.1109/TEC.1961.5219222.

[68] David W. Hightower, “A Solution to Line-Routing Problems on the Continuous Plane”, *Proc. of 6th Design Automation Conference*, pp. 1–24, 1969, DOI: 10.1145/800260.809014.

[69] Jens Lienig / Göran Jerke / Thorsten Adler, “AnalogRouter: A New Approach of Current-Driven Routing for Analog Circuits”, *Proc. of Design, Automation and Test in Europe Conference*, p. 819, Mar. 2001, DOI: 10.1109/DATE.2001.915167.

[70] Changxu Du / Yici Cai / Xianlong Hong, “A Novel Analog Routing Algorithm with Constraints of Variable Wire Widths”, *Proc. of International Conference on Communications, Circuits and Systems*, pp. 2459–2463, Jun. 2006, DOI: 10.1109/ICCCAS.2006.285173.

[71] Muhammet Mustafa Ozdal / Renato Fernandes Hentschke, “Exact Route Matching Algorithms for Analog and Mixed Signal Integrated Circuits”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 231–238, Nov. 2009, DOI: 10.1145/1687399.1687442.

[72] Qiang Gao / Yin Shen / Yici Cai / Hailong Yao, “Analog Circuit Shielding Routing Algorithm Based on Net Classification”, *Proc. of IEEE International Symposium on Low-Power Electronics and Design*, pp. 123–128, Aug. 2010, DOI: 10.1145/1840845.1840872.

[73] Qiang Gao / Hailong Yao / Qiang Zhou / Yici Cai, “A Novel Detailed Routing Algorithm with Exact Matching Constraint for Analog and Mixed Signal Circuits”, *Proc. of 12th International Symposium on Quality Electronic Design*, pp. 1–6, Mar. 2011, DOI: 10.1109/ISQED.2011.5770700.

[74] Ricardo Martins / Nuno Lourenço / Nuno Horta, “Multi-Objective Multi-Constraint Routing of Analog ICs Using a Modified NSGA-II Approach”, *Proc. of International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, pp. 65–68, Sep. 2012, DOI: 10.1109/SMACD.2012.6339418.

[75] Hailong Yao / Yici Cai / Qiang Gao, “LEMAR: A Novel Length Matching Routing Algorithm for Analog and Mixed Signal Circuits”, *Proc. of 17th Asia and South Pacific Design Automation Conference*, pp. 157–162, Jan./Feb. 2012, DOI: 10.1109/ASPDAC.2012.6164937.

[76] Hung-Chih Ou / Hsing-Chih Chang Chien / Yao-Wen Chang, “Nonuniform Multilevel Analog Routing with Matching Constraints”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1942–1954, Dec. 2014, DOI: 10.1109/TCAD.2014.2363394.

[77] Muhammet Mustafa Ozdal / Renato Fernandes Hentschke, “Algorithms for Maze Routing with Exact Matching Constraints”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 101–112, Jan. 2014, DOI: 10.1109/TCAD.2013.2279516.

[78] Chia-Yu Wu / Helmut Gräb / Jiang Hu, “A Pre-Search Assisted ILP Approach to Analog Integrated Circuit Routing”, *Proc. of 33rd IEEE International Conference on Computer Design*, pp. 244–250, Oct. 2015, DOI: 10.1109/ICCD.2015.7357110.

[79] Michael Burstein / Richard Pelavin, “Hierarchical Wire Routing”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 2, no. 4, pp. 223–234, Oct. 1983, DOI: 10.1109/TCAD.1983.1270040.

[80] William A. Dees Jr. / Patrick G. Karger, “Automated Rip-up and Reroute Techniques”, *Proc. of 19th Design Automation Conference*, pp. 432–439, Jun. 1982, DOI: 10.1109/DAC.1982.1585535.

[81] Pedram Khademsameni / Marek Syrzycki, “A Tool for Automated Analog CMOS Layout Module Generation and Placement”, *Proc. of Canadian Conference on Electrical and Computer Engineering*, vol. 1, pp. 416–421, May 2002, DOI: 10.1109/CCECE.2002.1015261.

[82] Yen-Tai Lai / Yung-Chuan Jiang / Chi-Chou Kao, “DTA: Layout Design Tool for CMOS Analog Circuit”, *Proc. of IEEE Asia-Pacific Conference on Circuits and Systems*, vol. 1, pp. 537–540, Dec. 2004, DOI: 10.1109/APCCAS.2004.1412817.

[83] Di Long / Yijie Zeng / Changxu Du / Xianlong Hong / Sheqin Dong, “A Novel Performance-Driven Automatic Layout Tool for Analog Circuit”, *Proc. of International Conference on Communications, Circuits and Systems*, vol. 2, pp. 1344–1348, Jun. 2004, DOI: 10.1109/ICCCAS.2004.1346420.

[84] Nuno Lourenço / Nuno Horta, “LAYGEN – An Evolutionary Approach to Automatic Analog IC Layout Generation”, *Proc. of 12th IEEE International Conference on Electronics, Circuits and Systems*, pp. 1–4, Dec. 2005, DOI: 10.1109/ICECS.2005.4633414.

[85] Ricardo Martins / Nuno Lourenço / Nuno Horta, “LAYGEN II: Automatic Analog ICs Layout Generator Based on a Template Approach”, *Proc. of 14th Conference on Genetic and Evolutionary Computation*, pp. 1127–1134, Jul. 2012, DOI: 10.1145/2330163.2330319.

[86] Nuno Lourenço / Nuno Horta, “GENOM-POF: Multi-Objective Evolutionary Synthesis of Analog ICs with Corners Validation”, *Proc. of 14th Conference on Genetic and Evolutionary Computation*, pp. 1119–1126, Jul. 2012, DOI: 10.1145/2330163.2330318.

[87] R. Martins / N. Lourenço / S. Rodrigues / J. Guilherme / N. Horta, “AIDA: Automated Analog IC Design Flow from Circuit Level to Layout”, *Proc. of International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, pp. 29–32, Sep. 2012, DOI: 10.1109/SMACD.2012.6339409.

[88] Lihong Zhang / Ulrich Kleine, “A Novel Analog Layout Synthesis Tool”, *Proc. of IEEE International Symposium on Circuits and Systems*, vol. 5, pp. 101–104, May 2004, DOI: 10.1109/ISCAS.2004.1329468.

[89] Lihong Zhang / Ulrich Kleine / Yingtao Jiang, “An Automated Design Tool for Analog Layouts”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 8, pp. 881–894, Aug. 2006, DOI: 10.1109/TVLSI.2006.878475.

[90] Linfu Xiao / Evangeline F. Y. Young / Xiaoyong He / K. P. Pun, “Practical Placement and Routing Techniques for Analog Circuit Designs”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 675–679, Nov. 2010, DOI: 10.1109/ICCAD.2010.5654239.

[91] Yu-Ming Yang / Iris Hui-Ru Jiang, “Analog Placement and Global Routing Considering Wiring Symmetry”, *Proc. of 11th International Symposium on Quality Electronic Design*, pp. 618–623, Mar. 2010, DOI: 10.1109/ISQED.2010.5450510.

[92] Jackey Z. Yan / Chris Chu, “DeFer: Deferred Decision Making Enabled Fixed-Outline Floorplanner”, *Proc. of 45th Design Automation Conference*, pp. 161–166, Jun. 2008, DOI: 10.1145/1391469.1391512.

[93] Hung-Chih Ou / Hsing-Chih Chang Chien / Yao-Wen Chang, “Simultaneous Analog Placement and Routing with Current Flow and Current Density Considerations”, *Proc. of 50th Design Automation Conference*, pp. 1–6, May/Jun. 2013, ISBN: 978-1-4503-2071-9.

[94] Mark Po-Hung Lin / Po-Hsun Chang / Shuenn-Yuh Lee / Helmut E. Gräb, “DeMixGen: Deterministic Mixed-Signal Layout Generation with Separated Analog and Digital Signal Paths”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 8, pp. 1229–1242, Aug. 2016, DOI: 10.1109/TCAD.2015.2501295.

[95] Sambuddha Bhattacharya / Nuttorn Jangkrajarng / C.-J. Richard Shi, “Multilevel Symmetry-Constraint Generation for Retargeting Large Analog Layouts”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 945–960, Jun. 2006, DOI: 10.1109/TCAD.2005.855982.

[96] Michael Eick / Martin Strasser / Kun Lu / Ulf Schlichtmann / Helmut E. Gräb, "Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 2, pp. 180–193, Feb. 2011, DOI: 10.1109/TCAD.2010.2097172.

[97] Po-Cheng Pan / Hung-Ming Chen / Yi-Kan Cheng / Jill Liu / Wei-Yi Hu, "Configurable Analog Routing Methodology via Technology and Design Constraint Unification", *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 620–626, Nov. 2012, ISBN: 978-1-4503-1573-9.

[98] OpenAccess Coalition, "OpenAccess", *online*, URL: `https://projects.si2.org/oac_index.php`.

[99] Enrico Malavasi / Edoardo Charbon, "Constraint Transformation for IC Physical Design", *IEEE Transactions on Semiconductor Manufacturing*, vol. 12, no. 4, pp. 386–395, Nov. 1999, DOI: 10.1109/66.806115.

[100] Bogdan Arsintescu, "Constraint Management and Transformations", Ph.D. Thesis, Delft University of Technology, 1998, ISBN: 90-5326-028-5.

[101] Andreas Krinke / Maximilian Mittag / Göran Jerke / Jens Lienig, "Extended Constraint Management for Analog and Mixed-Signal IC Design", *Proc. of European Conference on Circuit Theory and Design*, pp. 1–4, Sep. 2013, DOI: 10.1109/ECCTD.2013.6662319.

[102] Maximilian Mittag / Andreas Krinke / Göran Jerke / Wolfgang Rosenstiel, "Hierarchical Propagation of Geometric Constraints for Full-Custom Physical Design of ICs", *Proc. of Design, Automation and Test in Europe Conference*, pp. 1471–1474, Mar. 2012, DOI: 10.1109/DATE.2012.6176599.

[103] Andreas Krinke / Göran Jerke / Jens Lienig, "Constraint Propagation Methods for Robust IC Design", *Proc. of 8th GMM/ITG/GI-Symposium Reliability by Design (ZuE)*, pp. 7–14, Sep. 2015, ISBN: 978-3-8007-4071-0.

[104] Jan Freuer / Göran Jerke / Joachim Gerlach / Wolfgang Nebel, "On the Verification of High-Order Constraint Compliance in IC Design", *Proc. of Design, Automation and Test in Europe Conference*, pp. 26–31, Mar. 2008, DOI: 10.1109/DATE.2008.4484655.

[105] Jacques Cohen, "Constraint Logic Programming Languages", *Communications of the ACM*, vol. 33, no. 7, pp. 52–68, Jul. 1990, DOI: 10.1145/79204.79209.

[106] Enrico Malavasi / Edoardo Charbon / Bogdan Arsintescu / William Kao, "A Constraint Management System for IC Physical Design", *Proc. of 11th Brazilian Symposium on Integrated Circuit Design*, pp. 240–243, Oct. 1998, DOI: 10.1109/SBCCI.1998.715450.

[107] Ammar Nassaj, "A New Methodology for Constraint-Driven Layout Design of Analog Circuits", Ph.D. Thesis, Technische Universität Dresden, 2012, ISBN: 978-3-18-342420-7.

[108] Martin Fowler, "Domain-Specific Languages", Addison-Wesley Signature Series, First Edition, Addison-Wesley Professional, New Jersey, Oct. 2010, ISBN: 978-0-321-71294-3.

[109] Timothy J. Barnes, "SKILL: A CAD System Extension Language", *Proc. of 27th Design Automation Conference*, pp. 266–271, Jun. 1990, DOI: 10.1109/DAC.1990.114865.

[110] "Sk2Py", *online*, URL: `http://sk2py.sourceforge.net/`.

[111] Andreas Müller / Jürgen Schattke, "Adapting C++ Based PCells to OpenAccess", *Proc. of CDNLive! Silicon Valley*, 6 pages, Sep. 2005.

[112] John D. Williams, “STICKS – A Graphical Compiler for High Level LSI Design”, *Proc. of National Computer Conference*, pp. 289–295, Jun. 1978, DOI: 10.1109/AFIPS.1978.187.

[113] Warren E. Cory, “Layla: A VLSI Layout Language”, *Proc. of 22nd Design Automation Conference*, pp. 245–251, Jun. 1985, DOI: 10.1109/DAC.1985.1585948.

[114] Didier Lacroix / Sarah Menkis, “An Interactive Graphical Approach to Module Generator Development”, *Proc. of Custom Integrated Circuits Conference*, pp. 30.1.1–30.1.5, May 1990, DOI: 10.1109/CICC.1990.124833.

[115] Cadence Design Systems, Inc., “Virtuoso Parameterized Cell Reference Manual”, *Virtuoso Layout Suite*, Sep. 1992.

