University of Naples "Federico II"

Department of Engineering Management (DIEG)

PhD Program in Science and Technology Management

XXII Cycle

PhD Thesis

Agent-Based Methodologies for Facility Location Problems

PhD Candidate: Andrea Genovese

PhD Tutor: prof. Giuseppe Bruno

Certificate

This is to certify that the thesis titled Agent-Based Methodologies for Facility Location Problems being submitted by Andrea Genovese for the award of the Doctorate in Science and Technology Management at University of Naples “Federico II”, is a research work carried out by him under my supervision.
The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Giuseppe Bruno
Associate Professor
Department of Engineering Management (DIEG)
University of Naples “Federico II”

Table of Contents

Introduction

The daily work of professionals involves making a series of *decisions*. In fact, the world relies on systems designed by engineers and business people. Thus, the quality of decisions made by these two categories of professionals is of critical importance.

Decisions are made by looking at the relevant data and making judgments. Making decisions on issues with important consequences has become a highly complex problem due to the many competing forces under which the world is operating today. Anyone who holds a technical, managerial, or administrative job these days is faced with making decisions daily at work. Decisions may involve:

- Determining which ingredients and in what quantities to add to a mixture being made so that it will meet specifications on its composition;
- Selecting one among a small number of suppliers to order raw materials from;
- Determining the quantities of various products to manufacture in the next period;
- Allocating available funds among various competing agencies;
- Deciding which route to take to go to a given location;
- Selecting an appropriate location for an industrial facility;
- Determining how many check-in desks to open during airport operating hours.

A situation such as one of these requiring some decisions to be made is known as a Decision Making Problem. Today it is essential to make decisions on a rational basis: the most rational way for solving decision making problems is through quantitative analysis. This implies the adoption of the following steps:

- *Precise definition of the problem*;
- *Construction of a mathematical model of the problem*;
- *Solution of the model*
- *Implementation of the solution*.

The mathematical model associated with a decision making problem is charachterized by the following basic elements:

- An *Objective Function*, expressing a decision criterion to be optimized;
- A set of *Constraints*, limiting the feasible solutions to the problem.

As regards the solution of the mathematical model associated with the problem, several methods have been developed, according to the different complexity of the problem to be faced. Traditional approaches to solve these problems can be classified into two main categories: exact methods and heuristic methods.

For a long time, these optimization techniques have represented the only available approach to solve different types of decision-making problem, both at strategic and tactical levels.
In the last decade, agent-based computing has been suggested as a promising technique for problem whose domains are distributed, complex and heterogeneous (Weiss, 1999; Wooldridge, 2002), also thanks to the availability of many commercial and open source codes including graphical interfaces for the elements of the problem. Application to several classes of optimization problems have been developed, ranging from scheduling and supply chain planning to routing.
In this dissertation, a general Agent-Based framework for modeling various location problems is proposed. The high relevance of location problems in the operations research literature arises from their wide spectrum of real applications, including decision optimization in industrial management, logistics and territorial planning. Most of these optimization problems fall in the class of *NP*-hard problems, motivating the search for heuristic and approximated algorithms. Together with the description of an Agent-Based framework, we present some computational results confirming the suitability and the effectiveness of the proposed approach.
The thesis is organized as follows: in the first chapter, an overview of Agent-Based Modeling and its foundations is proposed; then, basic concepts regarding Optimization Problems and Methods are provided. In the third chapter, a review about Agent-Based Methods for Optimization Problems is proposed. Given that Agent-Based Models characteristics seem to be very suitable to Location Problems, chapter four provides an overview of the most common problems belonging to this class, while in chapter five a framework for modeling and solving Location Problems is presented, together with computational results and further directions of research.

Chapter 1
Agents and multi-agent systems

1.1 Introduction

Multi-Agent systems (MASs) consist of a set of elements (agents) characterized by attributes, which interact with each other through the definition of appropriate rules in a given environment. MASs can be useful to reproduce many systems related to economics and social sciences, where the structure can be designed through a network (Billari *et al.* (2006), Conte *et al.* (1997)). Through ABMs, it is possible implementing an environment with its features, forecasting and exploring its future scenarios, experimenting possible alternative decisions, setting different values for the decision variables and analyzing the effects of these changes (see Axelrod (1997)).
At an aggregated level, the use of ABMs can help in understanding general properties and patterns concerning the whole scenario (Billari *et al.* (2006)) that could not be deduced nor forecasted by the observation of each agent, due to the complexity of the interactions occurring among the elements of the system.
Nowadays, the *Agent* concept is finding wide application in several contexts, ranging from Artificial Intelligence research to the development of methodologies for complex decision making problems.
In this introductory chapter the *Agent* concept with its characteristics and properties is illustrated. Then, Multi-Agent systems are introduced with a description of interaction and communication protocols among agents.

1.2 Fundamental definitions

At this moment, the literature is not able to provide a universally accepted definition of the *Agent* concept. As this concept is being utilized in several and distinct disciplinary areas, different characteristics can be considered more relevant according to the specific application field.
However, a growing number of researchers agree on considering the definition of Wooldridge and Jennings (1995) that can be stated as follows:

"An Agent is a computer system operating in an environment, capable of acting autonomously in order to achieve some predetermined objectives".

The elements of this definition can be clarified as follows:

- A Computer System is a set of hardware and software components brought together in order to perform some tasks;
- The environment is not further defined, as its characteristics can depend on various factors;
- The agent is defined as *autonomous* as it acts on the basis of its own objectives. Thus, it controls its internal state and its behavior.

Figure 1.1 depicts an abstract schema of an agent and its environment. In particular, the agent acts on the basis of its objectives; the output of these actions can modify the environment. The action to be performed is selected, among a set of feasible actions, according to a set of data coming from the environment that the agent is able to interpret and register.

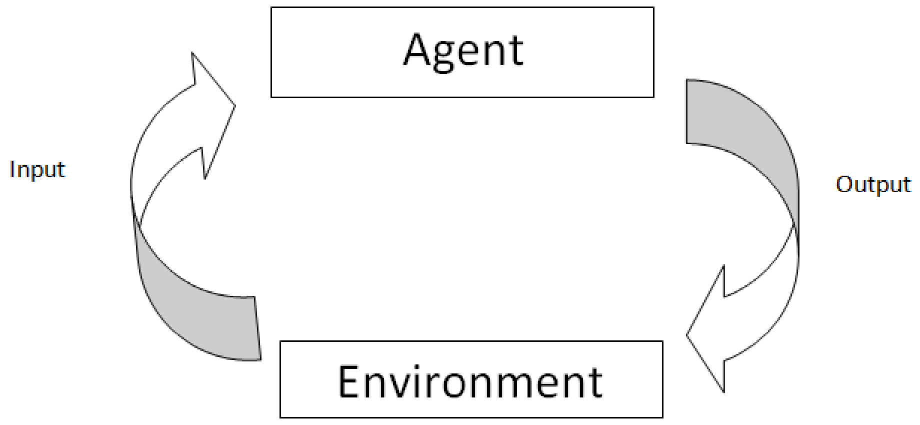


Figure 1.1 – An Agent and its Environment

Thus, the set of actions that an agent can perform depends on the characteristics of the environment in the instant when the action takes place.

However, in many situations characterized by an high complexity, the agent doesn't have a global control vision of the environment, but only a partial one; consequently, it can influence or be influenced only by a part of the environment. Hence, an agent decisional behavior can strongly depend on its "position" in the environment.

It emerges that the characteristics of the environment can influence the achievement of agents' objectives. For this reason, it is relevant to classify environment characteristics according to a synthetic schema, like the one proposed by Russel and Norvig (1997) based on the following features:

- *Accessibility*. A completely accessible environment is an environment in which an agent is capable of accessing all the information about the state of the environment itself. This characteristic is not likely to be verified in highly complex environments (like, for example, the Internet network) that report a lower accessibility level.
- *Determinism*. In a deterministic environment the action of an agent determines one and only one effect; thus, there is no uncertainty about the state that an agent can reach as a result of a given action. For agents operating in non-deterministic environments behavior predictability is difficult.
- *Episodicity*. An episodic environment is characterized by the independence of the actions of each agent on the history of the agent itself. Such an environment is endowed with a low complexity level, as an agent can decide its behavior without recalling its past states.
- *Staticity*. In a static environment, only agents can modify environment characteristics; on the contrary, in a dynamic environment, environment characteristics change according to time.
- *Continuity*. If the environment allows only a limited and fixed number of actions that can be executed, it can be defined as discrete; otherwise, it is a continuous environment.

Different degrees of these characteristics correspond to different complexity degrees. It can be stated that, from a theoretical point of view, the most complex environment is a scarcely accessible, non-deterministic, non-episodic, dynamic and continuous one.

An elementary example of an agent operating in its environment can be represented by a simple control system, like a thermostat placed in a room. The objective of the thermostat is to keep constant the temperature in the room. The agent has two possible courses of action: switching on or not a heating system. This choice is made on the basis of the temperature of the room, measured by the agent. The action selected by the agent will influence the state of the environment (the temperature of the room); at the next iteration, the agent will perceive another state of the environment and will change, if necessary, its course of action.

1.3 Agents' characteristics

Agents are often defined as *intelligent*. Also in this case, a widely accepted definition of *intelligence* cannot be retrieved in the extant literature. The most widespread one defines an *intelligent agent* through the characteristics of *autonomy* and *flexibility*.

Having already clarified the meaning of *autonomy*, it must be highlighted that *flexibility* implies the following three features:

- *Re-activeness.* Agents answer in a precise way to signals coming from the environment.
- *Pro-activeness.* Agents are endowed with goal-directed behaviors. They take the initiative in order to satisfy their design objectives.
- *Social ability.* Interactions occur among entities through a communication language in order to satisfy the design objectives.

Usually, the first two characteristics are not present at the same time in an agent. The pro-activeness implies that agents, on the basis of some inputs, choose, autonomously and depending on their specific objectives, a course of action.

On the contrary, a reactive agent simply answers to the signals coming from the environment; it does not have the ability to pursue an objective.

In the following, several classes of agents will be introduced, each of them presenting different degrees of the characteristics previously introduced.

1.4 Agents' categories

Four categories of agents, based on different degrees of the previously illustrated characteristics, will be introduced (Weiss, 1999):

- *Standard agents;*
- *Purely reactive agents;*
- *Perceptive agents*;
- *Agents with state*.

For each category of agents, properties and characteristics will be described according to the mathematic formulation introduced by Wooldridge (1999).

1.4.1 Standard Agents

The first category of agents is constituted by the standard agents that can be graphically represented as in Figure 1.1. The mathematical formulation proposed by Wooldridge (1999) is based on the assumption that the couple agent/environment can be represented by the two following sets:

- A set $S = \{s_1, \dots, s_n\}$, whose elements are the states of the environment;
- A set $A = \{a_1, \dots, a_m\}$ whose elements are the actions that an agent can perform.

In each instant the environment is in a given state.
An action can be formalized as:

$$action\colon S^* \to A$$

Where S^* is a sequence of states from S and A represents the action developed by the agent, determined by the states of the environment.
Thus, given the configuration reached by the environment, the agent chooses to perform an action assuming, as input, data coming from the environment itself. On the other hand, the state reached by the environment will be determined by the result of the action implemented by the agent at the previous iteration that depends on the environment itself. This process explains the *loop* illustrated in figure 1.1.
The behavior of the environment can be modeled as:

$$environment\colon S \times A \to \varphi(s)$$

For each state $s \in S$ and for each action $a \in A$ the environment identifies the set of the states $\varphi(s)$ that can be reached by performing the action *a* by an agent in the state *s*. If one and only one solution corresponds to each couple state-action, the environment is a deterministic one.
Thus, the interaction between the agent and the environment can be represented as a sequence of states determined by the series of actions performed by the agent.
For instance, in the sequence

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u}$$

s_0 is the initial environment state (for instance, the state in which the agent starts operating) a_u is the uth action that an agent chooses to perform, s_u is the uth state of the environment that derives from the action a_{u-1} performed by the agent in the state s_{u-1}.
The previous sequence represents the history of the evolution of an agent in its environment, if the following two conditions are respected:

$$\forall\, u \in N, a_u = action((s_0, s_1, \dots, s_u))$$

$$\forall\, u \in N : u > 0, s_u \in environment(s_{u-1}, a_{u-1})$$

The behavior of a standard agent can be tracked through the set of all the sequences that satisfy these two conditions (Wooldridge, 1999).

1.4.2 Purely reactive agents

Some agents decide action to be performed without taking in account their past history. These agents are defined *purely reactive* because they just directly answer to changes in the environment, without pursuing an objective.

Thus, a purely reactive agent can be represented as

$$action: S \rightarrow A$$

Hence, actions are just based on the current environment state. It is useful to notice that a purely reactive agent is a particular standard agent whose environment state sequence just includes the current state.

The previous example of the thermostat represents a typical example of reactive agent. A typical action can be depicted as follows:

$$action(s) = \begin{cases} switched\ off & if\ s = desired\ temperature \\ switched\ on & otherwise \end{cases}$$

1.4.3 Perceptive agents

Perceptive agents are able to *perceive* changes in the environment (Figure 1.2). This is formalized through the introduction of the *see* function that translates the ability of an agent of observing the environment in which it is placed, while the action is the decision making process associated with the selection of the course of action to be pursued. For instance, the *see* function can be a sensor that perceives the changes happening in the environment.