[116] M. Wolf / U. Kleine / J. Schulze, “New Description Language and Graphical User Interface for Module Generation in Analog Layouts”, *Proc. of IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 290–293, May/Jun. 1998, DOI: 10.1109/ISCAS.1998.705268.

[117] Markus Wolf, “Konstruktive Layoutgenerierung mit automatischer Neugenerierung unter geänderten Randbedingungen”, Ph.D. Thesis, Otto-von-Guericke-Universität, Magdeburg, 1999, ISBN: 3-8265-6413-8.

[118] Julia Perez / Leo Kasel, “Method and Apparatus for Compiling a Parameterized Cell”, *Patent*, Nov. 2007, Patent Number: US 7296248 B2.

[119] D. Friebel, “Automatic Analog IP Generation with 1Stone”, *MunEDA User-Group-Meeting Europe*, Nov. 2009, URL: `https://www.muneda.com/pdf/mugm/MUGM-Europe-2009-11-12_13.30.pdf`.

[120] SpringSoft, Inc., “Laker User Guide and Tutorial”, *Laker Custom Layout Automation System*, Jul. 2009, Laker 3.2v4p3.

[121] Tanner EDA, “High Performance Device Generation”, *White Paper*, Apr. 2010.

[122] AnaGlobe Technology, Inc., “GOLF PCell Designer”, *Brochure*, 2010, URL: `http://www.anaglobe.com/products/golf/`.

[123] Ciranova, Inc., “Ciranova PyCell Studio Tutorial”, May 2010, now available from Synopsys.

[124] J. Crossley / A. Puggelli / H.-P. Le / B. Yang / R. Nancollas / K. Jung / L. Kong / N. Narevsky / Y. Lu / N. Sutardja / E. J. An / A. L. Sangiovanni-Vincentelli / E. Alon, “BAG: A Designer-Oriented Integrated Framework for the Development of AMS Circuit Generators”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 74–81, Nov. 2013, DOI: 10.1109/ICCAD.2013.6691100.

[125] Benjamin Prautsch / Uwe Eichler / Sunil Rao / Björn Zeugmann / Ajith Puppala / Torsten Reich / Jens Lienig, “IIP Framework: A Tool for Reuse-Centric Analog Circuit Design”, *Proc. of 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, pp. 1–4, Jun. 2016, DOI: 10.1109/SMACD.2016.7520725.

[126] Göran Jerke / Vinko Marolt / Christel Bürzele / Peter Herth / Thomas Burdick, “Visual PCell Programming with Cadence PCell Designer”, *CDNLive! EMEA*, May 2013, Session CUS04, URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/CDNLive/Secured/Proceedings/EU/2013/CUS04.pdf`.

[127] Vinko Marolt / Jürgen Scheible / Ulrich Mauroschat / Matthias Bröckel, “Verfahren zur Generierung von elektronischen Schaltungen”, *Patent Application*, Oct. 2008, Patent Number: DE 102006062563 A1.

[128] Achim Graupner / Roland Jancke / Reimund Wittmann, “Generator Based Approach for Analog Circuit and Layout Design and Optimization”, *Proc. of Design, Automation and Test in Europe Conference*, pp. 1–6, Mar. 2011, DOI: 10.1109/DATE.2011.5763267.

[129] Torsten Reich / Uwe Eichler / Karl-Heinz Rooch / René Buhl, “Design of a 12-bit Cyclic RSD ADC Sensor Interface IC Using the Intelligent Analog IP Library”, *Proc. of ANALOG 2013*, pp. 30–35, Mar. 2013, ISBN: 978-3-8007-3467-2.

[130] Mohannad Elshawy / Mohamed Dessouky / Sherif Saif / Sherif Mansour / Ed Petrus, “Multi-Device Layout Templates for Nanometer Analog Design”, *Proc. of 9th International Design and Test Symposium*, pp. 83–88, Dec. 2014, DOI: 10.1109/IDT.2014.7038592.

[131] Simon Gohm / Daniel Marolt / Jürgen Scheible, “Parametrisierte Layout-Module im analogen IC-Entwurf”, *Proc. of MPC-Workshop*, vol. 48, pp. 57–63, Jul. 2012, ISSN: 1868-9221.

[132] Daniel Marolt / Jürgen Scheible / Göran Jerke, “A Practical Layout Module PCell Concept for Analog IC Design”, *CDNLive! EMEA*, May 2013, Session CUS01, URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/CDNLive/Secured/Proceedings/EU/2013/CUS01.pdf`.

[133] Zhi-Wen Wang / I-Lun Tseng / Adam Postula, “Procedural Module Generation for Parameterized Layouts”, *Proc. of IEEE TENCON Spring Conference*, pp. 548–551, Apr. 2013, DOI: 10.1109/TENCONSpring.2013.6584505.

[134] Zhi-Wen Wang / I-Lun Tseng / Adam Postula, “Design and Representation of Parameterized Layouts for Octagonal Spiral Inductors”, *Proc. of IEEE International Symposium on Next-Generation Electronics*, pp. 333–336, Feb. 2013, DOI: 10.1109/ISNE.2013.6512359.

[135] Thorsten Adler / Jürgen Scheible, “An Interactive Router for Analog IC Design”, *Proc. of Design, Automation and Test in Europe Conference*, pp. 414–420, Feb. 1998, DOI: 10.1109/DATE.1998.655890.

[136] Mircea R. Stan / Fatih Hamzaoglu / David Garrett, “Non-Manhattan Maze Routing”, *Proc. of 17th Brazilian Symposium on Integrated Circuit Design*, pp. 260–265, Sep. 2004, DOI: 10.1109/SBCCI.2004.240979.

[137] Benjamin Prautsch / Uwe Eichler / Torsten Reich / Ajith Puppala / Jens Lienig, “Abstract Technology Handling for Generator-Based Analog Circuit Design”, *Proc. of 8th GMM/ITG/GI-Symposium Reliability by Design (ZuE)*, pp. 56–61, Sep. 2015, ISBN: 978-3-8007-4071-0.

[138] Benjamin Prautsch / Uwe Eichler / Torsten Reich / Jens Lienig, “Explicit Feature and Edge Insertion for Improved Analog Layout Generators in Advanced Semiconductor Technologies”, *Proc. of ANALOG 2016*, pp. 22–27, Sep. 2016, ISBN: 978-3-8007-4265-3.

[139] Vinko Marolt, “AE PCell-Based Layout Generators”, *Robert Bosch GmbH*, Jun. 2007, unpublished presentation, image recreated for reasons of confidentiality.

[140] Göran Jerke / Vinko Marolt / Christel Bürzele / Daniel Marolt / Peter Herth / Thomas Burdick, “Advanced Application of Cadence PCell Designer”, *CDNLive! EMEA*, May 2014, Session CUS06, URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/CDNLive/Secured/Proceedings/EU/2014/CUS06.pdf`.

[141] Karl Sperl, “Ein neuartiger Automatisierungsansatz zur Kombination von Modulgeneratoren mit parametrisierten Verdrahtungs-Zellen im analogen Layoutentwurf”, M.Sc. Thesis, Reutlingen University, Mar. 2014.

[142] Matthias Greif / Daniel Marolt / Jürgen Scheible, “Konzeptstudie eines durchgängig auf parametrisierten Modulgeneratoren basierenden Entwurfsflusses für Analogdesign”, *Proc. of MPC-Workshop*, vol. 53, pp. 23–30, Feb. 2015, ISSN: 1868-9221.

[143] Pranav Bhushan / Raja Mitra, “Schematic PCell Implementation in Virtuoso Platform”, *Proc. of International Cadence Users Group Conference*, 12 pages, Sep. 2004, URL: `http://www.geocities.ws/pranav_bs/docs/33_paper.pdf`.

[144] Pranav Bhushan / Raj Arumugam, “Schematic PCells, Future of Deep Submicron Custom IC Design”, *Proc. of CDNLive! Silicon Valley*, 12 pages, Sep. 2005, URL: `http://www.geocities.ws/pranav_bs/docs/1288_paper.pdf`.

[145] Daniel Marolt / Jürgen Scheible, “Parameterized Cells in Analog IC Design – Example of a Schematic/Symbol Current Mirror PCell”, *Design, Automation and Test in Europe Conference*, Mar. 2012, demo and poster presentation at University Booth.

[146] Eduard Raines, “Rapid Analog Prototyping (RAP): Circuit Prospector and Generic Modgen”, *CDNLive! Boston*, Sep. 2015, Session CUS102, URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/CDNLive/Secured/Proceedings/MA/2015/CUS102.pdf`.

[147] Ender Yilmaz / Günhan Dündar, “New Layout Generator for Analog CMOS Circuits”, *Proc. of 18th European Conference on Circuit Theory and Design*, pp. 36–39, Aug. 2007, DOI: 10.1109/ECCTD.2007.4529530.

[148] Mark Po-Hung Lin / Yao-Wen Chang / Chih-Ming Hung, “Recent Research Development and New Challenges in Analog Layout Synthesis”, *Proc. of 21st Asia and South Pacific Design Automation Conference*, pp. 617–622, Jan. 2016, DOI: 10.1109/ASPDAC.2016.7428080.

[149] Sherif Hammouda / Hazem Said / Mohamed Dessouky / Mohamed Tawfik / Quang Nguyen / Wael Badawy / Hazem Abbas / Hussein Shahein, “Chameleon ART: A Non-Optimization Based Analog Design Migration Framework”, *Proc. of 43rd Design Automation Conference*, pp. 885–888, Jul. 2006, DOI: 10.1145/1146909.1147134.

[150] Zheng Liu / Lihong Zhang, “A Performance-Constrained Template-Based Layout Retargeting Algorithm for Analog Integrated Circuits”, *Proc. of 15th Asia and South Pacific Design Automation Conference*, pp. 293–298, Jan. 2010, DOI: 10.1109/ASPDAC.2010.5419880.

[151] Zheng Liu / Lihong Zhang, “Performance-Constrained Template-Driven Retargeting for Analog and RF Layouts”, *Proc. of 20th Great Lakes Symposium on VLSI*, pp. 429–434, May 2010, DOI: 10.1145/1785481.1785581.

[152] Ching-Yu Chin / Po-Cheng Pan / Hung-Ming Chen / Tung-Chieh Chen / Jou-Chun Lin, “Efficient Analog Layout Prototyping by Layout Reuse with Routing Preservation”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 40–47, Nov. 2013, DOI: 10.1109/ICCAD.2013.6691095.

[153] Yi-Peng Weng / Hung-Ming Chen / Tung-Chieh Chen / Po-Cheng Pan / Chien-Hung Chen / Wei-Zen Chen, “Fast Analog Layout Prototyping for Nanometer Design Migration”, *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 517–522, Nov. 2011, DOI: 10.1109/ICCAD.2011.6105379.

[154] Po-Hsun Wu / Mark Po-Hung Lin / Tsung-Yi Ho, “Analog Layout Synthesis with Knowledge Mining”, *Proc. of European Conference on Circuit Theory and Design*, pp. 1–4, Aug. 2015, DOI: 10.1109/ECCTD.2015.7300111.

[155] Po-Hsun Wu / Mark Po-Hung Lin / Tung-Chieh Chen / Ching-Feng Yeh / Xin Li / Tsung-Yi Ho, "A Novel Analog Physical Synthesis Methodology Integrating Existent Design Expertise", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 199–212, Feb. 2015, DOI: 10.1109/TCAD.2014.2379630.

[156] A. E. Eiben / J. E. Smith, "Introduction to Evolutionary Computing", Natural Computing Series, First Edition, Springer, Berlin Heidelberg, 2003, ISBN: 978-3-642-07285-7.

[157] Lihong Zhang / Changsheng Xie / Xiandeng Pei / Ulrich Kleine, "Analog Module Placement Design Using Genetic Algorithm", *Tsingshua Science and Technology*, vol. 8, no. 2, pp. 161–168, Apr. 2003, ISSN: 1007-0214.

[158] Lihong Zhang / Ulrich Kleine, "A Genetic Approach to Analog Module Placement with Simulated Annealing", *Proc. of IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 345–348, May 2002, DOI: 10.1109/ISCAS.2002.1009848.

[159] James Kennedy / Russell C. Eberhart / Yuhui Shi, "Swarm Intelligence", Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann, San Francisco, CA, USA, 2001, ISBN: 978-1-55860-595-4.

[160] Alberto Colorni / Marco Dorigo / Vittorio Maniezzo, "Distributed Optimization by Ant Colonies", *Proc. of European Conference on Artificial Life*, pp. 134–142, Dec. 1991, URL: `http://www.academia.edu/download/4418203/ic.06-ecal92.pdf`.

[161] James Kennedy / Russell C. Eberhart, "Particle Swarm Optimization", *Proc. of IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, Nov./Dec. 1995, DOI: 10.1109/ICNN.1995.488968.

[162] Smrity Ratan / Debalina Mondal / R. Anima / Chandan Kumar / Amit Kumar / Rajib Kar, "Area Optimization of Two-Stage Amplifier Using Modified Particle Swarm Optimization Algorithm", *Proc. of International Conference on Advances in Computing, Communications and Informatics*, pp. 230–235, Sep. 2016, DOI: 10.1109/ICACCI.2016.7732052.

[163] Meng-Lin Yu, "A Study of the Applicability of Hopfield Decision Neural Nets to VLSI CAD", *Proc. of 26th Design Automation Conference*, pp. 412–417, Jun. 1989, DOI: 10.1109/DAC.1989.203433.

[164] Chen-Xiong Zhang / Andreas Vogt / Dieter A. Mlynski, "Floorplan Design Using a Hierarchical Neural Learning Algorithm", *Proc. of IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 2060–2063, Jun. 1991, DOI: 10.1109/ISCAS.1991.176809.

[165] Rung-Bin Lin / Eugene Shragowitz, "Fuzzy Logic Approach to Placement Problem", *Proc. of 29th Design Automation Conference*, pp. 153–158, Jun. 1992, DOI: 10.1109/DAC.1992.227844.

[166] Frederico Rocha / Ricardo Martins / Nuno Lourenço / Nuno Horta, "Electronic Design Automation of Analog ICs Combining Gradient Models with Multi-Objective Evolutionary Algorithms", Series: SpringerBriefs in Computational Intelligence, First Edition, Springer International Publishing, 2014, ISBN: 978-3-319-02188-1.

[167] Himanshu Gupta / Bahniman Ghosh, "Analog Circuits Design Using Ant Colony Optimization", *International Journal of Electronics, Computer and Communications Technologies*, vol. 2, no. 3, pp. 9–21, Apr. 2012, ISSN: 2180-3536.

[168] P. Prem Kumar / K. Duraiswamy, "An Optimized Device Sizing of Analog Circuits Using Particle Swarm Optimization", *Journal of Computer Science*, vol. 8, no. 6, pp. 930–935, 2012, DOI: 10.3844/jcssp.2012.930.935.

[169] Keven Brown / Eberhard von Kitzing, “Evolution and Bahá’í Belief”, Series: Studies in the Bábi and Bahá’í Religions, vol. 12, Kalimát Press, Los Angeles, CA, 2001, ISBN: 978-1-890688-08-0.

[170] Jürgen Scheible / Jens Lienig, “Automation of Analog IC Layout – Challenges and Solutions”, *Proc. of International Symposium on Physical Design*, pp. 33–40, Mar./Apr. 2015, DOI: 10.1145/2717764.2717781.

[171] Ricardo Martins / Nuno Lourenço / Nuno Horta, “Analog Integrated Circuit Design Automation: Placement, Routing and Parasitic Extraction Techniques”, Springer, Jul. 2016, ISBN: 978-3-319-34059-3.

[172] Ted Manikas, “MCNC Benchmark Netlists for Floorplanning and Placement”, *online*, Jul. 2012, URL: `http://lyle.smu.edu/~manikas/Benchmarks/MCNC_Benchmark_Netlists.html`.

[173] “Analog Benchmarks Reloaded”, *International Workshop on Design Automation for Analog and Mixed-Signal Circuits*, Nov. 2014, Session IV, URL: `https://users.ece.cmu.edu/~xinli/2014_ams/index.html`.

[174] Kazuhiro Oda / Louis A. Prado / Anthony J. Gadient, “A New Methodology for Analog/Mixed-Signal (AMS) SoC Design that Enables AMS Design Reuse and Achieves Full-Custom Performance”, *Proc. of 9th IEEE Electronic Design Processes Workshop*, 6 pages, Apr. 2002, URL: `http://edpsieee.ieeesiliconvalley.org/edp02/PAPERS/edp02-s6_1.pdf`.

[175] Göran Jerke, “PCell Verification”, *Fachgruppe Layoutentwurf*, Feb. 2016, Fachgruppentreffen Böblingen.

[176] Kerstin Langner / Jürgen Scheible, “Formal Verification of a Transistor PCell”, *Proc. of 13th Conference on Ph.D. Research in Microelectronics and Electronics*, pp. 205–208, Jun. 2017, DOI: 10.1109/PRIME.2017.7974143.

[177] Robert Brayton / Jason Cong, “NSF Workshop on EDA: Past, Present, and Future (Part 2)”, *IEEE Design and Test of Computers*, vol. 27, no. 3, pp. 62–74, May/Jun. 2010, DOI: 10.1109/MDT.2010.70.

[178] Andreas Gerlach / Jürgen Scheible / Thoralf Rosahl / Frank-Thomas Eitrich, “A Generic Topology Selection Method for Analog Circuits Demonstrated on the OTA Example”, *Proc. of 11th Conference on Ph.D. Research in Microelectronics and Electronics*, pp. 77–80, Jun./Jul. 2015, DOI: 10.1109/PRIME.2015.7251338.

[179] Yervant Zorian / Dimitris Gizopoulos, “Guest Editor’s Introduction: Design for Yield and Reliability”, *IEEE Design and Test of Computers*, vol. 21, no. 3, pp. 177–182, May/Jun. 2004, DOI: 10.1109/MDT.2004.12.

[180] Francis Heylighen / Carlos Gershenson, “The Meaning of Self-Organization in Computing”, *IEEE Intelligent Systems*, pp. 72–75, May 2003, URL: `http://pcp.vub.ac.be/Papers/IEEE.Self-organization.pdf`.

[181] John Horgan, “From Complexity to Perplexity”, *Scientific American*, vol. 272, no. 6, Jun. 1995.

[182] Werner Ebeling / Jan Freund / Frank Schweitzer, “Komplexe Strukturen: Entropie und Information”, B. G. Teubner Stuttgart, Leipzig, 1998, ISBN: 978-3-322-85167-3.

[183] Carlos Gershenson / Francis Heylighen, “How Can We Think the Complex?”, *Managing Organizational Complexity: Philosophy, Theory and Application* (edited by Kurt Richardson), ch. 3, pp. 47–61, Information Age Publishing, 2005, URL: `https://arxiv.org/abs/nlin/0402023`.

[184] John T. Emlen Jr., “Flocking Behavior in Birds”, *The Auk*, vol. 69, no. 2, pp. 160–170, Apr. 1952, DOI: 10.2307/4081266.

[185] C. C. Trowbridge / H. K. Job, “On the Origin of the Flocking Habit of Migratory Birds”, *The Popular Science Monthly*, vol. 84, pp. 209–217, Mar. 1914, Public Domain content, URL: `http://www.archive.org/stream/popularsciencemo84newy#page/214/mode/1up`.

[186] Randal S. Olson / Arend Hintze / Fred C. Dyer / David B. Knoester / Christoph Adami, “Predator Confusion is Sufficient to Evolve Swarming Behaviour”, *Journal of the Royal Society Interface*, vol. 10, no. 85, pp. 1–8, Aug. 2013, DOI: 10.1098/rsif.2013.0305.

[187] Steven J. Portugal / Tatjana Y. Hubel / Johannes Fritz / Stefanie Heese / Daniela Trobe / Bernhard Voelkl / Stephen Hailes / Alan M. Wilson / James R. Usherwood, “Upwash Exploitation and Downwash Avoidance by Flap Phasing in Ibis Formation Flight”, *Nature*, vol. 505, pp. 399–402, Jan. 2014, DOI: 10.1038/nature12939.

[188] Simeon Andrew Ning, “Aircraft Drag Reduction Through Extended Formation Flight”, Ph.D. Thesis, Stanford University, Department of Aeronautics and Astronautics, Aug. 2011, URL: `https://purl.stanford.edu/dp147ff0571`.

[189] Mitchell G. Ash, “Gestalt Psychology in German Culture, 1890–1967: Holism and the Quest for Objectivity”, Cambridge University Press, Cambridge, United Kingdom, 1995, ISBN: 0-521-47540-6.

[190] Russ Dewey, “Psychology: An Introduction – Chapter Four: Senses (The Whole is Other than the Sum of the Parts)”, *cites F. Heider (1977)*, 2007, URL: `http://www.intropsych.com/ch04_senses/whole_is_other_than_the_sum_of_the_parts.html`.

[191] Harold J. Morowitz, “The Emergence of Everything: How the World Became Complex”, Oxford University Press, USA, 2002, ISBN: 978-0-19-513513-8.

[192] “Self-Organizing Systems: The Emergence of Order” (edited by F. Eugene Yates / Alan Garfinkel / Donald O. Walter / Gregory B. Yates), Series: Life Science Monographs, First Edition, Plenum Press, New York and London, 1987, ISBN: 978-1-4612-8227-3, DOI: 10.1007/978-1-4613-0883-6.

[193] Carl Anderson, “Self-Organization in Relation to Several Similar Concepts: Are the Boundaries to Self-Organization Indistinct?”, *Biological Bulletin*, vol. 202, no. 3, pp. 247–255, Jun. 2002.

[194] “Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering” (edited by Giovanna Di Marzo Serugendo / Anthony Karageorgos / Omer F. Rana / Franco Zambonelli), Series: Lecture Notes in Computer Science, vol. 2977, Springer, Berlin Heidelberg, 2004, ISBN: 978-3-540-21201-0, DOI: 10.1007/b95863.

[195] P. Massioni / M. Verhaegen, “Distributed Control for Identical Dynamically Coupled Systems: A Decomposition Approach”, *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 124–135, Jan. 2009, DOI: 10.1109/TAC.2008.2009574.

[196] Mikhail Prokopenko, “Guided Self-Organization”, *Human Frontier Science Program Journal*, vol. 3, no. 5, pp. 287–289, Oct. 2009, DOI: 10.2976/1.3233933.

[197] Mitchel Resnick, “Decentralized Modeling and Decentralized Thinking”, *Modeling and Simulation in Precollege Science and Mathematics* (edited by W. Feurzeig / N. Roberts), pp. 114–137, Springer, New York, 1999.

[198] Yaneer Bar-Yam, “A Mathematical Theory of Strong Emergence Using Multiscale Variety”, *Complexity*, vol. 9, no. 6, pp. 15–24, Aug. 2004, DOI: 10.1002/cplx.20029.

[199] Mark A. Bedau, "Artificial Life: Organization, Adaptation and Complexity From the Bottom Up", *Trends in Cognitive Sciences*, vol. 7, no. 11, pp. 505–512, Nov. 2003, DOI: 10.1016/j.tics.2003.09.012.

[200] "The MIT Encyclopedia of the Cognitive Sciences" (edited by Frank C. Keil / Robert Andrew Wilson), MIT Press, Massachusetts, 2001, ISBN: 978-0-262-73144-7.

[201] Aristotle, "Metaphysics, Book H 1045a 8–10".

[202] George Henry Lewes, "Problems of Life and Mind", First Series, vol. 2, Trübner, London, 1875, ISBN: 1-4255-5578-0.

[203] Stuart A. Kauffman, "Reinventing the Sacred", Basic Books, New York, NY, USA, 2008, ISBN: 978-0-465-00300-6.

[204] Francis Heylighen, "Self-Organization, Emergence and the Architecture of Complexity", *Proc. of 1st European Conference on System Science*, pp. 23–32, Paris, 1989.

[205] Philip W. Anderson, "More and Different: Notes from a Thoughtful Curmudgeon", World Scientific, London, 2011, ISBN: 978-981-4350-12-9.

[206] James P. Crutchfield, "Is Anything Ever New? Considering Emergence", *Santa Fe Institute Studies in the Sciences of Complexity* (edited by G. Cowan / D. Pines / D. Melzner), Series: Integrative Themes, vol. XIX, Addison-Wesley, Reading, MA, 1994.

[207] Steven B. Johnson, "Only Connect", *online*, Oct. 2001, URL: `http://www.theguardian.com/books/2001/oct/15/society`.

[208] Mark A. Bedau, "Downward Causation and the Autonomy of Weak Emergence", *Principia*, vol. 6, no. 1, pp. 5–50, 2002, published by NEL – Epistemology and Logic Research Group, Federal University of Santa Catarina, Brazil.

[209] Jochen Fromm, "Types and Forms of Emergence", *online*, Jun. 2005, URL: `http://arxiv.org/abs/nlin/0506028`.

[210] Robert B. Laughlin, "A Different Universe: Reinventing Physics from the Bottom Down", Basic Books, New York, NY, USA, Mar. 2005, ISBN: 978-0-465-03828-2.

[211] Mark A. Bedau, "Weak Emergence", *Philosophical Perspectives: Mind, Causation, and World* (edited by James Tomberlin), vol. 11, pp. 375–399, Blackwell Publishers, Oxford, 1997.

[212] William Edward Seager, "Emergence and Supervenience", *online*, URL: `http://www.utsc.utoronto.ca/~seager/emsup.pdf`.

[213] Gerald E. Marsh, "The Demystification of Emergent Behavior", *online*, 2009, URL: `http://arxiv.org/ftp/arxiv/papers/0907/0907.1117.pdf`.