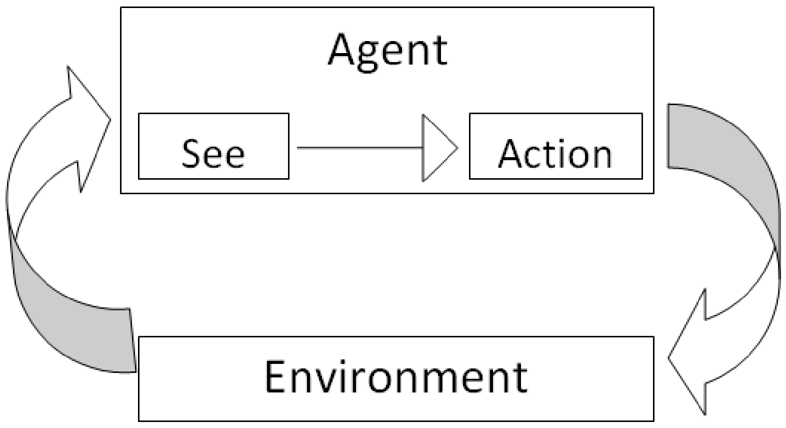


Figure 1.2 – A Perceptive Agent and its Environment

The *see* function allows transforming an environment state into an agent's perception, as listed below:

$$see:\ S \rightarrow P$$

The *action* function allows transforming the perception sequence into an action.

$$action\colon\ P^{*} \rightarrow A$$

If the agent can distinguish as many perceptions as the available states, then it is *omniscient* and it has a perfect perception of the environment. If the agent has no perception ability, thus the agent is not capable of distinguishing between different environment states.

These simple definitions allow us to explore some interesting properties of agents and perception. Suppose that we have two environment states, $s_1 \in S$ and $s_2 \in S$, such that $s_1 \neq s_2$ but $see(s_1) = see(s_2)$. Then two different environment states are mapped to the same percept, and hence the agent would receive the same perceptual information from different environment states. As far as the agent is concerned, therefore, s_1 and s_2 are indistinguishable.

1.4.4 Agents with state

As represented in figure 1.3, an agent with state memorizes the states it has visited. If I identifies the set of all the internal states of an agent, it can be introduced a new function, named *next*, that links an internal state to a perception in order to reach another internal state:

$$next\colon\ I \times P \rightarrow I$$

while the perception function is:

$$see\colon\ S \rightarrow P$$

and the action one:

$$action\colon\ I \rightarrow A$$

Thus, the agent starts from an initial state i_0 and analyzes the environment; on the basis of the environment state s_1 it produces a perception; the internal state of the agent is then modified by the function $next(i_0, see(s_1))$. The action selected by the agent will be:

$$action\ (next(i_0, see(s_1))$$

Then, the agent restarts the process for a new cycle.

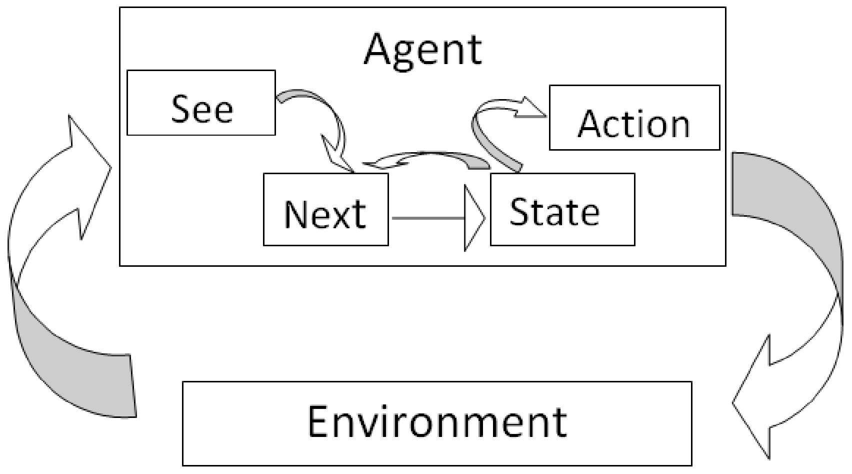


Figure 1.3 – An Agent with State and its Environment

1.5 Hybrid agents' architectures

In this paragraph, other kinds of agents' architectures will be introduced. These architectures try to combine characteristics from agents belonging to different classes. In particular, the aim is to match the capability to react to changes in the environment (typical of purely reactive agents) and a sufficient degree of pro-activeness (typical of standard deliberative agents), in order to pursue long-term objectives.

Thus, two more agents categories can be presented:

- *Belief-Desire-Intentions* agents;
- *Layered Architectures*.

1.5.1 Belief-Desire-Intentions agents

The architecture of this class of agents derives from a *practical reasoning* that in each time instant allows the agent to decide and plan its action on the basis of its own objectives. A *BDI* (acronymous of *Beliefs-Desire-Intentions*) agent makes its evolution depend on:

- *Beliefs*, representing what the agent knows or believes to know;
- *Desires*, representing what the agent wants to obtain;
- *Intentions*, representing what the agent chooses to obtain (a subset of the *Desires* set).

Figure 1.4 shows the behavior of a *BDI* agent as described by Bratman et al. (1988). Seven main components can be identified:

- A set of current *Beliefs*, reproducing the information on the environment;

- A *Believe Revision Function* (*BRF*) that receives a perception and the current beliefs of the agent as input to create a new set of beliefs;
- An *Option Generation* function that determines the *Desires* of the agent on the basis of its own beliefs and its intentions;
- The above introduced *Desires* set;
- A *filter*, representing the deliberative part of the process, that determines the intentions of the agent depending on its beliefs and desires;
- The above introduced *Intentions* set;
- An *execute* function, that selects the action to be enacted taking in account current intentions.

A more formalized structure can be introduced defining the following sets:

- Bel, the set of all the possible beliefs in a given instant;
- Des, the set of all the possible desires in a given instant;
- Int , the set of all the possible intentions in a given instant.

Thus, in a given instant, the state of an agent can be identified with the triple *(B,D,I)*, in which $B \subseteq Bel, D \subseteq Des$, e $I \subseteq Int$.

The above introduced *BRF* can be formulated as follows:

$$\varphi_1(Bel \times P) \rightarrow (Bel)$$

Thus, starting from current beliefs and perceptions (the cartesian product of those sets representing the domain of the function), φ_1 determines a new beliefs set.

The *Option Generation* function (φ_2) is defined as:

$$\varphi_2(Bel \times Int) \rightarrow (Des)$$

φ_2 allows to define, starting from current beliefs and intentions, a new set of desires.

The filter function represents the deliberative process; it produces the new set of intentions, on the basis of the beliefs and of the previously modified desires, as follows:

$$\varphi_3(Bel \times Des \times Int) \rightarrow Int$$

The filter function φ_3 requires the respect of the following constraint:

$$\big(\forall\, B \in (Bel), \forall\, D \in (Des), \forall\, I \in (Int)\big), filter\,(B, D, I) \subseteq (Int) \cup (Des)$$

This means that the filter function must not introduce any new belief or intention; on the contrary, the output of the filter function must belong to the union of the previously defined intentions and desires sets.

Finally, the execute function associates an action with each intention, as follows:

$$execute:\ \varphi_4(Int) \rightarrow A$$

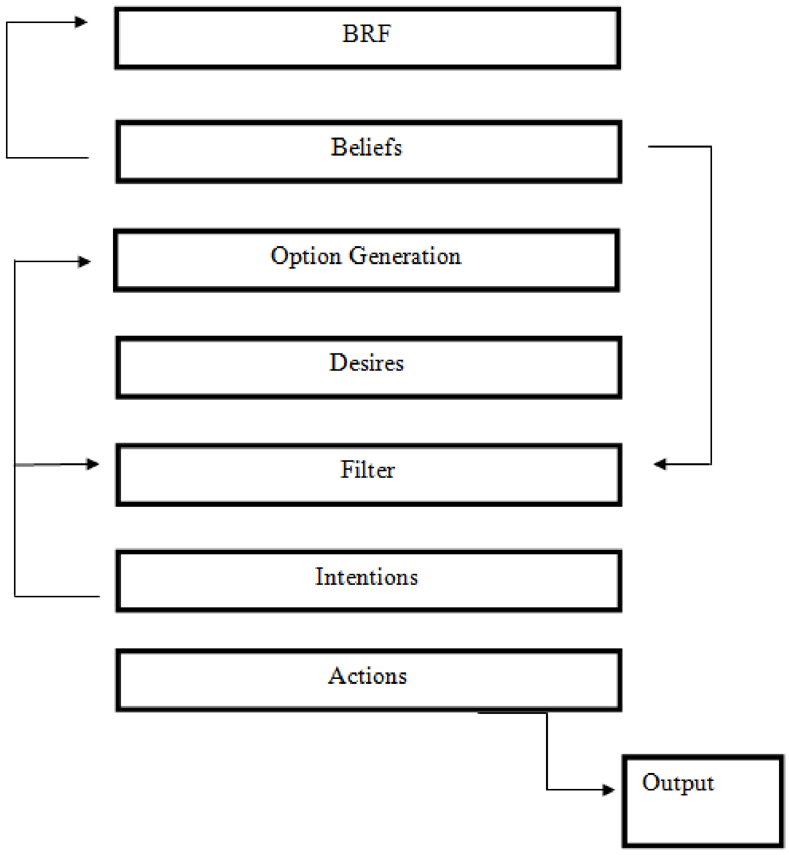


Figure 1.4 - BDI agent theoretical working schema

1.5.2 Layered Architectures

Starting from the need for combining reactive and proactive behaviors, Muller (1995) proposed the idea of agents based on *layered architectures*. In this kind of agents, several characteristics (typical of different categories of agents) are reproduced through layer-based hierarchical structures, in which each layer is representative of a part of the agent corresponding to a given characteristic.

A layered architecture must include at least two layers: a proactive one and a reactive one. Two different hierarchical structures can be utilized to manage interactions among layers:

- Horizontal Structure: each layer is connected to both input and output sensors; therefore, each layer behaves like a single agent;
- Vertical Structure: the input sensor is connected to the first layer, while the output sensor is connected to the last one.

The horizontal structure (represented in Figure 1.5) is straightforward: if an agent has to exhibit *n* different kinds of behaviors, *n* layers have to be implemented.

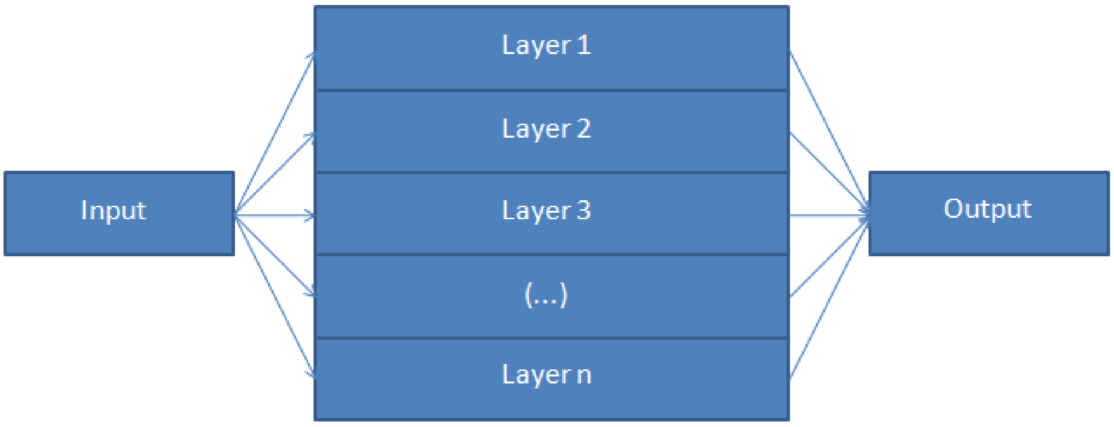


Figure 1.5 - Horizontal structure

This approach presents some weaknesses: the general behavior of an agent could not be coherent, as each layer could compete with the other ones. To this aim, a negotiation function has to be introduced, in order to assign, in a given time instant, the capability of pursuing an action to a given layer. However, even if this modification can solve the coordination problems connected to this particular structure, it can cause an increase in the complexity of the model: if any of the n layers can choose among m actions, the negotiation function has to choose among m^n possible couples layer-action. If m and n are very large, this can be a serious disadvantage.

In the vertical structure, there are two possible configurations:

- *One Pass Architectures*, in which control flows sequentially through each layer, until the final layer generates action output (Figure 1.6);
- *Two Passes Architectures*, in which information flows up the architecture (the first pass) and control then flows back down (Figure 1.6).

In both one pass and two pass vertically layered architectures the complexity of interactions between layers is reduced: since there are *n-1* interfaces between n layers, then if each layer is capable of suggesting m actions, there are at most $m^2(n-1)$ interactions to be considered between layers. This is clearly much simpler than the horizontally layered case.

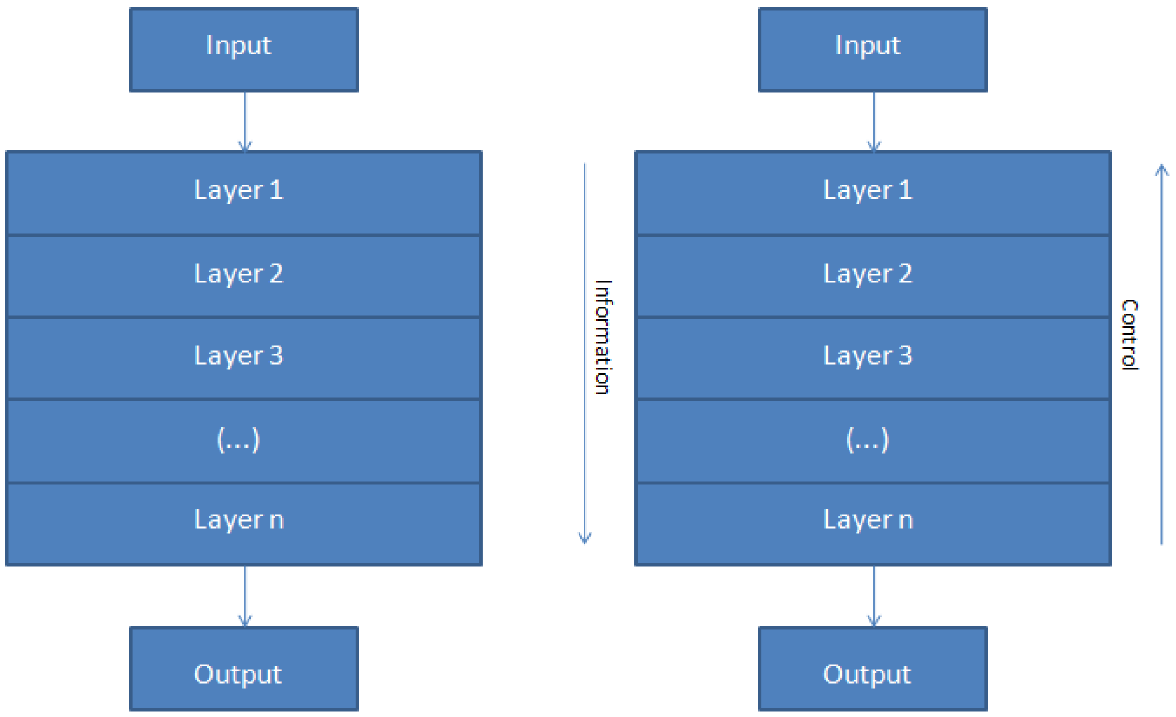


Figure 1.6 - Vertical Structures: one pass (left); two passes (left)

1.6 Multi-agent systems

Agents operate and exist in some environment, which ypically is both computational and physical. The environment might be open or closed, and it might or might not contain other agents. Although there are situations where an agent can operate usefully by itself, the increasing interconnection and networking of computers is making such situations rare, and usually the agent interacts with other agents. Whereas the previous paragraph defined the structure and characteristics of an individual agent, the focus of this paragraph is on systems with multiple agents, in order to learn how to analyze, describe, and design environments in which agents can operate effectively and interact with each other productively.

But why should we be interested in distributed systems of agents? Indeed, centralized solutions are generally more efficient: anything that can be computed in a distributed system can be moved to a single computer and optimized to be at least as efficient. However, distributed computations are sometimes easier to understand and easier to develop, especially when the problem being solved is itself distributed. Distribution can lead to computational algorithms that might not have been discovered with a centralized approach. There are also times when a centralized approach is impossible, because the systems and data belong to

independent organizations that want to keep their information private and secure for competitive reasons.
The information involved is necessarily distributed, and it resides in information systems that are large and complex in several senses:

- they can be geographically distributed;
- they can have many components;
- they can have a huge content, both in the number of concepts and in the amount of data about each concept;
- they can have a broad scope, i.e., coverage of a major portion of a significant domain.

Also, the components of the systems are typically distributed and heterogeneous. The topology of these systems is dynamic and their content is changing rapidly.
As will be shown in the following, multi-agent systems prove to be a viable approach for studying systems endowed with these characteristics (Weiss, 1999).

1.6.1 The need for coordination

Within a multi-agent system, each agent has objectives to pursue. Obviously, these objective can contrast with other agents' ones. Thus, to pursue their own objectives, agents have to communicate.
Communication capabilities include the abilities to receive and send messages. This is necessary to ensure a coordination mechanism among agents themselves, in order to prevent and avoid conflicts among agents' objectives.
Coordination mechanisms can be essentially partitioned into two main categories:

- Cooperation, a coordination form among non-competitive agents;
- Competition, a coordination form among competitive agents (agents endowed with conflicting objectives) that exchange messages in order to get a final agreement.

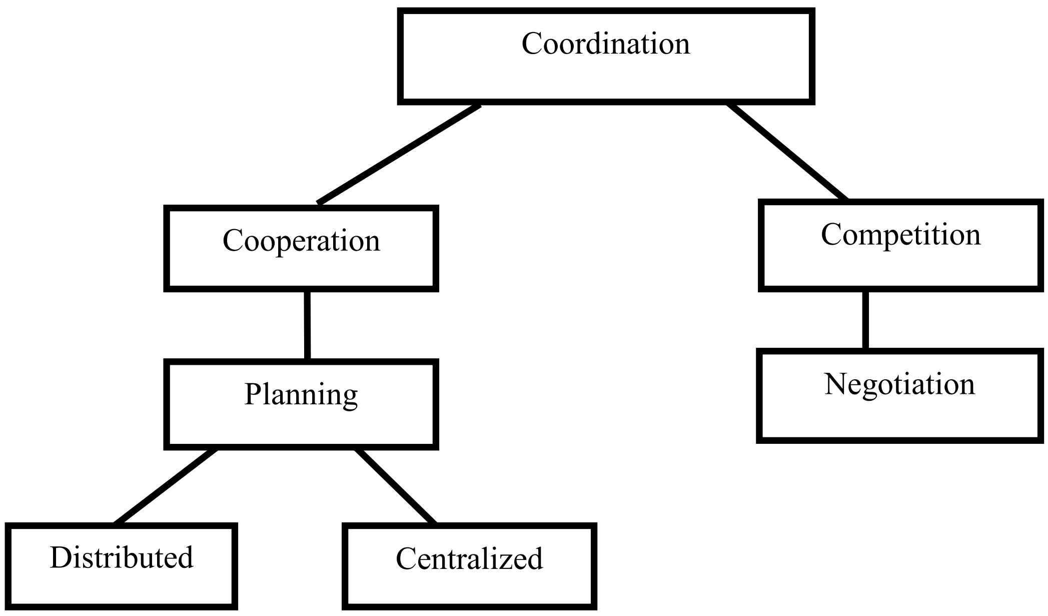


Figure 1.7 - Coordination mechanisms (Weiss, 1999).

Implementing a cooperation-based coordination mechanism means employing planning approaches to reduce resource contention and to ensure the achievement of global objectives. These planning approaches can be distinguished into two main categories:

- Distributed approaches, in which agents are endowed with self-organizing approaches for resource sharing and goal pursuing;
- Centralized approaches, in which a *mediator* agent is assigned with the task of regulating and supervise agents' behaviors.

Implementing a competition-based coordination mechanism means reproducing negotiation forms among agents.

In order to coordinate their actions, agents employ two different protocols:

- Interaction protocols, that govern the exchange of a series of messages among agents — a conversation;
- Communication protocols, that rule the way in which a single message is composed.

Communication protocols enable agents to exchange and understand messages. Interaction protocols enable agents to have conversations, which for our purposes are structured exchanges of messages.

1.6.2 Interaction Protocols

If the agents have their own objectives to pursue, the interaction protocol is aimed at maximizing the pay-off of the single agent; otherwise, if agents are endowed with similar objectives, the protocol is oriented to pursue a global objective without the introduction of a centralized control. In the last case, it is necessary to decide how to:

- Determine shared objectives;
- Determine common tasks;
- Avoid unnecessary conflicts;
- Share knowledge and experience among agents.

As a concrete example of these, a communication protocol might specify that the following types of messages can be exchanged between two agents:

- Proposal of a course of action;
- Acceptance a course of action;
- Rejection of a course of action;
- Disagreement with a proposed course of action;
- Counterproposal of a course of action.

Given two agents *a* and *b*, interaction protocols phases can be synthesized in the following stages (Weiss, 1999) (Figure 1.8):

- Agent *a* proposes a course of action to agent *b*;
- Agent *b* evaluates the proposal and:
    - sends acceptance to agent *a*, or
    - sends rejection agent *a*, or
    - sends disagreement to agent *a*, or
    - sends counterproposal to agent *a*.

In the following, two main interaction protocols will be described: the first one based on a cooperative approach, the second one on negotiation approach, that allows to take in account competitive agents.

Cooperation-based interaction protocols

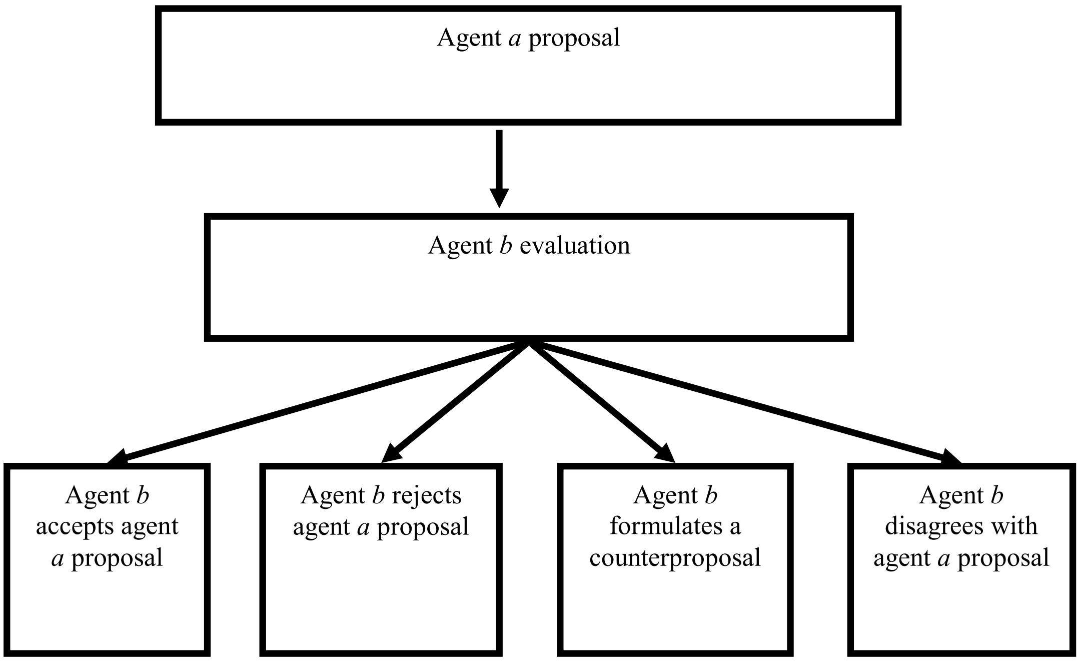
A basic strategy shared by many of the protocols for cooperation is to decompose and then distribute tasks. Such an approach can reduce the complexity of a task: smaller subtasks require less capable agents and fewer resources. However, the system must decide among alternative decompositions, if available, and the decomposition process must consider the resources and capabilities of the agents. Also, there might be interactions among the subtasks and conflicts among the agents.

Task decomposition might be done spatially, based on the layout of information sources or decision points, or functionally, according to the expertise of available agents. Once tasks are decomposed, they can be distributed according to the following criteria:

- Avoid overloading critical resources;
- Assign tasks to agents with matching capabilities;
- Make an agent with a wide view assign tasks to other agents;
- Assign overlapping responsibilities to agents to achieve coherence;
- Assign highly interdependent tasks to agents in spatial or semantic proximity. This minimizes communication and synchronization costs;
- Reassign tasks if necessary for completing urgent tasks.

The following mechanisms are commonly used to distribute tasks:

- Market mechanisms: tasks are matched to agents by generalized agreement or mutual selection (analogous to pricing commodities);
- Contracting mechanism: announce, bid, and award tasks;
- Multi-agent planning: planning agents have the responsibility for task assignment;
- Organizational structure: agents have fixed responsibilities for particular tasks.



Agent *a* proposal
Agent *b* evaluation
Agent *b* accepts agent *a* proposal
Agent *b* rejects agent *a* proposal
Agent *b* formulates a counterproposal
Agent *b* disagrees with agent *a* proposal

Figure 1.8 - Interaction protocol example (Weiss, 1999).

Negotiation-based interaction protocols:

A frequent form of interaction that occurs among agents with different goals is defined as *negotiation*. Negotiation is a process by which a joint decision is reached by two or more agents, each trying to reach an individual goal or objective (Weiss, 1999). The agents first communicate their positions, which might conflict, and then try to move towards agreement by making concessions or searching for alternatives.

Negotiation's main features can be listed as follows:

- The language used by participant agents;
- The protocol used to represent negotiations;
- The decision making rules each agent utilizes to determine its positions and the criteria to reach an agreement.

Two approaches can be utilized to set negotiation techniques up:

- Environment-centered;
- Agent-centered.

Environment-centered techniques focus on the following problem: "*How can the rules of the environment be designed so that the agents in it, regardless of their origin, capabilities, or intentions, will interact productively and fairly?*". The resultant negotiation mechanism should have the following attributes:

- Efficiency: agents have to minimize resource wasting to reach an agreement;
- Stability: no agent should have an incentive to deviate from agreed-upon strategies;
- Simplicity: the negotiation mechanism should impose low computational on the agents.
- Distribution: there is no central controller;
- Symmetry: the mechanism should not be biased against any agent.

According to Rosenschein e Zlotkin (1999), three environment types can be identified on the basis of the previously defined characteristics:

- Task-oriented domain: it is characterized by non-conflicting tasks among agents; tasks can be divided among agents; the object of the negotiation is the distribution of tasks among agents, in order to achieve an allocation that allows each agent maximizing its pay-off;
- State-oriented domain: it is characterized by the presence of complex interactions in agents' actions. This means that agents' actions can both help or disturb other agents'

plans. Therefore, the negotiation process will be focused on developing shared plans, in such a way to maximize positive interactions among agents.

- Worth-oriented domain: each agent is endowed with a function that measures the acceptability degree of the reached state. Thus, each agent can also accept sub-optimal solutions, if this allows obtaining a better solution at an aggregated level.

Agent-centered negotiation mechanisms focus on the following problem: "*Given an environment in which my agent must operate, what is the best strategy for it to follow*?" (Weiss, 1999). Several mechanisms based on this architecture have been developed; one of the most utilized ones assumes that agents are economically rational individuals. Agents formulate a set of courses of actions; a cost function is associated with each course of action. Each agent is oriented to maximize its utility and to minimize its costs. There are three possible solutions to rule agents' interactions:

- Conflict: if there are no plans that are efficient for more agents; thus, agents compete to get the best plans;
- Compromise: agents try to pursue their own objectives by competing; if no efficient solutions are possible, they choose to cooperate;
- Cooperation: agents always choose to pursue common objectives.