[214] Ernst Mayr, "The Growth of Biological Thought: Diversity, Evolution, and Inheritance", Harvard University Press, 1982, ISBN: 0-674-36446-5.

[215] Anil K. Seth, "Measuring Autonomy and Emergence via Granger Causality", *Artificial Life*, vol. 16, no. 2, pp. 179–196, 2010, DOI: 10.1162/artl.2010.16.2.16204.

[216] Paul C. W. Davies, "Emergent Biological Principles and the Computational Properties of the Universe", *Complexity*, vol. 10, no. 2, pp. 11–15, Nov. 2004, DOI: 10.1002/cplx.20059.

[217] David J. Chalmers, "Strong and Weak Emergence", *The Re-Emergence of Emergence: The Emergent Hypothesis from Science to Religion* (edited by Philip Clayton / Paul Davies), Oxford University Press, Oxford, 2006, ISBN: 978-0199287147.

[218] Noam Miller / Robert Gerlai, "From Schooling to Shoaling: Patterns of Collective Motion in Zebrafish (Danio Rerio)", *PLoS ONE*, vol. 7, no. 11, pp. 1–6, Nov. 2012, DOI: 10.1371/journal.pone.0048865.

[219] Frank Fraser Darling, "A Herd of Red Deer: A Study in Animal Behaviour (Wild Lives)" (edited by Walter Stephen), Luath Press Limited, Edinburgh, Jun. 2008, ISBN: 978-1-906307-42-4.

[220] Chad M. Topaz / Maria R. D'Orsogna / Leah Edelstein-Keshet / Andrew J. Bernoff, "Locust Dynamics: Behavioral Phase Change and Swarming", *PLoS Computational Biology*, vol. 8, no. 8, pp. 1–11, Aug. 2012, DOI: 10.1371/journal.pcbi.1002642.

[221] Bolei Zhou / Xiaogang Wang / Xiaoou Tang, "Understanding Collective Crowd Behaviors: Learning a Mixture Model of Dynamic Pedestrian-Agents", *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2871–2878, Jun. 2012, DOI: 10.1109/CVPR.2012.6248013.

[222] Gordon Firestein (Seacology USA), "School of Jacks", *online*, Jan. 2005, licensed under CC BY-SA 3.0, URL: `https://commons.wikimedia.org/w/index.php?curid=22766988`.

[223] Shyamvs78 (own work), "Spotted Deer Group in Jim Corbett National Park (India)", *online*, Feb. 2008, licensed under CC BY 3.0, URL: `https://commons.wikimedia.org/w/index.php?curid=3521093`.

[224] Waugsberg (own work), "Swarm of Bees in the Air Shortly Before Landing on a Tree", *online*, May 2007, licensed under CC BY-SA 3.0, URL: `https://commons.wikimedia.org/w/index.php?curid=2133292`.

[225] Matt Morgen, "Crowd Driven from Tompkins Square by the Mounted Police in the Tompkins Square Riot of 1874", *Frank Leslie's Illustrated Newspaper*, Jan. 1874, Public Domain content, URL: `https://commons.wikimedia.org/w/index.php?curid=2186990`.

[226] L. David Mech, "Alpha Status, Dominance, and Division of Labor in Wolf Packs", *Canadian Journal of Zoology*, vol. 77, no. 8, pp. 1196–1203, 1999, DOI: 10.1139/z99-099.

[227] Manuele Brambilla / Eliseo Ferrante / Mauro Birattari / Marco Dorigo, "Swarm Robotics: A Review from the Swarm Engineering Perspective", *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, Mar. 2013, DOI: 10.1007/s11721-012-0075-2.

[228] Mehmet Karatay (own work), "Safari Ants on the Chogoria Route of Mount Kenya", *online*, May 2007, licensed under CC BY-SA 3.0, URL: `https://commons.wikimedia.org/w/index.php?curid=2179109`.

[229] S. Goss / S. Aron / J.-L. Deneubourg / J. M. Pasteels, "Self-Organized Shortcuts in the Argentine Ant", *Naturwissenschaften*, vol. 76, no. 12, pp. 579–581, 1989, DOI: 10.1007/BF00462870.

[230] Claire Detrain / Jean-Louis Deneubourg, "Self-Organized Structures in a Superorganism: Do Ants 'Behave' Like Molecules?", *Physics of Life Reviews*, vol. 3, no. 3, pp. 162–187, Sep. 2006, DOI: 10.1016/j.plrev.2006.07.001.

[231] Jon Nelson, "Origin of Diversity in Falling Snow", *Atmospheric Chemistry and Physics*, vol. 8, pp. 5669–5682, Sep. 2008.

[232] Kenneth G. Libbrecht, "The Physics of Snow Crystals", *Reports on Progress in Physics*, vol. 68, pp. 855–895, Mar. 2005, DOI: 10.1088/0034-4885/68/4/R03.

[233] Charles Schmitt (own work), "Stellate Snowflake", *online*, Mar. 2014, licensed under CC BY-SA 4.0, URL: `https://commons.wikimedia.org/w/index.php?curid=44338386`.

[234] A. V. Getling, “Rayleigh-Bénard Convection: Structures and Dynamics”, Advanced Series in Nonlinear Dynamics, vol. 11, World Scientific Publishing, Mar. 1998, ISBN: 978-981-02-2657-2.

[235] A. V. Getling / O. Brausch, “Cellular Flow Patterns and Their Evolutionary Scenarios in Three-Dimensional Rayleigh-Bénard Convection”, *Physical Review E*, vol. 67, no. 4, pp. 1–4, Apr. 2003, DOI: 10.1103/PhysRevE.67.046313.

[236] Heinz Georg Schuster / Wolfram Just, “Deterministic Chaos: An Introduction”, Fourth, Revised and Enlarged Edition, John Wiley & Sons, Weinheim, Jan. 2005, ISBN: 978-3-527-40415-5.

[237] R. B. Levien / S. M. Tan, “Double Pendulum: An Experiment in Chaos”, *American Journal of Physics*, vol. 61, no. 11, pp. 1038–1044, Nov. 1993, DOI: 10.1119/1.17335.

[238] Andy Martin / Kristian Helmerson, “Emergence: The Remarkable Simplicity of Complexity”, *online*, Oct. 2014, URL: `http://theconversation.com/emergence-the-remarkable-simplicity-of-complexity-30973`.

[239] “The Science of Fractal Images” (edited by Heinz-Otto Peitgen / Dietmar Saupe), Springer Verlag, 1988, ISBN: 978-1-4612-8349-2.

[240] Solkoll (own work), “Koch Snowflake”, *online*, Feb. 2005, Public Domain content, URL: `https://commons.wikimedia.org/w/index.php?curid=59048`.

[241] Douglas H. Werner / Suman Ganguly, “An Overview of Fractal Antenna Engineering Research”, *IEEE Antennas and Propagation Magazine*, vol. 45, no. 1, pp. 38–57, Feb. 2003.

[242] Tom Addiscott, “Emergence or Self-organization? Look to the Soil Population”, *Communicative and Integrative Biology*, vol. 4, no. 4, pp. 469–470, Jul. 2011, DOI: 10.4161/cib.4.4.15547.

[243] Tom De Wolf / Tom Holvoet, “Emergence Versus Self-Organisation: Different Concepts But Promising When Combined”, *Engineering Self-Organising Systems: Methodologies and Applications* (edited by S. Brückner *et al.*), Series: Lecture Notes in Computer Science, vol. 3464, pp. 1–15, Springer, Berlin Heidelberg, 2005.

[244] Jürgen Appelo, “Self-Organization vs. Emergence”, *online*, Oct. 2009, URL: `http://noop.nl/2009/10/self-organization-vs-emergence.html`.

[245] Andy Brandt, “The Triangle of Self-Organization”, *online*, Jul. 2013, URL: `http://pragmaticleader.net/blog/2013/7/3/the-triangle-of-self-organization`.

[246] Deborah M. Gordon, “From Division of Labor to the Collective Behavior of Social Insects”, *Behavioral Ecology and Sociobiology*, pp. 1–8, Dec. 2015, DOI: 10.1007/s00265-015-2045-3.

[247] Steven Berlin Johnson, “Emergence: The Connected Lives of Ants, Brains, Cities, and Software”, Scribner, New York, NY, USA, 2001, ISBN: 978-0-684-86876-9.

[248] Humberto R. Maturana / Francisco J. Varela, “Autopoiesis and Cognition: The Realization of the Living”, Series: Boston Studies in the Philsosophy and History of Science, vol. 42, D. Reidel Publishing, Dordrecht, Holland, 1980, ISBN: 978-9-027-71016-1.

[249] Niklas Luhmann, “Die Gesellschaft der Gesellschaft”, Suhrkamp, Frankfurt am Main, 1997, ISBN: 3-518-58240-2.

[250] Niklas Luhmann, “Operational Closure and Structural Coupling: The Differentiation of the Legal System”, *Cardoso Law Review*, vol. 13, pp. 1419–1441, 1992.

[251] “Luhmann Observed: Radical Theoretical Encounters” (edited by Anders la Cour / Andreas Philippopoulos-Mihalopoulos), Palgrave Macmillan, Jun. 2013, ISBN: 978-1-137-01528-0.

[252] Christopher G. Langton, "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation", *Physica D: Nonlinear Phenomena*, vol. 42, no. 1–3, pp. 12–37, Jun. 1990, DOI: 10.1016/0167-2789(90)90064-V.

[253] James P. Crutchfield / Karl Young, "Computation at the Onset of Chaos", *Entropy, Complexity, and the Physics of Information* (edited by W. Zurek), Series: SFI Studies in the Sciences of Complexity, vol. 8, pp. 223–269, Addison-Wesley, Reading, Massachusetts, 1990.

[254] Jochen Fromm, "Edge of Chaos", *online*, May 2009, URL: `http://www.wiki.cas-group.net/index.php?title=File:Edge_of_Chaos.png`.

[255] Roger Lewin, "Complexity: Life at the Edge of Chaos", Second Edition, University of Chicago Press, London, UK, 1999, ISBN: 0-226-47654-5.

[256] Katrina Schwartz, "On the Edge of Chaos: Where Creativity Flourishes", *online*, May 2014, URL: `http://ww2.kqed.org/mindshift/2014/05/06/on-the-edge-of-chaos-where-creativity-flourishes/`.

[257] Robert M. Bilder / Kendra S. Knudsen, "Creative Cognition and Systems Biology on the Edge of Chaos", *Frontiers in Psychology*, vol. 5, no. 1104, Sep. 2014, DOI: 10.3389/fpsyg.2014.01104.

[258] Gary William Flake, "The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation", MIT Press, Cambridge, Massachusetts, 1998, ISBN: 978-0-262-56127-3.

[259] Elisabeth Göbel, "Theorie und Gestaltung der Selbstorganisation", Series: Betriebswirtschaftliche Forschunsergebnisse, vol. 111, Duncker & Humblot, Berlin, 1998, ISBN: 978-3-428-49434-7.

[260] Heinz von Foerster, "Principles of Self-Organization – In a Socio-Managerial Context", *Self-Organization and Management of Social Systems: Insights, Promises, Doubts, and Questions* (edited by Hans Ulrich / Gilbert J. B. Probst), Springer Series in Synergetics, vol. 26, pp. 2–24, Springer-Verlag, Berlin Heidelberg, 1984, ISBN: 978-3-642-69764-7.

[261] F. G. Varela / H. R. Maturana / R. Uribe, "Autopoiesis: The Organization of Living Systems, Its Characterization and a Model", *Biosystems*, vol. 5, no. 4, pp. 187–196, May 1974, DOI: 10.1016/0303-2647(74)90031-8.

[262] Scott Camazine / Jean-Louis Deneubourg / Nigel R. Franks / James Sneyd / Guy Theraulaz / Eric Bonabeau, "Self-Organization in Biological Systems", Series: Princeton Studies in Complexity, Princeton University Press, 2003, ISBN: 978-0-691-11624-2.

[263] Francis Heylighen, "The Science of Self-Organization and Adaptivity", *Knowledge Management, Organizational Intelligence and Learning, and Complexity* (edited by L. Douglas Kiel), Series: The Encyclopedia of Life Support Systems, pp. 1–26, EOLSS Publishers Company Limited, United Kingdom, Jan. 2009, ISBN: 978-1-848-26913-2.

[264] Pierre Paul Grassé, "La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs", *Insectes Sociaux*, vol. 6, no. 1, pp. 41–83, Mar. 1959.

[265] Leslie Marsh / Christian Onof, "Stigmergic Epistemology, Stigmergic Cognition", *Cognitive Systems Research*, vol. 9, no. 1–2, pp. 136–149, Mar. 2008, DOI: 10.1016/j.cogsys.2007.06.009.

[266] Marco Dorigo / Mauro Birattari / Thomas Stützle, "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique", *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[267] Carlos Gershenson, “Self-Organizing Traffic Lights”, *Complex Systems*, vol. 16, no. 1, pp. 29–53, 2005.