1.6.3 Communication protocols

Communication protocols aim at creating common languages among agents. There are three fundamental aspects that have to be defined to create a communication protocol:

- Syntax, regarding how communication symbols are structured;
- Semantics, regarding what the symbols denote;
- Pragmatics, regarding how the symbols are interpreted.

The basic element of a communication protocol is a message that is made up of sequences of symbols. Messages can be classified according to several dimensions, as reported in table 1.1. There are two basic message types: assertions and queries. Assertions express actions requests that agents exchange each other; queries are a more complex message type, as they require the capabilities both to express and answer questions.

Every agent, whether active or passive, must have the ability to accept information. In its simplest form, this information is communicated to the agent from an external source by means of an assertion. In order to assume a passive role in a dialog, an agent must additionally be able to answer questions, i.e., it must be able to accept a query from an external source and send a reply to the source by making an assertion.

Dimensions	**Effects**
Descriptive vs. Prescriptive	Some messages describe phenomena, while others prescribe behavior. Descriptions are important for human comprehension, but are difficult for agents to mimic. Appropriately, then, most agent communication languages are designed for the exchange of information about activities and behavior.
Personal vs. Conventional Meaning	An agent might have its own meaning for a message, but this might differ from the meaning conventionally accepted by the other agents with which the agent communicates. To the greatest extent possible, multi-agent systems should opt for conventional meanings, especially since these systems are typically open environments in which new agents might be introduced at anytime.
Subjective vs. Objective Meaning	Similar to conventional meaning, where meaning is determined external to an agent, a message often has an explicit effect on the environment, which can be perceived objectively. The effect might be different than that understood internally, i.e., subjectively, by the sender or receiver of the message.
Speaker's vs. Society's Perspective	Independent of the conventional or objective meaning of a message, the message can be expressed according to the viewpoint of the speaker or hearer or other observers.
Cardinality	A message sent privately to one agent would be understood differently than the same message broadcast publicly.
Contexuality	Messages cannot be understood in isolation, but must be interpreted in terms of the mental states of the agents, the present state of the environment, and the environment's history: how it arrived at its present state. Interpretations are directly affected by previous messages and actions of the agents.
Coverage	Smaller languages are more manageable, but they must be large enough so that an agent can convey the meanings it intends.

Table 1.1 - Messages dimension (Weiss, 1999)

In order to assume an active role in a dialog, an agent must be able to issue queries and make assertions. With these capabilities, the agent then can potentially control another agent by causing it to respond to the query or to accept the information asserted. This means of control can be extended to the control of subagents, such as neural networks and databases. An agent functioning as a peer with another agent can assume both active and passive roles in a dialog. It must be able to make and accept both assertions and queries.
Table 1.2 summarizes agents' capabilities and categories.

	Basic Agent	**Passive Agent**	**Active Agent**	**Peer Agent**
Receives Assertions	X	X	X	X
Receives Queries		X		X
Send Assertions		X	X	X
Sends Queries			X	X

Table 1.2 - Agents' capabilities and categories (Weiss, 1999)

The data structure of a communication protocol presents five fields:

- Sender;
- Receiver;
- Language in the protocol;
- Encoding and decoding functions;
- Actions to be taken by the receiver.

The protocol is defined as binary if the communication takes place exclusively between two agents; it is defined as n-ary if the communication is among a sender and more receivers.

1.7 Agents' societies

As previously illustrated, intelligent agents do not function in isolation. They are part of the environment in which they operate, and the environment typically contains other such intelligent systems.
When environments are too large, complex, dynamic, and open to be managed centrally or via predefined techniques the only feasible alternative is to provide distributed control. A way to build such control structure is provided by societies of agents.
A group of agents can form a society in which each agent plays a different role. The group defines the roles, and the roles define the commitments associated with them (Weiss, 1999). When an agent joins a group, he joins in one or more roles, and acquires the commitments of that role. Agents join a group autonomously, but are then constrained by the commitments for the roles they adopt. The groups define the social context in which the agents interact.

The basic mechanism that rules agents' societies is the concept of *social commitment*. Social commitments are the commitments of an agent to another agent, and represent a flexible means through which the behavior of autonomous agents is constrained.

Social commitments enact social dependencies: suppose that an agent x depends (by a social commitment) on an agent y to complete an action a that allows reaching the state p that represents the objective of the agent x. If the agent x is not capable of reaching the state p by itself, the agent x socially depends on the agent y.

Social dependencies are enacted on a voluntary basis, as agents choose to become part of an agents' society. Moreover, social dependencies can be mutual if the agent x has to perform the action a_x that is necessary to the agent y and the agent y itself has to perform the action a_y that is necessary to the agent x, being the respective objectives p_x and p_y.

Thus, an agent society has the following characteristics:

- All the agents share a common objective;
- Each agent acquires a role as member of the group or of a subgroup;
- Accepting commitments deriving by its role, each agent requires being part of the society.

The development of the society can be described through three levels, as proposed by Dignum et al. (2002) and illustrated in figure 1.9:

- Organizational model, that describes society's structure in terms of roles, relationships between roles and prescriptive behavioral norms;
- Social model, that allows inserting agents within the organizational structure previously introduced, by defining contract protocols for establishing commitments;
- Interaction model, that defines the interaction protocol the society is based on.

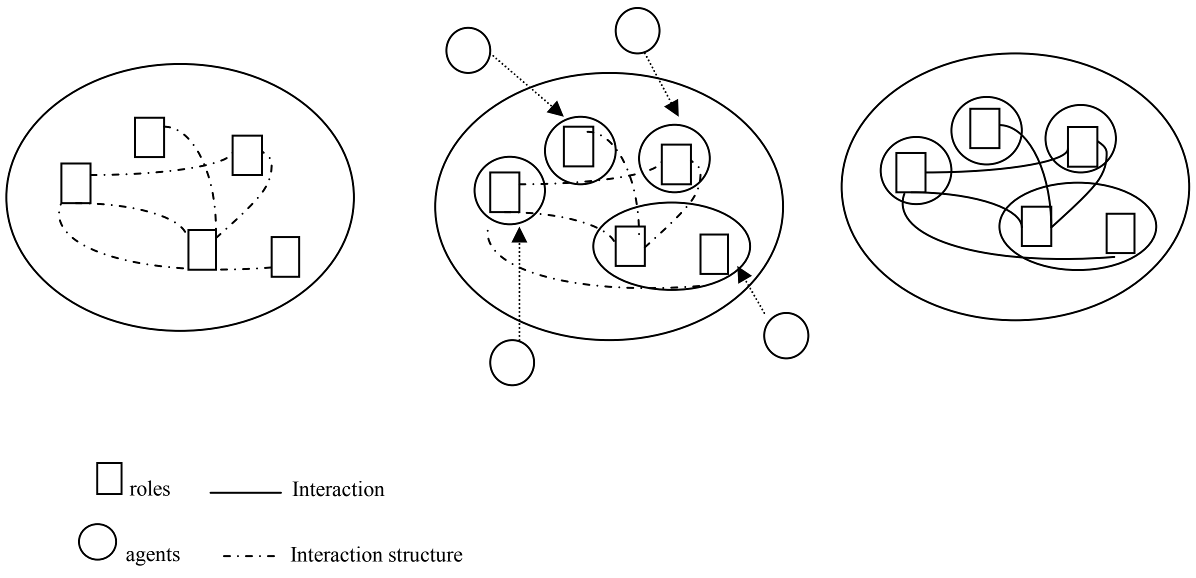


Figure 1.9 - Theoretical model for an agent society

The main difference between multi-agent systems and society of agents relies in the fact that agents operating in MASs operate to pursue their own objective, even if a coordination mechanism is provided; in agents' societies, agents belong to a group and pursue an objective that is expressed by the whole society.

1.8 Summary

Synthesizing, the development of an Agent-Based Model needs a complete description for a set of basic *building blocks*, listed as follows (Billari *et al.* (2006) and Weiss (1999)).

- *The object of the simulation.* It has to be specified what is the phenomenon/problem to be reproduced, defining the environment where the simulation takes place and its characteristics, following the schema previously depicted.
- *The agents' population.* Agents can be grouped in different categories with common characteristics reproducing the various components of the system.
- *The adaptive capability of each agent category.* Agents of each category present a specific adaptive capability, i.e. the degree of re-activeness and pro-activeness.
- *The interaction and communication paradigm among agents.* Each agent can interact with agents of the same or of other categories according to the above listed different paradigms. On the base of the selected paradigm, the agents evolve in the simulation space in a different way.

1.9 Conclusions

In this chapter, an illustration of the *agent* concept has been provided. In particular, basic agents' characteristics have been depicted.

Evolution rules of the agents have been described through a mathematical formalism, in order to set the stage for the introduction of coordination forms among agents that constitute the building blocks for multi-agent systems and agents' societies.

In the following of this work, the applicability of multi-agent systems to decision-making problems will be surveyed.

Chapter 2
Optimization models and methods: generalities

2.1 Introduction

The daily work of professionals involves making a series of *decisions*. In fact, the world relies on systems designed by engineers and business people. Thus, the quality of decisions made by these two categories of professionals is of critical importance.

Decisions are made by looking at the relevant data and making judgments. Making decisions on issues with important consequences has become a highly complex problem due to the many competing forces under which the world is operating today. Nevertheless, it is still usual for professionals to formulate decisions just relying on their own *gut feeling*. This method very often leads to decisions quite far from being optimal. In fact many bad decisions are still being made daily due to this.

Anyone who holds a technical, managerial, or administrative job these days is faced with making decisions daily at work. Decisions may involve:

- Determining which ingredients and in what quantities to add to a mixture being made so that it will meet specifications on its composition;
- Selecting one among a small number of suppliers to order raw materials from;
- Determining the quantities of various products to manufacture in the next period;
- Allocating available funds among various competing agencies;
- Deciding which route to take to go to a given location;
- Selecting an appropriate location for an industrial facility;
- Determining how many check-in desks to open during airport operating hours.

A situation such as one of these requiring some decisions to be made is known as a Decision Making Problem.

As introduced above, in the past decisions were made exclusively on intuitive judgment based on past experience. Today it is essential to make decisions on a rational basis. The most rational way for decision making is through quantitative analysis which consists of the following steps.

- *Precise definition of the problem*. This step requires gathering all relevant data and information on the problem itself. In particular, the initial statement of the problem may be vague or imprecise, and it can need some refinement.
- *Construction of a mathematical model of the problem*. Such a model abstracts the essence of the decision problem. The model should express the various quantities involved in the problem in the form of mathematical functions of decision variables, and express the relationships among them using appropriate equations or inequalities. A problem expressed through a mathematical model is said to be in its *Mathematical Programming* form. However, real world problems are usually too complex to be captured in a mathematical model; thus, a model is a simplification that provides a sufficiently precise representation of the main features such that the conclusions obtained from it also remain valid to the original problem to a reasonable degree of approximation.
- *Solution of the model*. This phase allows deriving the solution, namely the decisional outcome for the problem. Depending on the different complexity of the model representing the problem, different solution strategies can be adopted. For some of the models we have efficient *algorithms* (namely, a procedure) and high quality software systems implementing them. For some others we do not yet have efficient algorithms, and when the model is large, applying existing algorithms might take unreasonable times.
- *Implementation of the solution*. The obtained solution is checked for practical feasibility. If it is found to be unfeasible, the model could require some modifications before being solved again. If the quality of the solution is not satisfactory, some refinements are needed in the solution technique.

In the following of the chapter, details will be provided about Mathematical Programming problems, their complexity, and the different solution procedure that can be adopted.

2.2 Mathematical Programming

A Mathematical Programming (MP) problem is a problem that can be reduced to the following general form:

The research of the values of the variables ($x_1, x_2, \dots, x_n$) that allow to maximize or minimize a function $z=f(x_1, x_2, \dots, x_n)$ respecting the following conditions:

$$g_i(x_1, x_2, \dots, x_n) \{\leq, =, \geq\} b_i \quad (i = 1, \dots, m)$$

in which b_i represent constant scalars and g_i are scalar functions.

Variables $(x_1, x_2, \ldots , x_n)$ are usually referred as control or *decision variables*; they can be grouped in a decision variables vector $\underline{\mathbf{x}}$. A set of *n* values assigned to vector $\underline{x}$ components is a solution to the problem. It can be represented as a point in a Euclidean *n*-dimensional space E^n.

Relationships $g_i(\underline{\mathbf{x}})\ \{::\}\ b_i$ have the task to limit the values that decision variables can assume; they are usually referred as problem *constraints*. A solution that respects all the constraints is referred as a *feasible solution*; if the solution does not meet this condition, it is referred as an *infeasible solution*.

The set X_a (subset of E^n) including all the feasible solutions to the problem represents the *feasible domain* (also called *region*, or *set*) associated with the problem. If the problem has no constraints, the feasible domain coincides with E^n. It is also possible that the constraints determine an empty feasible set; in this case, the problem is said to be *infeasible*.

The function $z=f(\underline{\mathbf{x}})$ to be optimized is generally called *objective function*. Every solution that reports the optimal value of *z* is defined as *optimal solution*. The optimal value of *z* can also be a not finite value.

A solution $\underline{\mathbf{x}}^*$ is said to be a *global maximum* [*global minimum*] for the objective function $f(\underline{\mathbf{x}})$ if the following condition is met:

$$f(\underline{\mathbf{x}}^*)\geq f(\underline{\mathbf{y}}) \qquad [f(\underline{\mathbf{x}}^*)\leq f(\underline{\mathbf{y}})], \quad \forall y \in X_a$$

On the other hand, a solution $\underline{\mathbf{x}}'$ is said to be a *local maximum* [*local minimum*] for the objective function $f(\underline{\mathbf{x}})$ if it is possible to identify a neighborhood of $\underline{\mathbf{x}}'$ of radius ε, I_ε, that meets the following condition:

$$f(\underline{\mathbf{x}}')\geq f(\underline{\mathbf{y}}) \qquad [f(\underline{\mathbf{x}}')\leq f(\underline{\mathbf{y}})], \quad \forall y \in I_\varepsilon$$

The formal definition of Mathematical Programming carries on some implicit limitations:

- Strict inequalities are not allowed; this means that the feasibility region of a mathematical programming problem has to be a closed domain;
- The problem is assumed to be deterministic.

2.3 Mathematical Programming problems classification

Mathematical Programming problems can be classified according to their characteristics. Three main aggregation keys can be defined to this aim:

- Objective function and constraints characteristics;
- Problem dimension (i.e. number of variables and constraints);

- Variables characteristics.

In the following, possible classification schemes are introduced for each category.

2.3.1 Objective function and constraints characteristics classification

A first classification can be introduced on the basis of the characteristics of the $f(\underline{\mathbf{x}})$ and g_i functions. This also influences the choice of the algorithms utilized to solve the problems. As illustrated in Figure 2.1, this classification mainly separates Mathematical Programming problems into *Linear Programming* and *Non-Linear Programming* problems.

Linear Programming

A Linear Programming (LP) problem is characterized by the fact that all the functions involved in the problem (objective function, constraints) are linear. Thus, a LP problem can be formulated as follows:

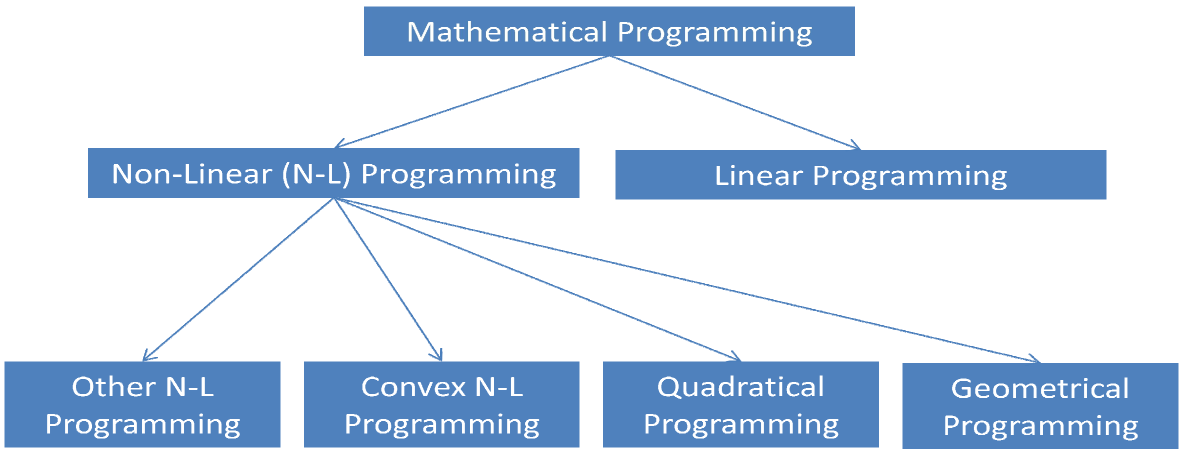


Figure 2.1 - Objective function and constraints characteristics based classification

$$z=\sum_j c_j x_j \qquad \textit{Max!} \quad [\textit{Min!}]$$

Subject to:

$$\sum_j a_{ij} x_j \{\leq,=,\geq\}\ b_i \qquad \forall i=1,2,\ldots,m$$

$$x_j \geq 0 \qquad \forall j=1,2,\ldots,n$$

Non-Linear Programming

This class includes all the problems that are not characterized by having both the objective function and constraints expressed by linear functions. This second class can be further partitioned into:

- *Geometrical Programming* problems, whose objective function and constraints are expressed by posynomial functions;
- *Quadratic Programming* problems, whose objective is expressed by a quadratic function, while constraints are linear;
- *Convex Non-Linear Programming* problems, whose objective and constraints are convex functions;
- *Other Non-Linear Programming* problems, not included in the previously stated sub-categories.

It is interesting to note that Linear Programming problems are a particular class of convex problems.

2.3.2 Problem dimension classification

Mathematical Programming problems can be grouped into three classes: *small*, *intermediate* and *large scale* problems.

A Mathematical Programming problem is defined as:

- A small scale problem, if it can be solved in a reasonable time by a pen-and-paper algorithm or just utilizing a table calculator;
- An intermediate scale problem, if it can be *directly* solved by utilizing available solution methods on a personal computer;
- A large scale problem, if it can be solved with available methods only in presence of particular structures that allow finding a solution operating on sub-problems.

It is easy to understand that small scale problem class does not change over time, while between the other two classes there is a frontier depending on personal computers performances.

2.3.3 Variables characteristics classification

Based on variables characteristics, it is possible to distinguish between *Continuous Mathematical Programming* (in which variables can assume any value in a non-enumerable set) and *Discrete Mathematical Programming* (in which variables are constrained to assume values in an enumerable set) (Figure 2.2).

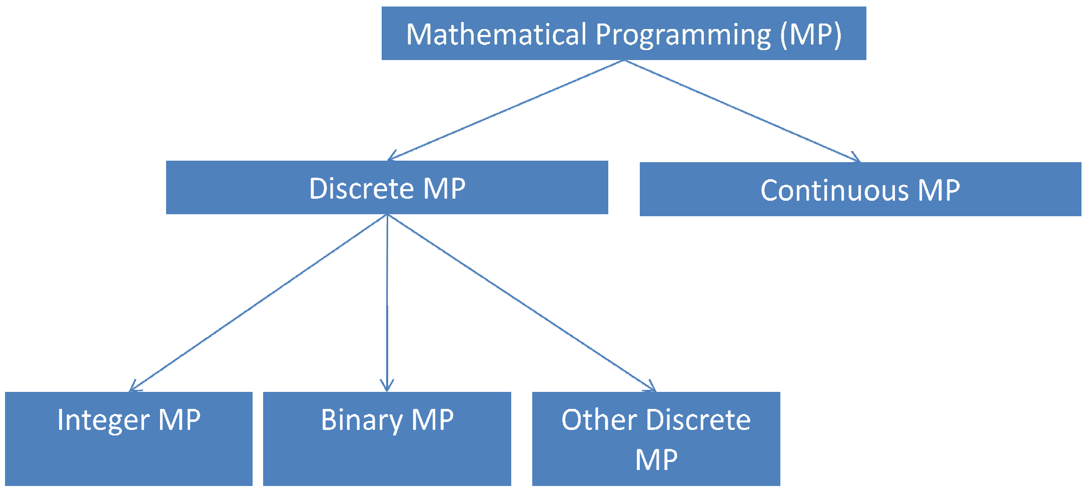


Figure 2.2 - Variables characteristics based classification

Discrete Mathematical Programming include, for instance, Linear Integer Programming problems, Binary Integer Programming problems and several other classical optimization problems (i.e. Bin Packing Problem, Traveling Salesman Problem, Chinese Postman Problem).

2.4 Complexity of Mathematical Programming problems

Solving a Mathematical Programming (MP) problem means developing an *algorithm*, namely a procedure (usually formalized through a programming language and implemented on a computer) devoted to the resolution of a problem through a sequence of operations.

A problem for which it is not possible to find an algorithm capable of solving it is said to be *unsolvable* or *undecidable*.

Obviously, it is not enough to design an algorithm that just provides a solution to the problem; algorithms have also to exploit available resources in an efficient way.

Algorithms efficiency can be evaluated on the basis of several criteria, as:

- The length of the code to be written to implement the procedure;
- The time required to an expert programmer to write the code;
- The debugging time required to fix programming errors;
- The time required to experimentally evaluate the algorithm performances.

Among the others, the most utilized criterion to estimate algorithms performances is the computational time evaluated being all the other resources (available memory, platform, programming language) the same. The faster the algorithm, the more it is considered to be efficient.

Obviously, computational time depends on the particular instance of the problem under consideration, and it increases if the *dimension n* (defined as data quantity required to represent it) of the instance increases too.

2.4.1 A first classification of Mathematical Programming problems

The *computational complexity* of an algorithm is defined as the variation of the computational time varying the dimensions of the instance of the problem.

Decidable problems are those problems solvable with a certain algorithms, no matter of its complexity. Thus, within this class of problems, it is possible to provide a further classification, defining a *complexity function f(n)* associated with an algorithm related to a decidable problem. It represents the number of operations required by the algorithm to solve, in the worst case, an instance of dimension n. In other words, $f(n)$ expresses the number of operations needed by the algorithm depending on the size of the instance of the problem; this function can present different shapes.

In order to compare functions endowed with different shapes the following notation can be introduced:

$$f(n)=O(g(n))$$

$$\text{if } \exists\ n^*\geq 0,\ c\geq 1 : f(n)\leq c\cdot g(n)\ \forall n\geq n^*$$

An algorithm that presents a complexity function $f(n)=O(n^k)$ requires a number of operations that is upper-bounded by a k-order polynomial function; synthetically, it is defined as a polynomial algorithm.

In a similar way,

$$f(n)=\Omega(g(n))$$

$$\text{if } \exists\ n^*\geq 0,\ c\geq 1 : f(n)\geq c\cdot \Omega(n)\ \forall n\geq n^*$$

Practically, $O(\cdot)$ and $\Omega(\cdot)$ allow to define an upper and a lower bound for the complexity function of an algorithm.

On the basis of these definitions, a mathematical programming problem can be defined as a *tractable problem* if there exists a polynomial algorithm capable of solving it; otherwise, the problem is said to be *intractable*.

The reason of this definition arises from simple considerations: exponential complexity algorithms (i.e. $f(n)=a^n$, with $a>1$) present incredibly high computational times even for limited size instances.

2.4.2 P and NP problems

Despite of its apparent simplicity, the previously illustrated classification is not applicable to several problems. Indeed, defining a given problem as *intractable* means being able to show that a polynomial algorithm capable of solving the problem itself cannot exist. This proof requires showing that any algorithm to solve the problem is lower-bounded by an exponential function. Given the difficulty of this process, only a limited number of problems have been qualified as intractable.

On the other hand, given the current state of the art, there are a large number of problems that cannot be classified as tractable neither intractable, as:

- A polynomial algorithm capable of solving the problem has not yet been developed;
- The intractability proof has not yet been provided.

Thus, at the moment these problems are *considered* intractable, even though they could become tractable as soon as a polynomial algorithm capable of solving it would be developed.

In order to overcome limitations of this classification schema, the *NP*-completeness theory has been developed. This theory is referred to the so-called *Decision Problems*, representing a category of problems formulated in such a way that just two answers are allowed: *Yes* or *No*.

Thus, a decision problem presents a different formulation if compared to Mathematical Programming problem; yet, it is possible to associate a decision problem with a Mathematical Programming problem; indeed, starting from a Mathematical Programming problem π, it is possible to define the following Decision Problem:

"*Does the problem π allow for a solution whose value is less or equal to x?*"

It is clear that such a kind of Decision Problem is easier to solve than the corresponding optimization problem.

Decision Problems can be grouped into two classes: *P* and *NP* problems. A Decision Problem belongs to *P* class if it is tractable, namely if there exists a polynomial algorithm that can provide a *Yes* or *No* answer to the problem itself. On the other hand, it belongs to *NP* class if there exists a polynomial algorithm capable of just verifying every *Yes* solution.

It is easy to understand that a *P* problem is also belonging to the *NP* class: indeed, if there exists an algorithm capable of solving the problem, it will also serve as verification procedure for every *Yes* solution.

Thus $P \subseteq NP$. However, given the current state of the art, it is not possible to state if $P \subset NP$ and then, $P \neq NP$ or, on the contrary, $P = NP$, as no proofs have been provided on this topic. Practically, it would be necessary to provide a Decision Problem belonging to the *NP* class that does not belong to the *P* class. In this case, the distinction between the two problem classes would make sense. However, even if, as stated before, no proof has been provided, the OR/MS community agrees on considering this hypothesis as the most realistic one. A coherent graphical representation of the two classes of Decision Problems is presented in Figure 2.3.

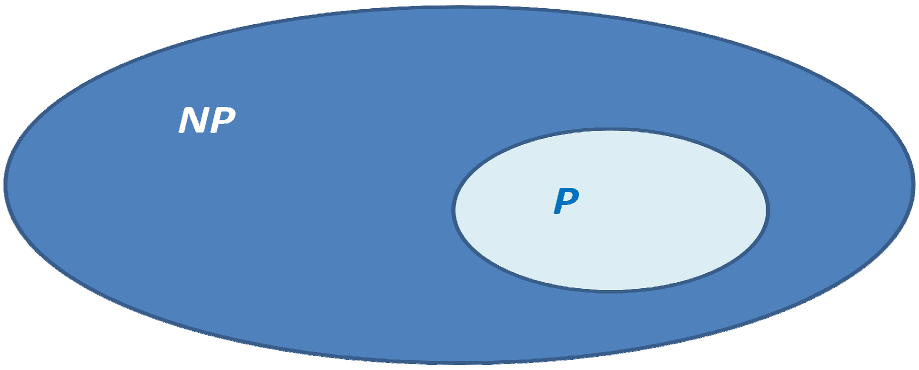


Figure 2.3 - Representation of *NP* and *P* classes

Further studies have allowed introducing another class of problems, the well-known *NP*-complete problems. In order to introduce this class it is necessary to illustrate the *reducibility* concept. A problem π can be reduced to a problem π'' if every π instance can be transformed in a π'' instance within a polynomial computational time.

On the basis of this definition, it can be stated that a problem π belongs to *NP*-complete class if the following two conditions are met:

- π   *NP;*
- Every *NP* problem can be reduced to π.

In other words, *NP*-complete problems represent a subset of the *NP* class, including the *hardest* problems (Figure 2.4). The possibility of reducing every *NP* problem to any *NP*-complete problem implies that the discovery of a polynomial algorithm capable of providing a *Yes* or *No* answer for any *NP*-complete problem would ensure a polynomial resolution method for all *NP* problems; this would provide a proof to the statement *P=NP*.