[268] H. van Dyke Parunak, “Making Swarming Happen”, *Proc. of Conference on Swarming and Network Enabled Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance*, pp. 26–40, Jan. 2003.

[269] Ralph Beckers / Owen E. Holland / Jean-Louis Deneubourg, “From Local Actions to Global Tasks: Stigmergy and Collective Robotics”, *Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic* (edited by Holk Cruse / Jeffrey Dean / Helge Ritter), Series: Studies in Cognitive Systems, vol. 26, pp. 1008–1022, Springer Netherlands, 2000, DOI: 10.1007/978-94-010-0870-9_63.

[270] Peter A. Corning, “Synergy and Self-Organization in the Evolution of Complex Systems”, *Systems Research*, vol. 12, no. 2, pp. 89–121, 1995, DOI: 10.1002/sres.3850120204.

[271] Hermann Haken, “Synergetics: Introduction and Advanced Topics”, Springer-Verlag, Berlin Heidelberg, 2004, ISBN: 978-3-540-40824-6.

[272] “International Encyclopedia of Ergonomics and Human Factors” (edited by Waldemar Karwowski), Second Edition, vol. 3, CRC Press, Taylor & Francis Group, Boca Raton, FL, 2006, ISBN: 978-0-415-30430-6.

[273] Peter A. Corning, “What is Life? Among Other Things, It’s a Synergistic Effect!”, *Cosmos and History: The Journal of Natural and Social Philosophy*, vol. 4, no. 1–2, pp. 233–243, 2008, ISSN: 1832-9101.

[274] Angus Stevenson, “Oxford Dictionary of English”, Third Edition, Oxford University Press, Aug. 2010, ISBN: 978-0199571123.

[275] Robley Dunglison, “Medical Lexicon: A Dictionary of Medical Science”, Ninth Edition, Blanchard and Lea, Philadelphia, 1853.

[276] Carlos Gershenson, “A General Methodology for Designing Self-Organizing Systems”, *ECCO Working Paper*, May 2005, URL: `http://arxiv.org/abs/nlin/0505009v3`.

[277] Peter A. Corning, “Nature’s Magic: Synergy in Evolution and the Fate of Humankind”, Cambridge University Press, May 2003, ISBN: 978-0-521-82547-4.

[278] “Innovations and Advanced Techniques in Computer and Information Sciences and Engineering” (edited by Tarek Sobh), Springer, Dordrecht, 2007, ISBN: 978-1-4020-6268-1.

[279] W. D. Hamilton, “Geometry for the Selfish Herd”, *Journal of Theoretical Biology*, vol. 31, no. 2, pp. 295–311, May 1971, DOI: 10.1016/0022-5193(71)90189-5.

[280] Friedrich Böhringer (own work), “Schafherde in Schoren”, *online*, Oct. 2009, licensed under CC BY-SA 2.5, URL: `https://commons.wikimedia.org/w/index.php?curid=8694470`.

[281] Yoav Shoham / Kevin Leyton-Brown, “Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations”, Cambridge University Press, 2009, ISBN: 978-0-521-89943-7.

[282] Adam Smith, “The Wealth of Nations”, Series: Oxford World’s Classics, Oxford University Press, May 2008, ISBN: 978-0-19-953592-7.

[283] William Ross Ashby, “An Introduction to Cybernetics”, Fourth Impression, Chapman & Hall Ltd, London, 1961, ISBN: 978-0-416-68300-4.

[284] Wordpress, “The First Law of Cybernetics”, *online*, Oct. 2011, URL: `https://firstlaw.wordpress.com/2011/10/18/ashbys-law/`.

[285] William Ross Ashby, “Principles of the Self-Organizing System”, *Principles of Self-Organization: Transactions of the University of Illinois Symposium* (edited by H. von Foerster / G. W. Zopf Jr.), pp. 255–278, Pergamon Press, London, UK, 1962.

[286] William Ross Ashby, “Requisite Variety and Its Implications for the Control of Complex Systems”, *Cybernetica*, vol. 1, no. 2, pp. 83–99, 1958.

[287] Stafford Beer, “The Heart of Enterprise”, Series: Managerial Cybernetics of Organization, vol. 2, Wiley, 1979, ISBN: 978-0-471-27599-2.

[288] “Artificial Life: An Overview” (edited by Christopher G. Langton), Series: A Bradford Book – Complex Adaptive Systems, MIT Press, Cambridge, Massachusetts, 1997, ISBN: 978-0-262-62112-0.

[289] John von Neumann, “The General and Logical Theory of Automata”, *Cerebral Mechanisms in Behavior – The Hixon Symposium* (edited by L. A. Jeffress), pp. 1–31, John Wiley & Sons, New York, NY, USA, 1951, presented September 20, 1948, in Pasadena.

[290] John von Neumann / Arthur W. Burks, “Theory of Self-Reproducing Automata”, University of Illinois Press, Urbana and London, 1966.

[291] Martin Gardner, “Mathematical Games – The Fantastic Combinations of John Conway’s New Solitaire Game ‘Life’”, *Scientific American*, vol. 223, pp. 120–123, Oct. 1970, ISBN: 0-89454-001-7.

[292] Elwyn R. Berlekamp / John H. Conway / Richard K. Guy, “Winning Ways for Your Mathematical Plays”, Second Edition, vol. 4, Taylor & Francis, 2004, ISBN: 978-1-568-81144-4.

[293] Christopher G. Langton, “Studying Artificial Life with Cellular Automata”, *Physica D: Nonlinear Phenomena*, vol. 22, no. 1–3, pp. 120–149, Oct. 1986, DOI: 10.1016/0167-2789(86)90237-X.

[294] A. Gajardo / A. Moreira / E. Goles, “Complexity of Langton’s Ant”, *Discrete Applied Mathematics*, vol. 117, no. 1–3, pp. 41–50, Mar. 2002, DOI: 10.1016/S0166-218X(00)00334-6.

[295] A. K. Dewdney, “Computer Recreations – The Cellular Automata Programs That Create Wireworld, Rugworld and Other Diversions”, *Scientific American*, vol. 262, pp. 146–149, 1990.

[296] “Cellular Automata: 10th International Conference on Cellular Automata for Research and Industry” (edited by Georgios C. Sirakoulis / Stefania Bandini), Series: Lecture Notes in Computer Science, vol. 7495, Springer, Berlin, Heidelberg, 2012, ISBN: 978-3-642-33349-1.

[297] Konrad Zuse, “Rechnender Raum”, Series: Schriften zur Datenverarbeitung, Springer Fachmedien, Wiesbaden, 1969, ISBN: 978-3-663-00810-1, DOI: 10.1007/978-3-663-02723-2.

[298] Stephen Wolfram, “A New Kind of Science”, Wolfram Media, Jun. 2002, ISBN: 978-1-57955-008-0.

[299] Steve Tadelis, “Game Theory: An Introduction”, Princeton University Press, Princeton, New Jersey, Jan. 2013, ISBN: 978-0-691-12908-2.

[300] John Nash, “Non-Cooperative Games”, *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, Sep. 1951, DOI: 10.2307/1969529.

[301] Roman L. Weil Jr., “The N-Person Prisoner’s Dilemma: Some Theory and a Computer-Oriented Approach”, *Systems Research and Behavioral Science*, vol. 11, no. 3, pp. 227–234, May 1966, DOI: 10.1002/bs.3830110310.

[302] Robert W. Rosenthal, "Games of Perfect Information, Predatory Pricing and the Chain-Store Paradox", *Journal of Economic Theory*, vol. 25, no. 1, pp. 92–100, Aug. 1981, DOI: 10.1016/0022-0531(81)90018-1.

[303] John von Neumann / Oskar Morgenstern, "Theory of Games and Economic Behavior", Sixtieth Anniversary Edition, Princeton University Press, Princeton, New Jersey, 2007, ISBN: 978-0-691-13061-3.

[304] Michael Wooldridge, "An Introduction to Multiagent Systems", Second Edition, John Wiley & Sons, Jun. 2009, ISBN: 978-0-470-51946-2.

[305] Stan Franklin / Art Graesser, "Is it an Agent, or Just a Program? A Taxonomy for Autonomous Agents", *Proc. of Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages*, pp. 21–35, Springer, 1996, ISBN: 3-540-62507-0.

[306] Stuart Russell / Peter Norvig, "Artificial Intelligence: A Modern Approach", Third Edition, Pearson Education Limited, Aug. 2013, ISBN: 978-1-292-02420-2.

[307] Jochen Fromm, "The Emergence of Complexity", Kassel University Press, Kassel, 2004, ISBN: 978-3-89958-069-3.

[308] Jacques Ferber, "Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence", Addison-Wesley Longman Publishing, Boston, MA, USA, Feb. 1999, ISBN: 978-0-201-36048-6.

[309] Riccardo Poli, "Analysis of the Publications on the Applications of Particle Swarm Optimisation", *Journal of Artificial Evolution and Application*, vol. 2008, no. 4, pp. 1–10, Jan. 2008, DOI: 10.1155/2008/685175.

[310] Shiyong Wang / Jiafu Wan / Daqiang Zhang / Di Li / Chunhua Zhang, "Towards Smart Factory for Industry 4.0: A Self-Organized Multi-Agent System with Big Data Based Feedback and Coordination", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 101, no. C, pp. 158–168, Jun. 2016, DOI: 10.1016/j.comnet.2015.12.017.

[311] Onn Shehory / Katia Sycara / Gita Sukthankar / Vick Mukherjee, "Agent Aided Aircraft Maintenance", *Proc. of 3rd Annual Conference on Autonomous Agents*, pp. 306–312, 1999, DOI: 10.1145/301136.301216.

[312] Vivek Kumar / S. Srinivasan, "A Review of Supply Chain Management Using Multi-Agent System", *International Journal of Computer Science Issues*, vol. 7, no. 5, pp. 198–205, Sep. 2010, ISSN: 1694-0814.

[313] T. Logenthiran / Dipti Srinivasan / Ashwin M. Khambadkone, "Multi-Agent System for Energy Resource Scheduling of Integrated Microgrids in a Distributed System", *Electric Power Systems Research*, vol. 81, no. 1, pp. 138–148, Jan. 2011, DOI: 10.1016/j.epsr.2010.07.019.

[314] Muaz A. Niazi / Amir Hussain, "Agent-Based Computing from Multi-Agent Systems to Agent-Based Models: A Visual Survey", *Scientometrics*, vol. 89, no. 2, pp. 479–499, Nov. 2011, DOI: 10.1007/s11192-011-0468-9.

[315] Nigel Gilbert, "Agent-Based Models", Series: Quantitative Applications in the Social Sciences, vol. 153, Sage Publications, 2008, ISBN: 978-1-4129-4964-4.

[316] Craig W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model", *Proc. of 14th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 25–34, Jul. 1987, DOI: 10.1145/37401.37406.

[317] Craig W. Reynolds, "Boids – Background and Update", *online*, Jul. 2007, URL: `http://www.red3d.com/cwr/boids/`.

[318] András Czirók / Mária Vicsek / Tamás Vicsek, “Collective Motion of Organisms in Three Dimensions”, *Physica A: Statistical Mechanics and its Applications*, vol. 264, no. 1–2, pp. 299–304, Feb. 1999, DOI: 10.1016/S0378-4371(98)00468-3.

[319] M. Ballerini / N. Cabibbo / R. Candelier / A. Cavagna / E. Cisbani / I. Giardina / V. Lecomte / A. Orlandi / G. Parisi / A. Procaccini / M. Viale / V. Zdravkovic, “Interaction Ruling Animal Collective Behavior Depends on Topological Rather than Metric Distance: Evidence from a Field Study”, *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1232–1237, Jan. 2008, DOI: 10.1073/pnas.0711437105.

[320] Yuki Sakamoto / Tatsuji Takahashi, “Metric and Topological Neighborhoods in Flocking Models”, *Proc. of 8th International Conference on Bioinspired Information and Communications Technologies*, pp. 118–121, 2014, DOI: 10.4108/icst.bict.2014.258036.

[321] Christopher Hartman / Bedřich Beneš, “Autonomous Boids”, *Computer Animation and Virtual Worlds*, vol. 17, no. 3–4, pp. 199–206, Jul. 2006, DOI: 10.1002/cav.123.

[322] Carlos Delgado-Mata / Jesus Ibanez Martinez / Simon Bee / Rocio Ruiz-Rodarte / Ruth Aylett, “On the Use of Virtual Animals with Artificial Fear in Virtual Environments”, *New Generation Computing*, vol. 25, no. 2, pp. 145–169, Feb. 2007, DOI: 10.1007/s00354-007-0009-5.

[323] Xiaoyuan Tu / Demetri Terzopoulos, “Artificial Fishes: Physics, Locomotion, Perception, Behavior”, *Proc. of 21st Annual Conference on Computer Graphics and Interactive Techniques*, pp. 43–50, Jul. 1994, DOI: 10.1145/192161.192170.