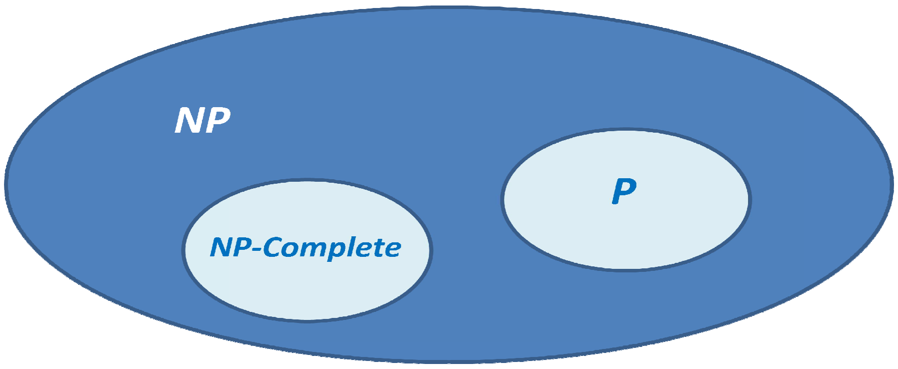


Figure 2.4 - Representation of *NP*, *NP-Complete* and *P* classes

Starting from *NP*-completeness definition, to show that a Decision Problem π^* is *NP*-complete the following steps have to be performed:

- Showing that π^* NP
- Showing that there exist another problem, π^{**} *NP*-complete class, that can be reduced to π^*.

The Satisfiability Problem has been the first problem that has been discovered to be *NP*-complete (Cook, 1971). This seminal result has allowed showing that many Decision Problems associated with common Mathematical Programming problems are *NP*-complete.

It is interesting to note that the reducibility operation is not commutative: thus, there are *NP* problems that are not *NP*-complete.

A Mathematical Programming problem whose corresponding Decision Problem belongs to *NP*-complete class is said to be *NP*-hard. Even if there is no formal proof, it is common opinion that a *NP*-hard problem has to be considered intractable from a computational point of view. Thus, in order to solve *NP*-hard problems of significant dimensions within reasonable computational times, it is necessary to employ heuristic algorithms that can provide good (but sub-optimal) solutions.

2.5 Heuristic techniques

The most of combinatorial optimization problems belong to the *NP*-hard class. As stated before, even if this is not formally proved, *NP*-hard problems are thought to be intractable. Thus, exact approaches can require extremely high computational times if applied to this kind of problems. Therefore, heuristic methods are often employed to deal with *NP*-Hard problems.

While exact methods (like, for example, *Branch and Bound* and *Cutting Planes* methods) provide optimal solution, heuristic methods provide the advantage of more reasonable computational times but often they fail to provide the optimal solution.

Thus, it is crucial to estimate the effectiveness of a heuristic technique. First of all, the quality of the provided solution has to be assessed. Precisely, given a Mathematical Programming problem, an instance *I*, a solution of the instance provided by an heuristic algorithm *EUR(I)* and a solution provided by an exact algorithm *OPT(I)*, a measure of the quality of the solution provided by the heuristic can be expressed through the following index (percent error):

$$e\% = \frac{|OPT(I) - EUR(I)|}{|OPT(I)|}$$

The above introduced index can be estimated in the average case or in the worst case. To have a measure of the percent error in the worst case, an upper bound for the percentage error can be introduced, as follows:

$$e\% = \frac{|OPT(I) - EUR(I)|}{|OPT(I)|} \leq \varepsilon$$

In practice, the heuristic algorithm provides a solution that, in the worst case, presents a percentage error equal to ε. Some algorithms allow individuating an upper bound for the percent error, introducing a kind of *warranty* on the maximum error they can provide.

However, it is not always possible to compute the percent error. Indeed, this implies the availability of an exact algorithm that provides the optimal solution to the problem. If such an algorithm is available, *e%* is computed by generating instances of the problem characterized by different dimensions and evaluating statistics about errors generated by the algorithm for each of the considered instances.

If there is no optimal algorithm available for the problem, the value *OPT(I)* cannot be carried out; thus, to get a value for *e%* there is the need of obtaining an estimation of the optimal solution; this can be done utilizing bounds to the optimal solution in the error evaluation process. For example, for a minimizing problem, an estimation of *e%* can be computed as follows:

$$e^{\sim}\% = \frac{|LBopt(I) - EUR(I)|}{|OPT(I)|}$$

where LBopt is a lower bound to the optimal solution of the problem. It is easy to understand that:

$$\frac{|LBopt(I) - EUR(I)|}{|OPT(I)|} \geq \frac{|OPT(I) - EUR(I)|}{|OPT(I)|}$$

Thus, replacing the optimal solution with estimations produces an overestimation of the error. However, though important, the quality of the solution is not the only parameter to be taken into account while designing and implementing a heuristic algorithm: computational times are also relevant. Usually, there is a strong and positive correlation between the quality of the solution that can be obtained and computational times: better solutions can be reached at expenses of an increase in computational times, often undesirable. Thus, projecting an efficient heuristic algorithm means finding a compromise between the need for solutions very close to the optimal one and fast procedures.

Moreover, other aspects cannot be neglected, like:

- *Easiness of implementation* (in terms of complexity of the programming language code to be written);
- *Flexibility* (in terms of easiness of adaptation of the algorithm to other instances and slightly different problems).

Heuristic algorithms can be classified according to the *search philosophy* adopted to find a solution. In particular, it is possible to distinguish between:

- *Greedy algorithms*, that gradually build a solution to the problem, stepping through partial solutions;
- *Local Search algorithms*, that starts from a solution and try to modify it in order to get some benefits in terms of objective function.

In the following, some principia of each category of algorithms will be illustrated.

2.5.1 Greedy algorithms

A generic greedy algorithm is made up of the following steps:

- *Initialization*: a starting element is chosen (often randomly) and added to the partial solution *S*.
- *Selection*: according to a given criterion, a new element is chosen and added to the partial solution *S*.
- *Halt Criterion*: if *S* is a feasible solution, the procedure stops; otherwise, it goes back to the *selection* stage.

From a mathematical point of view, greedy algorithms build a feasible solution through an iterative process. At every step, the algorithm adds to the solution the element that seems to be more *promising* in terms of objective function values, without any evaluation of the whole

solution. Thus, solutions provided by greedy algorithms are usually poor from a qualitative point of view.
Moreover, as it can be derived from the description, greedy algorithms strongly depend on the choice of the initial element. Therefore, in order to improve the quality of the solution, if possible, it can be convenient to run several times the procedure, selecting different starting elements. If computational times allow it, it can be even convenient to make the algorithm start from each possible starting element.

2.5.2 Local Search algorithms

Local search algorithms are also called *iterative amelioration* procedure. This kind of algorithm is based on the concept of *move*, namely and elementary modification to the solution. The application of a move produces a neighborhood of the current solution that is a set of feasible candidate solutions.
The building blocks of the algorithm can be listed as follows:

- Definition of a *move*;
- Definition of a new solution selection criterion;
- Definition of a stopping criterion, that allows the termination of the algorithm.

The stages of the algorithm can be summed up as follows:

- Stage 1: an initial solution S has to be provided, in order to enact the procedure. It can be obtained randomly or by using a greedy algorithm in a preliminary stage.
- Stage 2: through the application of the previously defined move, it is possible to identify a neighborhood of the current solution *N(S)*, namely a set of candidate solutions.
- Stage 3: a new current solution *S'*∈*N(S)* is chosen, based on the selection criterion.
- Stage 4: the stopping criterion is verified. If it is met, the procedure ends; otherwise, it goes back to Stage 2.

If the move at Stage 2 leads to a reduction in the objective function, it is accepted, and the configuration *S'* obtained is used as the starting point for a new test. In the contrary case, one returns to the preceding configuration, before making another attempt.
The process is made iterative until the stopping criterion is met. However, this algorithm of iterative improvement (also indicated as *classical method*, or *descent method*) does not lead, in general, to the global optimum, but only to a local minimum, which constitutes the best accessible solution taking the initial assumption into account.

In fact, assuming as an example the optimization problem whose objective function is depicted in Figure 2.5 and S_1 as starting solution, it is easy to understand that the above described procedure (also synthesized in Figure 2.4) will converge to the local minimum *S'*. Indeed, if *S'* is reached and a further application of the move produces the solution S_1, the objective function value of the latter is worse than the one reported in *S'*; thus, the algorithm stops.

To improve the effectiveness of the method, one can, of course, apply it several times, with arbitrarily selected different initial conditions, and retain as final solution the best local minima obtained. However, this procedure appreciably increases the computing time of the algorithm, and may not find the optimal configuration. The repeated application of descent method is particularly ineffective when the number of local minima grows exponentially with the size of the problem.

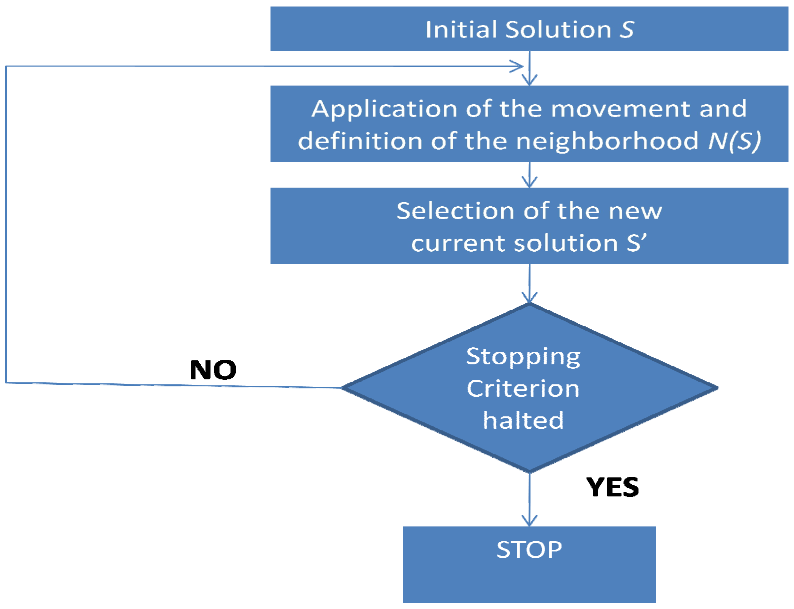

Initial Solution S
Application of the movement and definition of the neighborhood N(S)
Selection of the new current solution S'
Stopping Criterion halted
NO
YES
STOP

Figure 2.4: Schema of a Local Search algorithm

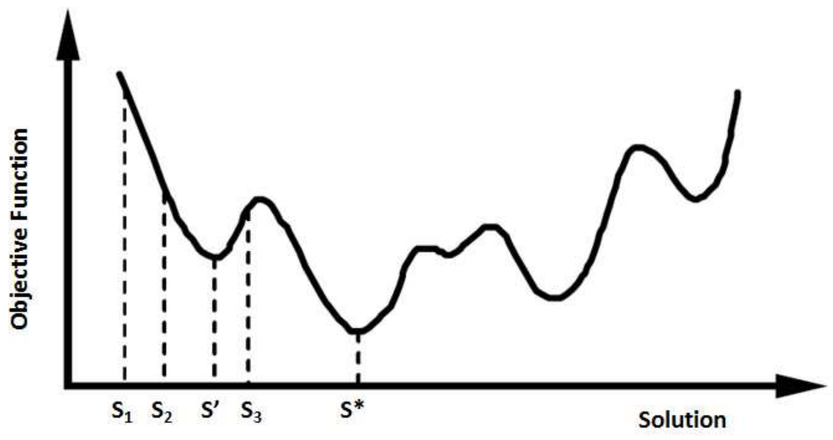


Figure 2.5: Shape of the objective function of an optimization problem

2.6 Metaheuristic algorithms

To overcome the obstacle of the local minima, an idea was demonstrated to be very profitable: the possibility of authorizing, from time to time, moves that produce a worsening in the objective function. This principle is the basic core of the most widespread *metaheuristic* algorithms. As the name of this class procedure suggests, they try to overcome heuristic algorithms limitations.

The introduction of mechanisms for controlling the degradations (specific to each metaheuristic) makes it possible to avoid the divergence of the process. Thus, it consequently becomes possible to be extracted from the trap which represents a local minimum, to leave to explore another more promising *valley*.

The so-called "distributed" metaheuristics (such as the evolutionary algorithms) are endowed with mechanisms allowing the departure of a particular solution out of a local *valley* of the objective function.

In the following, the most widespread metaheuristic algorithms are depicted.

2.6.1 Simulated Annealing

The *Simulated Annealing* method (Kirkpatrick et al., 1983) transposes the process of the *annealing* from physics to the solution of an optimization problem: the objective function of the problem, similar to the energy of a material, is minimized, with the help of the introduction of a fictitious *temperature*, which is, in this case, a simple controllable parameter of the algorithm.

In practice, the technique exploits the Metropolis algorithm, which enables us to describe the behavior of a thermodynamic system in "equilibrium" at a certain temperature. On the basis of a given configuration (for example, an initial placement of all the components), the system is subjected to an elementary modification defined by a move (for example, one relocates a component, or one exchanges two components). If this transformation causes a decrease in the objective function (or *energy*) of the system, it is accepted. On the other hand, if it causes an increase ΔE of the objective function, it can also be accepted, but with a probability $e^{-\Delta E/T}$, according to Metropolis acceptance rule.

This process is then repeated in an iterative manner, by keeping the constant temperature, until thermodynamic balance is reached, concretely at the end of a "sufficient" number of modifications. Then the temperature is lowered, before implementing a new series of transformations: the law of decrease by stages of the temperature is often empirical, just like the criterion of program termination.

The disadvantages of simulated annealing lie on one hand in the "adjustments", like the management of the decrease of the temperature; the user should have the know-how of "good" adjustments.