[324] Iain D. Couzin / Jens Krause / Richard James / Graeme D. Ruxton / Nigel R. Franks, “Collective Memory and Spatial Sorting in Animal Groups”, *Journal of Theoretical Biology*, vol. 218, pp. 1–11, 2002, DOI: 10.1006/yjtbi.3065.

[325] Kai Nagel / Michael Schreckenberg, “A Cellular Automaton Model for Freeway Traffic”, *Journal de Physique I*, vol. 2, no. 12, pp. 2221–2229, Dec. 1992, DOI: 10.1051/jp1:1992277.

[326] Tamás Vicsek / András Czirók / Eshel Ben-Jacob / Inon Cohen / Ofer Shochet, “Novel Type of Phase Transition in a System of Self-Driven Particles”, *Physical Review Letters*, vol. 75, no. 6, pp. 1226–1229, Aug. 1995, DOI: 10.1103/PhysRevLett.75.1226.

[327] Yue-Xian Li / Ryan Lukeman / Leah Edelstein-Keshet, “Minimal Mechanisms for School Formation in Self-Propelled Particles”, *Physica D: Nonlinear Phenomena*, vol. 237, no. 5, pp. 699–720, May 2008, DOI: 10.1016/j.physd.2007.10.009.

[328] Cynthia Nikolai / Gregory Madey, “Tools of the Trade: A Survey of Various Agent Based Modeling Platforms”, *Journal of Artificial Societies and Social Simulation*, vol. 12, no. 2, pp. 1–37, Mar. 2009, ISSN: 1460-7425.

[329] Fabio Luigi Bellifemine / Giovanni Caire / Dominic Greenwood, “Developing Multi-Agent Systems with JADE”, Wiley Series in Agent Technology, vol. 7, John Wiley & Sons, West Sussex, England, Mar. 2007, ISBN: 978-0-470-05840-4.

[330] Daniel Marolt / Jürgen Scheible / Göran Jerke / Vinko Marolt, “SWARM: A Multi-Agent System for Layout Automation in Analog Integrated Circuit Design”, *Agent and Multi-Agent Systems: Technology and Applications (10th KES International Conference, KES-AMSTA 2016)* (edited by Gordan Jezic / Yun-Heh Jessica Chen-Burger / Robert J. Howlett / Lakhmi C. Jain), Series: Smart Innovation, Systems and Technologies, vol. 58, ch. 2, pp. 15–31, Springer, Jun. 2016, DOI: 10.1007/978-3-319-39883-9_2.

[331] Thomas Burdick / Peter Herth / Göran Jerke / Christel Bürzele / Daniel Marolt / Vinko Marolt, “FR PCells”, *Patent Application*, Dec. 2015, title obscured for reasons of confidentiality.

[332] Daniel Marolt / Jürgen Scheible / Göran Jerke, “The Application of Layout Module Generators upon Circuit Structure Recognition”, *Proc. of CDNLive! EMEA*, 6 pages, May 2011, Session AC13.

[333] Kevin Kelly, “The Bottom Is Not Enough”, *online*, Feb. 2008, URL: `http://kk.org/thetechnium/the-bottom-is-n/`.

[334] G. H. Meisters, “Polygons Have Ears”, *The American Mathematical Monthly*, vol. 82, no. 6, pp. 648–651, Jun./Jul. 1975, DOI: 10.2307/2319703.

[335] Brian W. Kernighan / Shen Lin, “An Efficient Heuristic Procedure for Partitioning Graphs”, *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, Feb. 1970, DOI: 10.1002/j.1538-7305.1970.tb01770.x.

[336] C. M. Fiduccia / R. M. Mattheyses, “A Linear-Time Heuristic for Improving Network Partitions”, *Proc. of 19th Design Automation Conference*, pp. 175–181, Jun. 1982, DOI: 10.1109/DAC.1982.1585498.

[337] A. H. Farrahi / D. J. Hathaway / M. Wang / M. Sarrafzadeh, “Quality of EDA CAD Tools: Definitions, Metrics and Directions”, *Proc. of 1st International Symposium on Quality Electronic Design*, pp. 395–405, Mar. 2000, DOI: 10.1109/ISQED.2000.838903.

[338] Mohammad Tehranipoor, “CAD Algorithms – Placement”, *online*, Nov. 2008, URL: `www.engr.uconn.edu/~tehrani/teaching/cad/15_placement.pdf`.

[339] Ralph Otten, “Complexity and Diversity in IC Layout Design”, *Proc. of IEEE International Symposium on Circuits and Computers*, pp. 764–767, Oct. 1980, URL: `https://slideplayer.com/slide/13156322/`.

[340] D. W. Jepsen / C. D. Gelatt Jr., “Macro Placement by Monte Carlo Annealing”, *Proc. of IEEE International Conference on Computer Design*, pp. 495–498, Nov. 1983.

[341] Florin Balasa, “Device-Level Topological Placement with Symmetry Constraints”, *Analog Layout Synthesis – A Survey of Topological Approaches* (edited by Helmut E. Gräb), ch. 1, pp. 3–60, Springer, New York Dordrecht Heidelberg London, 2011, ISBN: 978-1-4419-6931-6, DOI: 10.1007/978-1-4419-6932-3.

[342] Maqsood Yaqub / Ronald Boellaard / Marc A. Kropholler / Adriaan A. Lammertsma, “Optimization Algorithms and Weighting Factors for Analysis of Dynamic PET Studies”, *Physics in Medicine and Biology*, vol. 51, no. 17, pp. 4217–4232, Aug. 2006, DOI: 10.1088/0031-9155/51/17/007.

[343] Carl Sechen / Alberto Luigi Sangiovanni-Vincentelli, “TimberWolf3.2: A New Standard Cell Placement and Global Routing Package”, *Proc. of 23rd Design Automation Conference*, pp. 432–439, Jun./Jul. 1986, DOI: 10.1109/DAC.1986.1586125.

[344] Robert Hooke, “Lectures De Potentia Restitutiva, or of Spring Explaining the Power of Springing Bodies to which are added some Collections”, John Martyn, Royal Society London, 1678, URL: `http://name.umdl.umich.edu/A44322.0001.001`.

[345] Jürgen Scheible, “Ein Softwarepaket zur vollautomatischen Plazierung von Bauteilen auf Leiterplatten für den industriellen Einsatz”, Ph.D. Thesis, Universität Karlsruhe, 1992, ISBN: 3-18-145920-8.

[346] Elena Lodi / Fabrizio Luccio / Cristina Mugnai / Linda Pagli, “On Two-Dimensional Data Organization I”, *Annales Societatis Mathematicae Polonae*, Series IV: Fundamenta Informaticae II, pp. 211–226, 1979.

[347] C. V. Deutsch / X. H. Wen, “An Improved Perturbation Mechanism for Simulated Annealing Simulation”, *Mathematical Geology*, vol. 30, no. 7, pp. 801–816, Oct. 1998, DOI: 10.1023/A:1021722508504.

[348] Carl Sechen, “Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing”, *Proc. of 25th Design Automation Conference*, pp. 73–80, Jun. 1988, DOI: 10.1109/DAC.1988.14737.

[349] D. F. Wong / H. W. Leong / C. L. Liu, “Simulated Annealing for VLSI Design”, Kluwer Academic Publishers, Boston, Lancaster, Dordrecht, 1988, ISBN: 978-1-4612-8947-0.

[350] Thomas H. Cormen / Charles E. Leiserson / Ronald L. Rivest / Clifford Stein, “Introduction To Algorithms”, Second Edition, MIT Press, Cambridge, Massachusetts, 2001, ISBN: 0-262-03293-7.

[351] Jørgen Bang-Jensen / Gregory Gutin / Anders Yeo, “When the Greedy Algorithm Fails”, *Discrete Optimization*, vol. 1, no. 2, pp. 121–127, Nov. 2004, DOI: 10.1016/j.disopt.2004.03.007.

[352] J. D. Conway / G. G. Schrooten, “An Automatic Layout Generator for Analog Circuits”, *Proc. of European Conference on Design Automation*, pp. 513–519, Mar. 1992, DOI: 10.1109/EDAC.1992.205989.

[353] Nuttorn Jangkrajarng / Sambuddha Bhattacharya / Roy Hartono / C.-J. Richard Shi, “IPRAIL – Intellectual Property Reuse-Based Analog IC Layout Automation”, *Integration*, vol. 36, no. 4, pp. 237–262, Nov. 2003, DOI: 10.1016/j.vlsi.2003.08.004.

[354] John K. Ousterhout, “Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 1, pp. 87–100, Jan. 1984, DOI: 10.1109/TCAD.1984.1270061.

[355] Daniel Marolt / Jürgen Scheible / Göran Jerke / Vinko Marolt, “SWARM: A Self-Organization Approach for Layout Automation in Analog IC Design”, *International Journal of Electronics and Electrical Engineering*, vol. 4, no. 5, pp. 374–385, Oct. 2016, DOI: 10.18178/ijeee.4.5.374-385.

[356] William Beebe, “Edge of the Jungle”, The Star Series, Henry Holt and Company, New York, 1921, URL: `http://ia600203.us.archive.org/25/items/edgejungle00beebgoog/edgejungle00beebgoog.pdf`.

[357] Theodore Christian Schneirla, “A Unique Case of Circular Milling in Ants, Considered in Relation to Trail Following and the General Problem of Orientation”, *American Museum Novitates*, no. 1253, pp. 1–26, Apr. 1944, URL: `http://digitallibrary.amnh.org/handle/2246/3733`.

[358] Daniel Marolt / Jürgen Scheible / Göran Jerke / Vinko Marolt, “A Self-Organization Approach for Layout Floorplanning Problems in Analog IC Design”, *Proc. of 12th Conference on Ph.D. Research in Microelectronics and Electronics*, pp. 1–4, Jun. 2016, DOI: 10.1109/PRIME.2016.7519454.

[359] Daniel Marolt / Jürgen Scheible / Göran Jerke / Vinko Marolt, “CAPABLE: A Layout Automation Framework for Analog IC Design”, *Proc. of MPC-Workshop*, vol. 54, pp. 49–59, Jul. 2015, ISSN: 1868-9221.

[360] Daniel Marolt / Jürgen Scheible / Göran Jerke / Vinko Marolt, “Analog Layout Automation via Self-Organization: Enhancing the Novel SWARM Approach”, *Proc. of 7th IEEE Latin American Symposium on Circuits and Systems*, pp. 55–58, Feb./Mar. 2016, DOI: 10.1109/LASCAS.2016.7451008.

[361] Daniel D. Gajski / Robert H. Kuhn, “Guest Editors’ Introduction: New VLSI Tools”, *IEEE Computer*, vol. 16, no. 12, pp. 11–14, Dec. 1983, DOI: 10.1109/MC.1983.1654264.

[362] Daniel Marolt / Thomas Burdick / Göran Jerke / Peter Herth / Vinko Marolt / Jürgen Scheible, “HIPE: Hierarchical Instance Parameter Editing of Parameterized Modules in Analog IC Design”, *Proc. of edaWorkshop 2016*, pp. 18–23, May 2016, ISBN: 978-3-86460-453-9.

[363] Göran Jerke / Vinko Marolt / Christel Bürzele / Daniel Marolt / Andreas Krinke / Peter Herth / Thomas Burdick, “Hierarchical Module Design with Cadence PCell Designer”, *CDNLive! EMEA*, Apr. 2015, Session CUS02, URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/CDNLive/Secured/Proceedings/EU/2015/CUS02.pdf`.

[364] Daniel Marolt / Matthias Greif / Jürgen Scheible / Göran Jerke, “PCDS: A New Approach for the Development of Circuit Generators in Analog IC Design”, *Proc. of 22nd Austrian Workshop on Microelectronics (Austrochip)*, pp. 1–6, Oct. 2014, DOI: 10.1109/Austrochip.2014.6946310.

[365] Matthias Greif / Daniel Marolt / Jürgen Scheible, “gPCDS: An Interactive Tool for Creating Schematic Module Generators in Analog IC Design”, *Proc. of 12th Conference on Ph.D. Research in Microelectronics and Electronics*, pp. 1–4, Jun. 2016, DOI: 10.1109/PRIME.2016.7519539.

[366] Göran Jerke / Vinko Marolt / Christel Bürzele / Peter Herth / Thomas Burdick, “Schematic and Symbol PCell Development with Cadence PCell Designer”, *CDNLive! EMEA*, May 2017, Session CUS10, URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/CDNLive/Secured/Proceedings/EU/2017/CUS10.pdf`.

[367] Andreas Gerlach / Thoralf Rosahl / Frank-Thomas Eitrich / Jürgen Scheible, “A Generic Topology Selection Method for Analog Circuits with Embedded Circuit Sizing Demonstrated on the OTA Example”, *Proc. of Design, Automation and Test in Europe Conference*, pp. 898–901, Mar. 2017, DOI: 10.23919/DATE.2017.7927115.

[368] Florian Leber / Jürgen Scheible, “Eine domänenspezifische Sprache für die prozedurale Dimensionierung im analogen IC Entwurf”, *Proc. of Informatics Inside 2017* (edited by Uwe Kloos / Natividad Martínez / Gabriela Tullius), pp. 119–120, May 2017, ISBN: 978-3-00-056455-0.

Further Sources

The bald head in Figure 3.1, Figure 3.2, Figure 3.13, and Figure 7.2 is a derivative of `https://www.iconfinder.com/icons/628287/anonym_avatar_default_head_person_unknown_user_icon` (created by Anna Litviniuk and licensed as free for commercial use).

The schematic and layout images were generated with the Virtuoso Custom IC Design Environment from Cadence Design Systems, using the Generic Process Design Kit (GPDK).

This thesis was typeset with TeXworks 0.6.2 (implemented by Jonathan Kew), using the document preparation software LaTeX (originally developed by Leslie Lamport), based on the typesetting system TeX (designed by Donald E. Knuth). The source text was written in Notepad++ 7.5.6 (devised by Don Ho). Drawings were created by the author using GIMP 2.6.8 (started by Spencer Kimball and Peter Mattis) and Inkscape 0.48.0 r9654 (initially forked from Sodipodi by Ted Gould, Bryce Harrington, Nathan Hurst, and “MenTaLguY”). The final PDF file was produced via pdfTeX-1.40.19 on December 7, 2018.

List of Figures

List of Tables

Appendix A

SWARM Outcomes for the Symmetric OTA Example

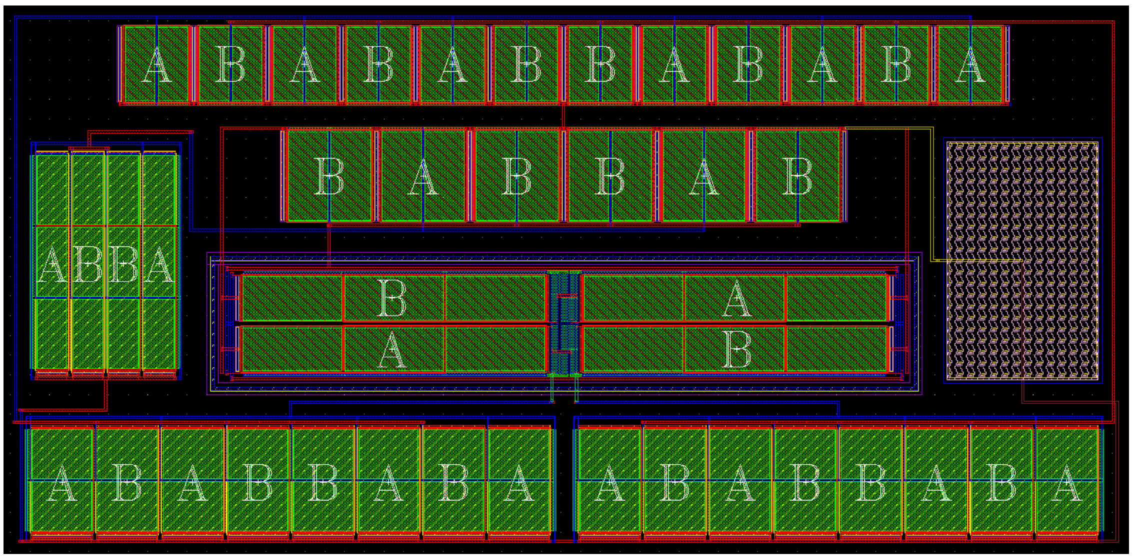


Figure A.1: Finalized layout of the Symmetric OTA, obtained by SWARM for a 2:1 aspect ratio.

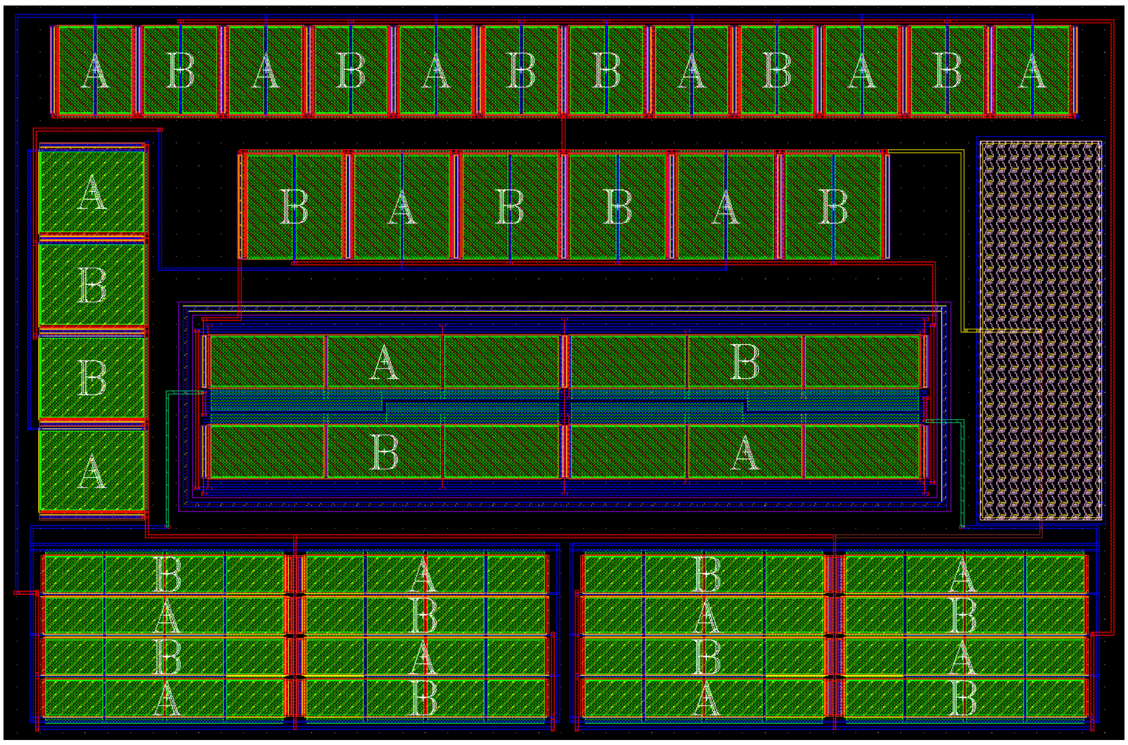


Figure A.2: Finalized layout of the Symmetric OTA, obtained by SWARM for a 3:2 aspect ratio.

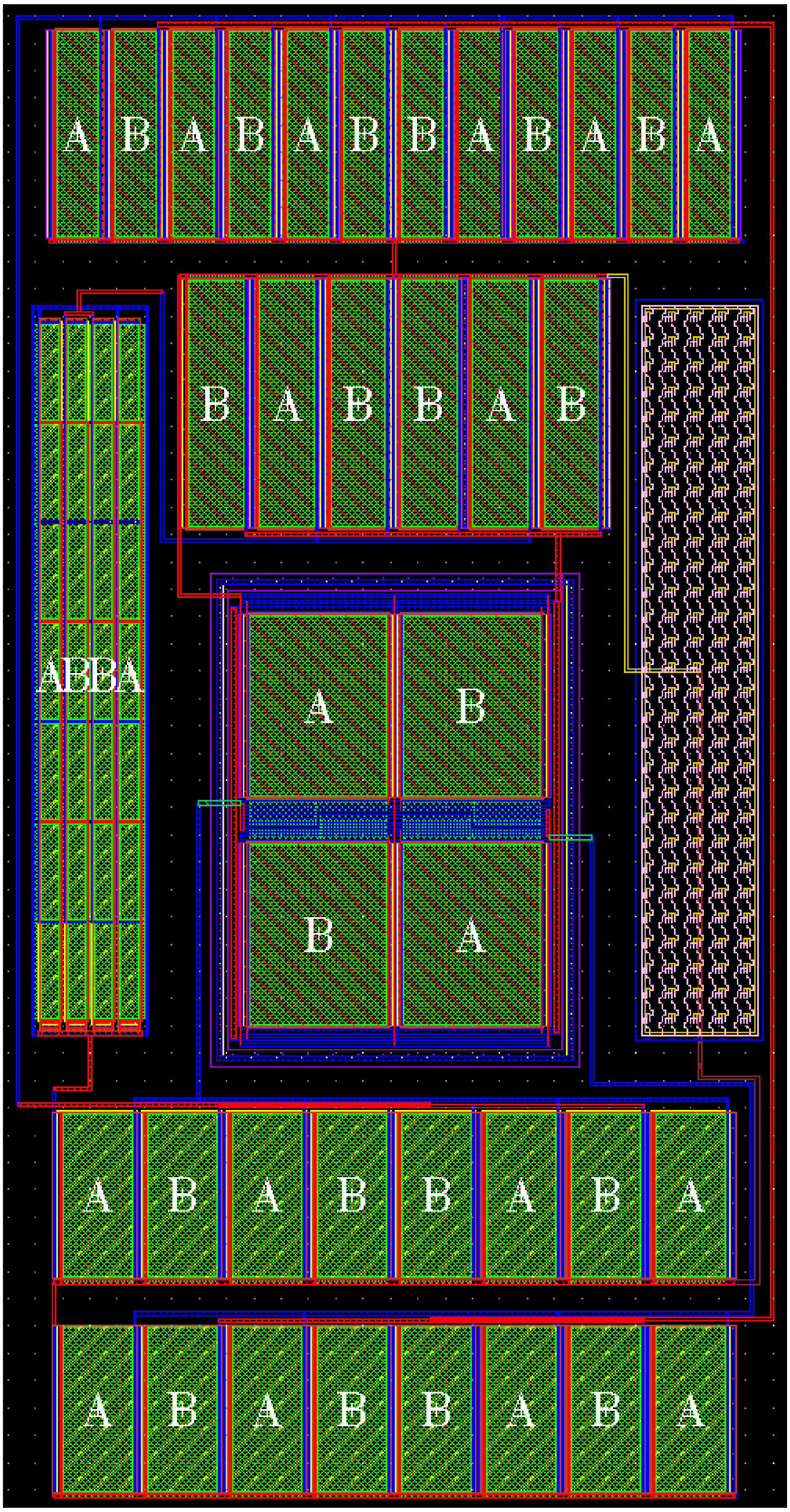


Figure A.3: Finalized layout of the Symmetric OTA, obtained by SWARM for a 1:2 aspect ratio.

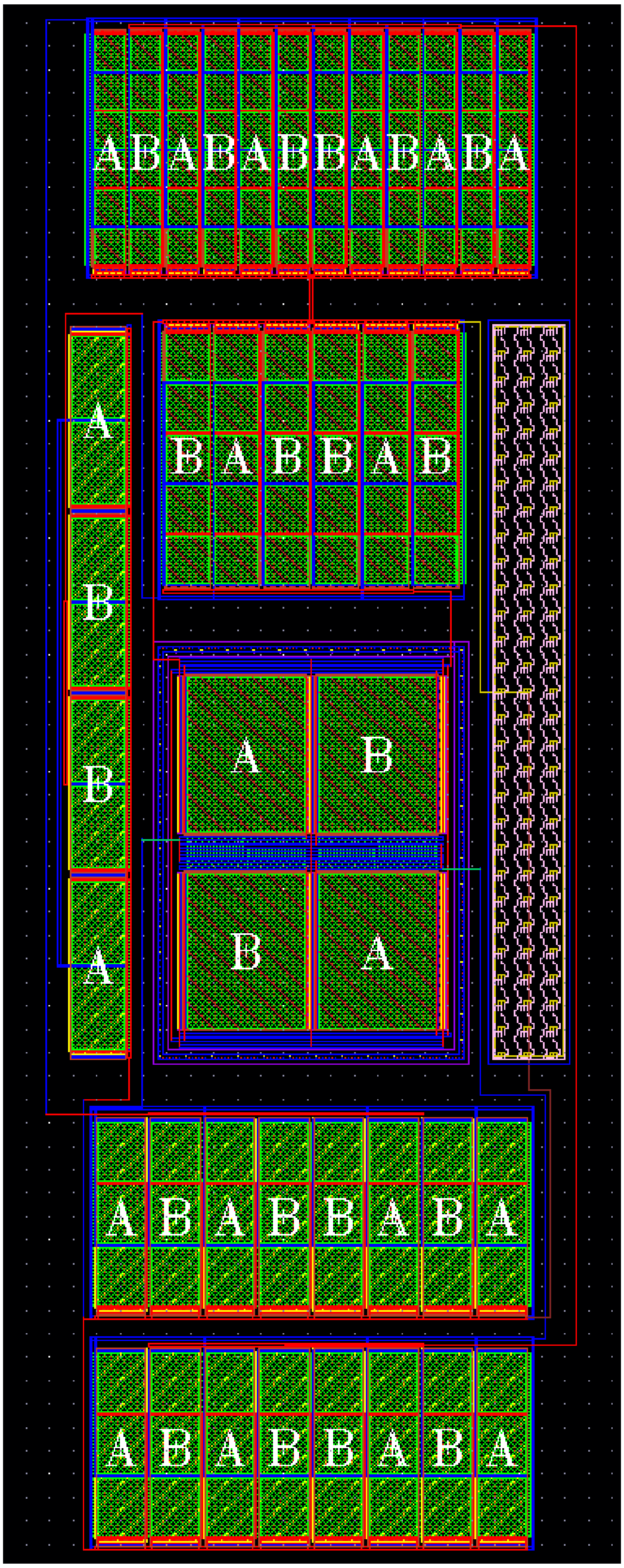


Figure A.4: Finalized layout of the Symmetric OTA, obtained by SWARM for a 1:3 aspect ratio.