In addition, the computational time can become very significant, which led to parallel implementations of the method. On the other hand, the simulated annealing method has the advantage of being flexible with respect to the evolutions of the problem and easy to implement. It gave good results for a number of problems, generally of big size. A synthetic schema of the simulated annealing schema is illustrated in Figure 2.6.

2.6.2 The Tabu Search

The method of search with tabus, or simply *Tabu Search* or *Tabu Method*, was formalized by Glover (1986). Its principal characteristic is based on the use of mechanisms inspired by the human *memory*. The Tabu Method takes, from this point of view, a path opposite to that of simulated annealing, which does not utilize memory at all, and thus it is not capable of learning from the past. On the other hand, the modeling of the memory introduces multiple degrees of freedom.

The guiding principle of the tabu method is simple: like simulated annealing, the tabu method works just on a single *current solution*, which is updated during successive "iterations".

At each iteration the mechanism of current solution update process (from a solution *S* to a solution *T*) comprises of two stages:

- the first one, based on an elementary move, builds the set of the *neighbors* of *S*, *N(S)*. *N(S)* is the set of the accessible configurations in only one elementary *move* of *s*;
- the second one evaluates the objective function *f* of the problem for each solution belonging to *N*(*S*). The configuration *T*, which succeeds *S* in the series of the solutions built by the tabu method, is the solution of *N*(*S*) in which *f* takes the minimal value.

It is worth to note that, this solution *T* is adopted even if it is worse than *S*, i.e. if $f(T)>f(S)$: due to this characteristic the Tabu Method facilitates to avoid the trapping of *f* in the local minima.

However, simply adopting the above described procedure does not ensure overcoming limitations of classical heuristic algorithms, as there is a significant risk to return to a solution already retained at the time of a preceding iteration: this can generate a cycle.

To avoid this phenomenon, the Tabu Method requires updating exploiting a list of prohibited moves, the so-called *Tabu List*. This list — that gave its name to the method — contains *m* moves ($T \rightarrow S$), which are the opposite of the last *m* moves ($S \rightarrow T$) carried out.

The algorithm models a rudimentary form of memory, the *short term memory* of the solutions visited recently.

Two additional mechanisms, named *intensification* and *diversification*, are often implemented to also equip the algorithm with a *long term memory*. This process does not exploit more the temporal proximity of particular events, but rather the frequency of their occurrence, over a longer period. The intensification consists in looking further into the exploration of certain areas of the solution space, identified as particularly promising ones. On the contrary, diversification is the periodic reorientation of the search for an optimum towards areas, seldom visited until now.

For certain optimization problems, the Tabu Method gave excellent results; moreover, in its basic form, the method comprises less parameters of adjustment than Simulated Annealing, which makes it easier to use.

However, the various additional mechanisms, like the intensification and diversification, bring a notable complexity. A synthetic schema of the algorithm is illustrated in Figure 2.7.

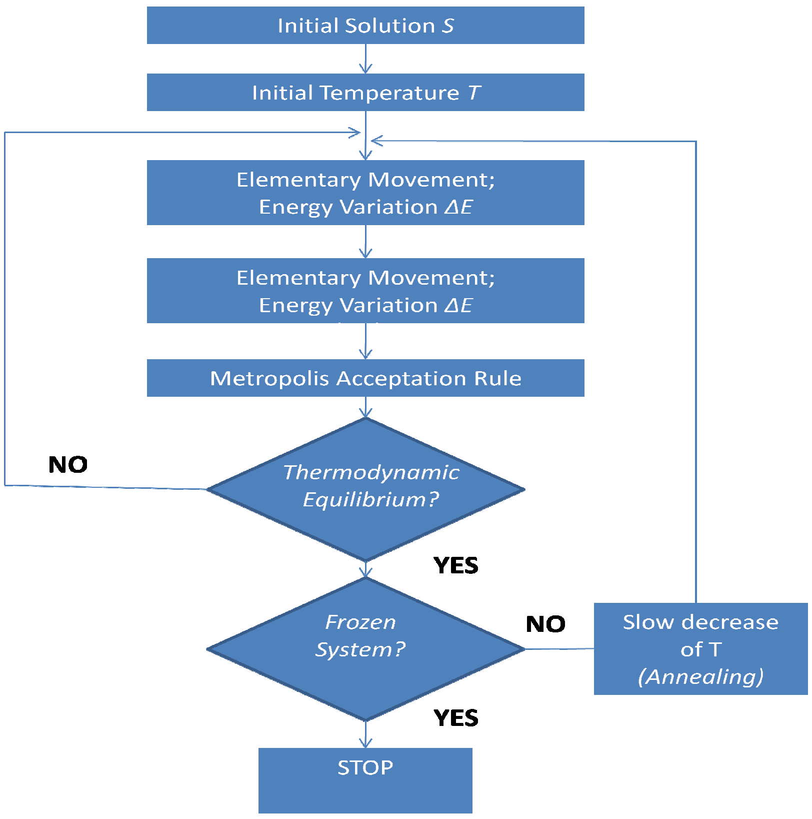

Initial Solution *S*
Initial Temperature *T*
Elementary Movement; Energy Variation *ΔE*
Elementary Movement; Energy Variation *ΔE*
Metropolis Acceptation Rule
NO
Thermodynamic Equilibrium?
YES
Frozen System?
NO
Slow decrease of T *(Annealing)*
YES
STOP

Figure 2.6 - Schema of a Simulated Annealing algorithm

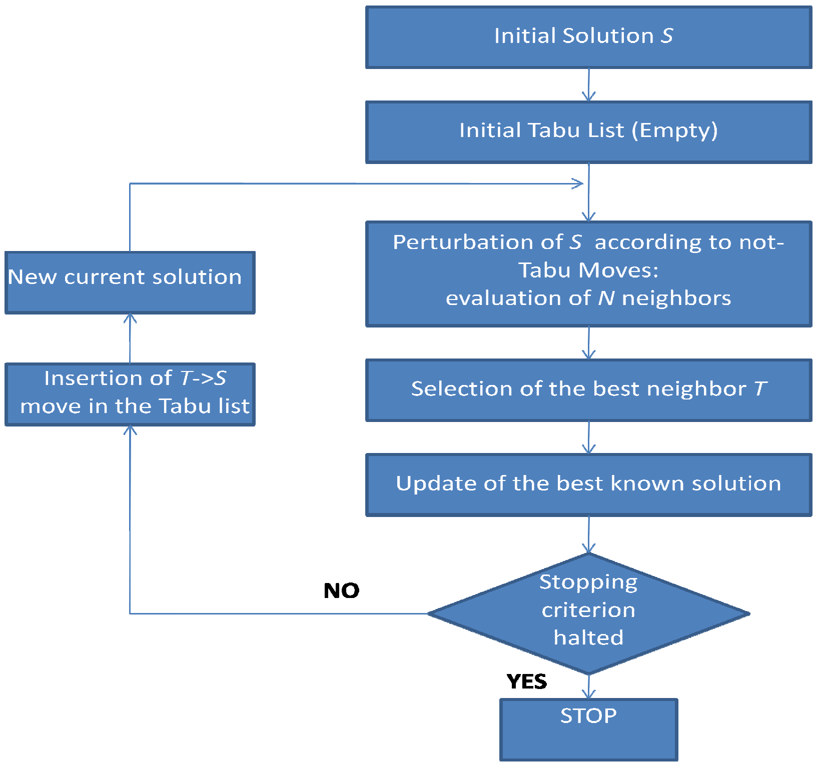

Initial Solution *S*
Initial Tabu List (Empty)
Perturbation of *S* according to not-Tabu Moves: evaluation of *N* neighbors
New current solution
Insertion of *T->S* move in the Tabu list
Selection of the best neighbor *T*
Update of the best known solution
NO
Stopping criterion halted
YES
STOP

Figure 2.7 - Schema of a Tabu Search algorithm

2.6.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are the search techniques inspired by the biological evolution of the species and appeared at the end of the 1950s (Fraser, 1957). Among several approaches (Holland, 1962; Fogel et al., 1966; Rechenberg, 1965), the genetic algorithms (GAs) are certainly the most well known example (Goldberg, 1989).

The evolutionary methods initially aroused a limited interest, because of their significant cost of execution. But they have experienced a considerable development, that can be attributed to the significant increase in the computing power of the computers.

The principle of an evolutionary algorithm can be simply described. A set of N points in a search space, chosen a priori at random, constitutes the *initial population*; each individual x of the population has a certain fitness value, which measures its degree of *adaptation* to the objective aimed. In the case of the minimization of an objective function z, the fitness of x will be higher, if $z(x)$ is smaller. An EA consists in evolving gradually, in successive *generations*, the composition of the population, by maintaining its size constant.

During generations, the objective is to overall improve the fitness of the individuals; such a result is obtained by simulating the two principal mechanisms which govern the evolution of the living beings, according to the theory of C. Darwin:

- *selection*, which supports the reproduction and the survival of the fittest individuals;
- *reproduction*, which allows mixing, the recombination and the variations of the hereditary features of the parents, to form offspring with new potentialities.

In practice, a representation must be chosen for the individuals of a population. Classically, an individual could be a list of integers for combinatorial problems, a vector of real numbers for numerical problems in continuous spaces, a string of binary digits for Boolean problems, or will be able to even combine these representations in complex structures, if it is required.

The passage from one generation to the next one proceeds in four phases: a phase of selection, a phase of reproduction (or variation), a phase of fitness evaluation and a phase of replacement. The selection phase designates the individuals who take part in the reproduction. They are chosen, possibly several times, a priori all the more often as they have high fitness. The selected individuals are then available for the reproduction phase. This one consists in applying variation operators to copies of the individuals previously selected to generate new individuals; the operators most often used are *crossover* (or *recombination*), which produces one or two offspring from two parents, and *mutation*, which produces a new individual from only one individual. The structure of the variation operators depends largely on the chosen representation for the individuals. The fitness of the new individuals is then evaluated, during the evaluation phase, from the objectives specified.

Lastly, the replacement phase consists in selecting the members of the new generations: one can, for example, replace the lowest fitness individuals of the population by the best produced individuals, in an equal number. The algorithm is terminated after a certain number of generations, according to a termination criterion arbitrarily specified by the user.

Because they handle a population of solution instances, the evolutionary algorithms are particularly indicated to propose a set of various solutions, when an objective function comprises several global optima. Thus, they can provide a sample of trade-off solutions, when solving problems involving several objectives, possibly contradictory. A general schema for an evolutionary algorithm is shown in Figure 2.8; Figure 2.9 shows the functioning of a cross-over operator for a Genetic Algorithm.

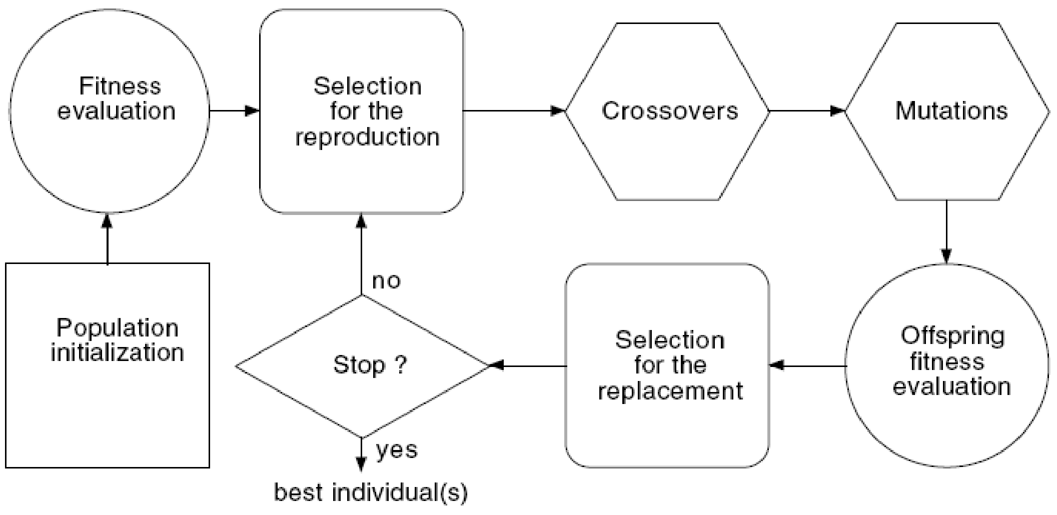

Fitness evaluation
Selection for the reproduction
Crossovers
Mutations
Population initialization
no
Stop ?
yes
best individual(s)
Selection for the replacement
Offspring fitness evaluation

Figure 2.8 - Schema of an Evolutionary Algorithm (Siarry and Taillard, 2006)

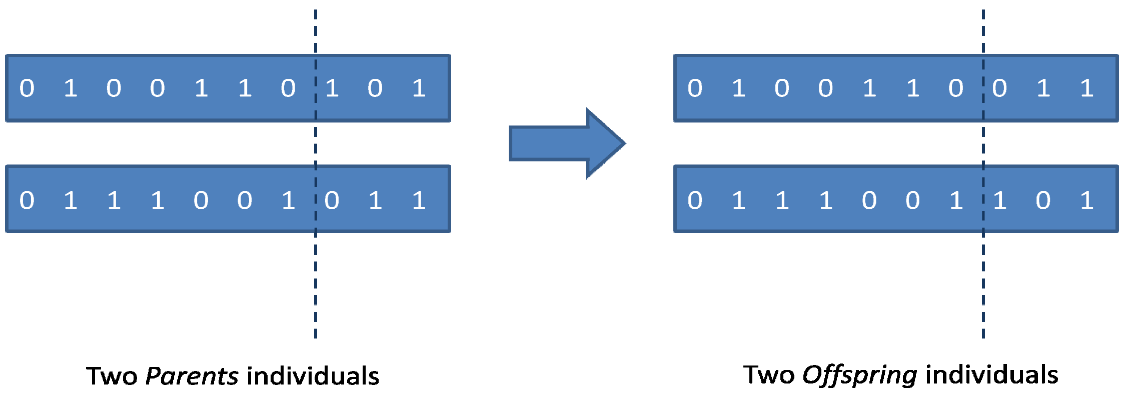


Figure 2.9 - Schema of a Crossover operator for a Genetic Algorithm

2.6.4 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a recently developed metaheuristic which exploits systematically the idea of neighborhood change as a way to escape from valleys that contain local minima. This metaheuristic exploits the following facts (Mladenovic and Hansen, 1997):

- A local minimum with respect to one neighborhood structure, is not necessarily so for another;
- A global minimum is a local minimum with respect to all possible neighborhood structures;
- For many problems local minima with respect to one or several neighborhoods are relatively close each other.