Appendix B

SWARM Outcomes for the Folded Cascode OTA Example

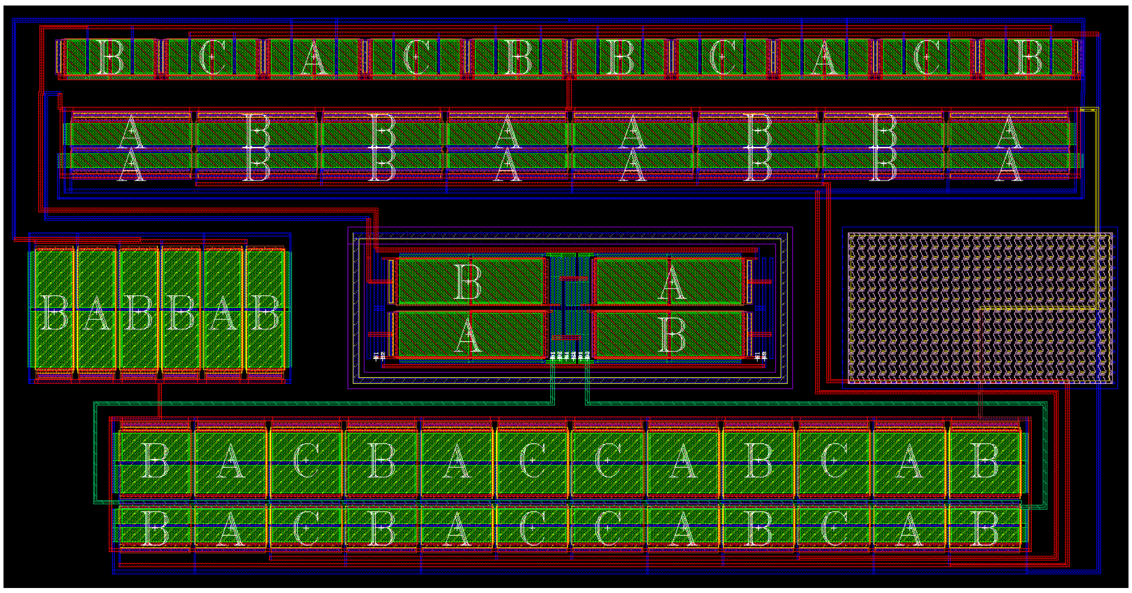


Figure B.1: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 2:1 aspect ratio.

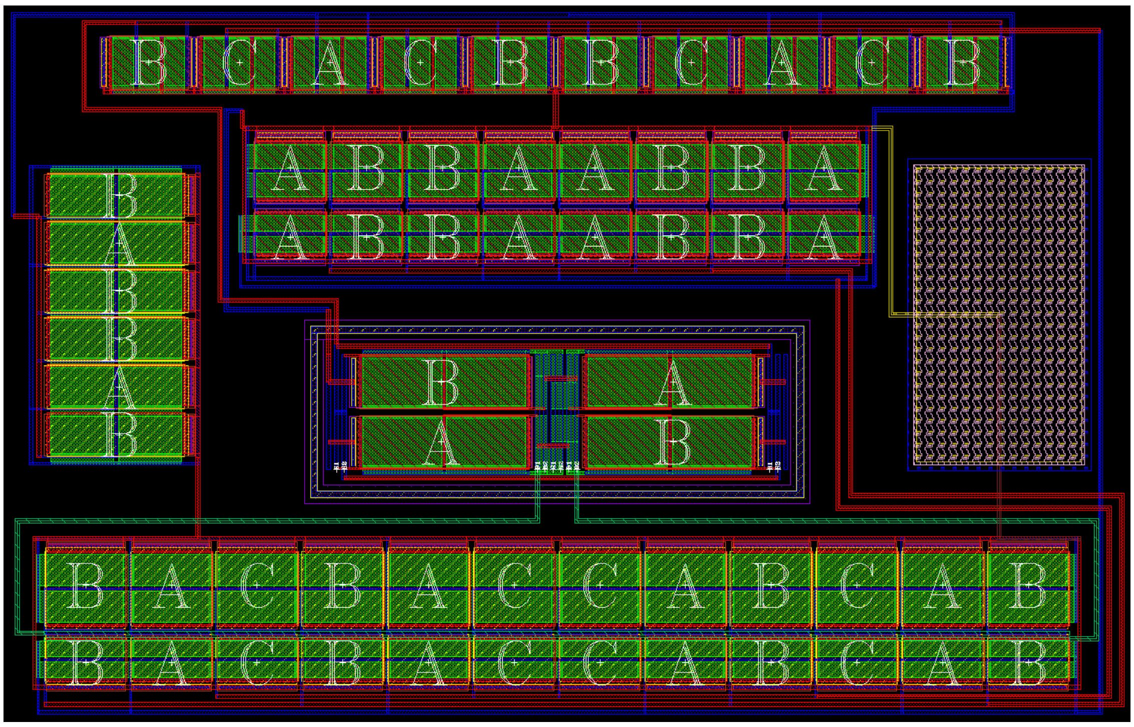


Figure B.2: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 3:2 aspect ratio.

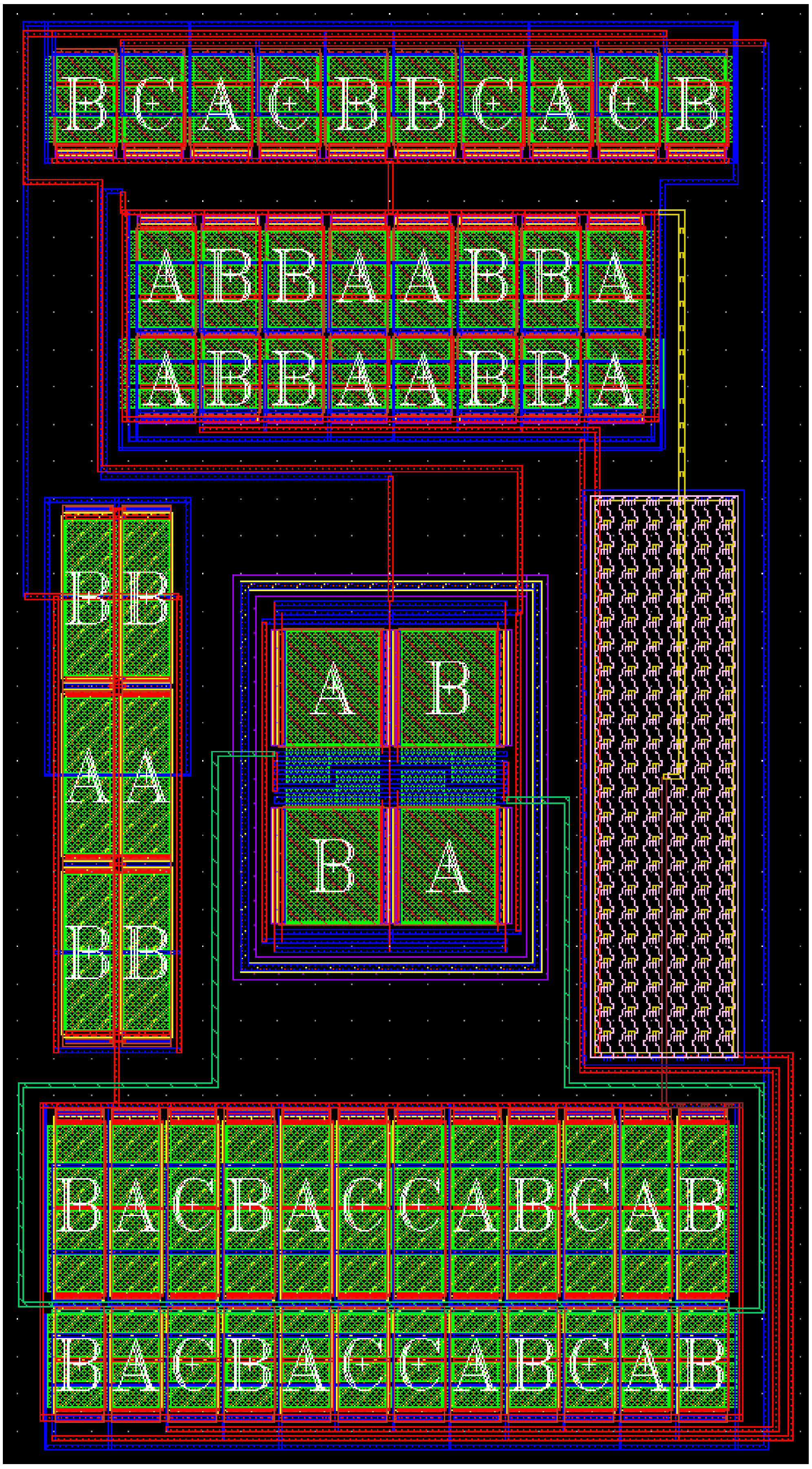


Figure B.3: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 1:2 aspect ratio.

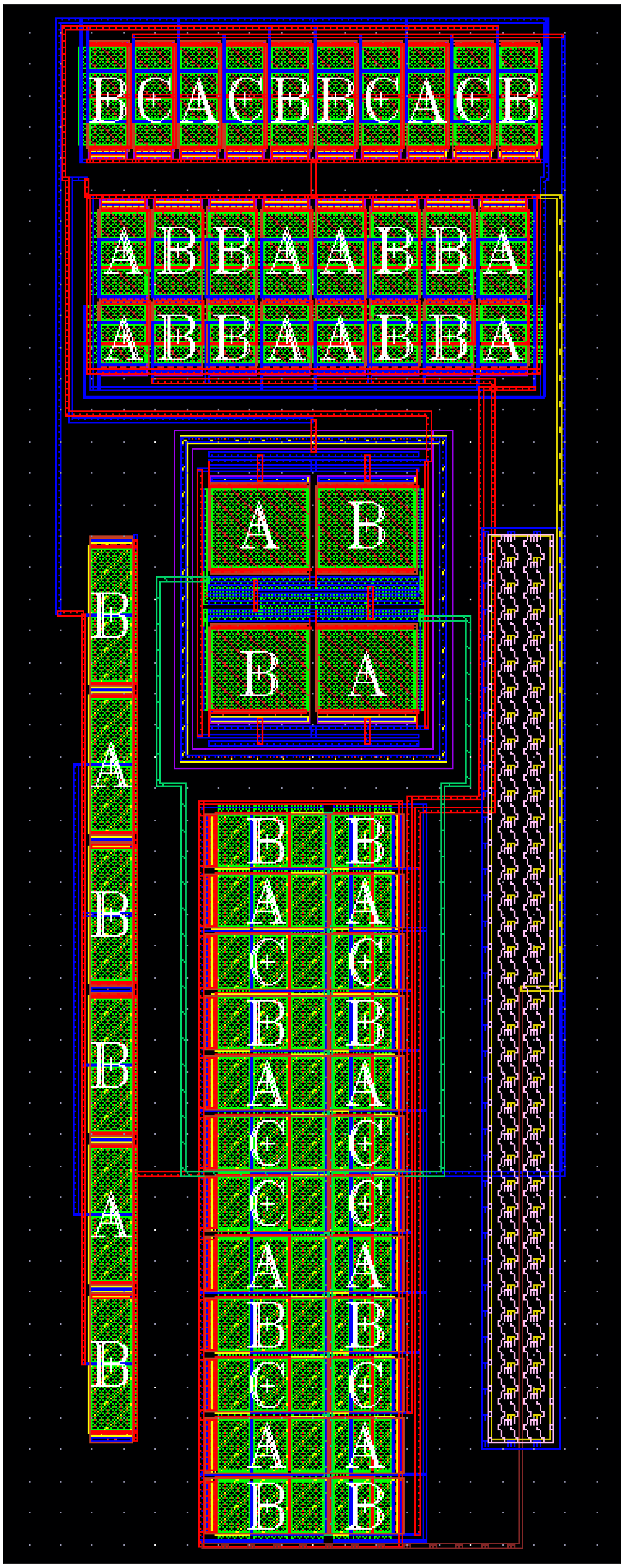


Figure B.4: Finalized layout of the Folded Cascode OTA, obtained by SWARM for a 1:3 aspect ratio.