The last empirical observation implies that often the knowledge of a local optimum can provide some information about the global one.

The basic idea of VNS is to select a series of neighborhood structures $\mathcal{N}_k$ (with $k=1,\ldots,k_{max}$), which define neighborhoods around any point of the feasible domain. The first step of the algorithm is to implement a local search procedure that leads to a local optimum x. A point x' is selected at random (thanks to a so-called *shaking* procedure) within the first neighborhood $\mathcal{N}_1(x)$ of x and a descent from x' is done with the local search routine this leads to a new local minimum x''. Then three results are possible:

- $x''= x$, i.e. one is again at the bottom of the same valley; in this case, the procedure is iterated considering a new neighborhood $\mathcal{N}_k$, $k\geq 2$;
- $x''\neq x$, but $f(x'')\geq f(x)$, i.e. another local optimum has been found, but its objective function value is not better the one associated with the current solution; in this case, the procedure is iterated using the next neighborhood;
- $x''\neq x$ and $f(x'')<f(x)$, i.e. a local optimum better than the current solution is found; in this case, the search procedure is re-centered around x'' considering the first neighborhood structure $\mathcal{N}_1(x'')$.

The procedure runs until a stopping condition, e.g. a maximum number of iterations or a maximum number of iterations since the last improvement, is satisfied.

2.6.5 Swarm Intelligence

Swarm Intelligence is a field of computer science that develops methods for solving complex computational problems inspired by behavior of real swarms or insect colonies (Kennedy et al., 2001). This class of algorithms starts from simple principles of self-organization and

communication observed in the real world of natural swarms; from these principles, it is possible to gain insights that can be utilized to understand complex collective behaviors and implemented in the design of algorithms and systems.
In the last decades, two main swarm intelligence methods for solving optimization problems have been widely developed and utilized in the OR community: the Ant Colony Optimization (ACO), mainly employed to deal with combinatorial optimization problems, and the Particle Swarm Optimization (PSO), simply applicable to continuous optimization problems.
In the following, the details of these two techniques are briefly analyzed.

Ant Colony Optimization

ACO is a metaheuristic for solving combinatorial optimization problems. It takes inspiration from the way ants find shortest paths from their nest to food. The algorithm is essentially based on the indirect communication of the ants thanks to the pheromone, a chemical compound released by ants in the environment and perceived by individuals belonging to the specie. Thus, ants mark their paths to the food sources, and these traces can be followed by other ants looking for food.

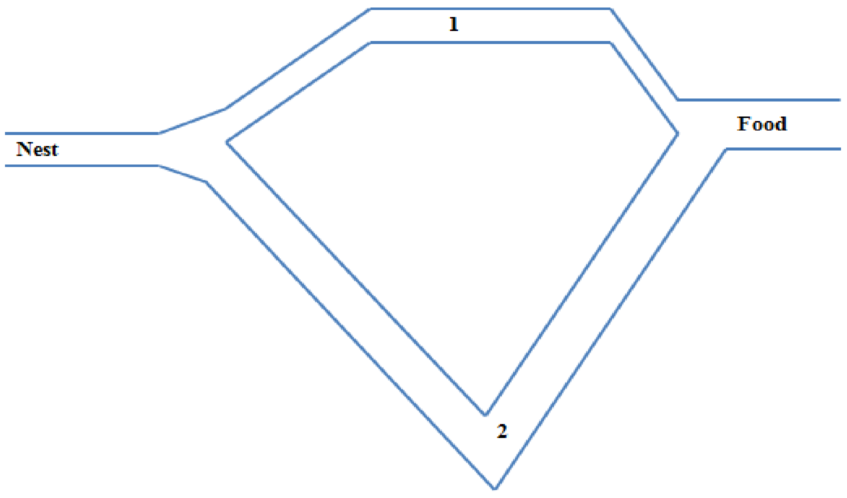


Figure 2.10 - The Double Bridge experiment

This evidence was confirmed by the so-called Double Bridge experiment (Deneubourg et al., 1990) (Figure 2.10). In the experiment, two paths are available to ants to reach the food sources starting from their nest. The longest path is twice as long as the shortest one. Interestingly, it was noticed that, after a few minutes, almost all the ants use the shortest path. This is due to the largest pheromone concentration on the shortest path, as ants choosing it are able to go back to the nest earlier. Obviously, if the two paths have the same length, ants will chose randomly their way to the food source.

Inspired by this experiment, Dorigo et al. (1991) designed an algorithm for the Traveling Salesperson Problem (TSP) and provided the foundations of the Ant Colony Optimization field. The idea is to have a colony of artificial ants and let them construct solutions for a combinatorial optimization problem. In practice, each ant is endowed with a greedy algorithm that allows building a solution through a sequence of decision. The sequence of decisions for constructing a solution can be seen as a decision graph; thus, ants are "walking" through the decision graph looking for good solutions. Doing this, artificial ants are endowed with communication mechanisms similar to the ones of real ants. Ants that found good solutions are allowed to mark the edges of the corresponding path with artificial pheromone that guides ants in the following iterations in the search process, similarly to an intensification mechanism. In order to implement something similar to diversification mechanisms for avoiding the search process being concentrated just on a part of the feasible domain, pheromone traces tend to slightly "evaporate" over the time. The process continues until some stopping criterion (a maximum number of iterations, a good quality solution) is met.

A summary of an ACO procedure can be stated as shown in Figure 2.11.

```
ACO scheme:
        Initialize pheromone values
        repeat
                for ant $k \in \{1,\dots, m\}$
                    construct a solution
                endfor
                forall pheromone values do
                    decrease the value by a certain percentage {evaporation}
                endfor
                forall pheromone values corresponding to good solutions do
                    increase the value by a certain percentage {intensification}
                endfor
        until stopping criterion is met
```

Figure 2.11 - Ant Colony Optimization schema

Nowadays, ACO algorithms have been designed for various combinatorial optimization problems, including dynamic and multi-objective problems (see , for instance, Cordon et al., 2002; Maniezzo et al., 2001) performing good computational results.

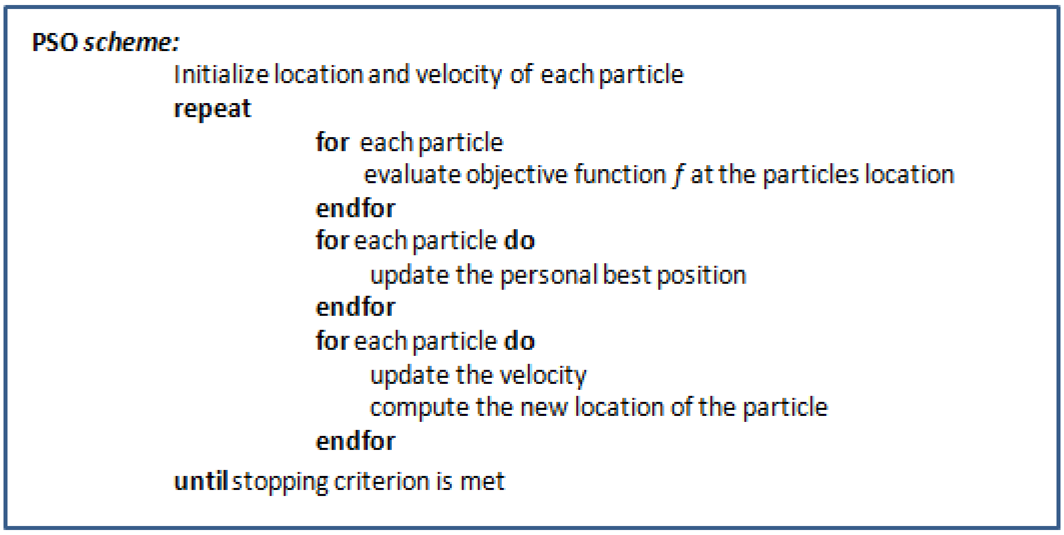
Particle Swarm Optimization

The Particle Swarm Optimization is based on the coordinated food search mechanism exhibited by swarm of birds. In a PSO algorithm the search process is modeled through a population of particles (the swarm) in a multidimensional search space (also called problem space).

Particles start from random locations with a certain velocity and look for improvements in a given objective function by moving through the search space, similarly to the process of food search in the reality.

In a typical PSO algorithm, each particle keeps track of the coordinates in the search space which are associated with the best solution it has found till so far. At a centralized level, the best solution found by all the particles is stored as well.

A summary of an ACO procedure can be stated as shown in Figure 2.12.



```
PSO scheme:
        Initialize location and velocity of each particle
        repeat
                for each particle
                    evaluate objective function $f$ at the particles location
                endfor
                for each particle do
                    update the personal best position
                endfor
                for each particle do
                    update the velocity
                    compute the new location of the particle
                endfor
        until stopping criterion is met
```

Figure 2.12: Particle Swarm Optimization schema

2.7 Some Considerations

In the presence of a concrete optimization problem, the principal difficulty with which an engineer is confronted, is the choice of an efficient method, able to produce an optimal solution (or of acceptable quality) at the cost of a “reasonable” computing time.

Compared to this pragmatic concern of the user, the theory is not yet of a great help, because the convergence theorems are often non-existent, or applicable under very restrictive assumptions. Moreover, the optimal adjustment of the various parameters of a metaheuristic which can be recommended theoretically is often inapplicable in practice, because it induces a prohibitive computing cost.

Consequently, the choice of a good method, and the adjustment of the parameters of this one, generally calls upon the know-how and the “experience” of the user, rather than the faithful application of well laid down rules.
In the last decade a new direction of research has emerged in the field of *hyper-heuristics*. The key idea is to devise new algorithms for solving problems by combining known heuristics in ways that allow each to compensate, to some extent, for the weaknesses of others (Ross, 2005). Hyper-heuristics can be thought as heuristics to choose the right heuristic for a given problem. Differently from heuristic algorithms (that work on a *solution space*) hyper-heuristics work with a search space of heuristics. The key intuition underlying them is that often there are a number of available straightforward heuristics that can work well for certain sort of instances for a given problem; it could be possible that, combining those algorithms in a certain way, an algorithm that will work well across a broader range of instances can be obtained. Further references can be obtained in Ross (2005).

Chapter 3
Agent-based Approaches for Optimization Problems

3.1 Introduction

After having described the agent paradigm and some generalities about multi-agent systems in Chapter 1 and illustrated an overview of optimization methods in Chapter 2, this chapter will be devoted to the illustration of the application of MASs to optimization problems. Indeed, in the last decade, growing attention has been addressed towards the development of methodologies based on MASs to model and solve classical Operational Research problems.
In this chapter, a first comparison among MASs-based approaches and classical optimization techniques will be provided, followed by an extensive review aimed at evaluating the impact of these methodologies in the Operational Research/Management Science (OR/MS) literature.

3.2 Agent-based Approaches vs Classical Approaches

For a long time, classical optimization techniques have represented the only available approach to solve different types of decision-making problem, both at strategic and tactical levels.
In the last decade, agent-based computing has been suggested as a promising technique for problem whose domains are distributed, complex and heterogeneous (Weiss, 1999; Wooldridge, 2002).
Parunak (1999) proposed a first formalization of a set of resources allocation problems using MASs.
Davidsson et al. (2007) proposed a theoretical framework for the comparison of the two approaches. The framework is based on a series of dimensions useful to classify decision-making problems; on the basis of these dimensions, the authors compare characteristics of agent-based and classical approaches, determining situation in which one methodology is preferred over the other one, with particular reference to a special class of resource allocation problems, namely dynamic distributed resource allocation. An adaptation of these considered parameters can be listed as reported in Table 3.1. Table 3.2 provides desired properties for solution methods according to each dimension.

Dimension	Description
Size	Number of Decision Variables, Parameters, Constraints
Modularity	Possibility of clearly identifying sub-domains and sub-problems
Time Scale/Changeability	How often the structure of the domain changes
Solution Quality	How important it is to find the optimal or near optimal solution
Computational Complexity	Number of operations required to solve the problem

Table 3.1 - Optimization problems classification framework

Dimension	Desired Properties of Solution Method
Size	Low computational complexity
Modularity	Support for modular decomposition
Time Scale/Changeability	High reactivity and modifiability, short response time
Solution Quality	Ability to find optimal or near optimal solutions
Computational Complexity	Low number of operations, short computational times

Table 3.2 - Desired Properties of Solution Methods according to each dimension

Comparing the two approaches (MASs-based and classic optimization) according to *size*, since agent-based approaches support the dividing of the global problem into a number of smaller local allocation problems, large-sized problems could be handled well in such cases the problem is modular. On the other hand, the complexity and the size of the problem may affect the solution time dramatically when applying an optimization method. Since optimization techniques attempt to achieve global optimality, capitalizing on partial modularity in order to handle large-sized problems is difficult.

Concerning *modularity*, as agent-based approaches are modular by nature they are very suitable for highly modular domains. However, if the modularity of the domain is low they may be very difficult to apply.

Moreover, since agents are able to continuously monitor the state of its local environment and typically do not have to make very complex decisions, they are able to react to changes fast, providing some advantages if the domain of the problem is characterized by a high *changeability/time scale* level. On the other hand, optimization techniques often require a relatively long time to respond to changes in variables and parameters of the problem, as they often need a complete restart. Hence a rather high degree of predictability is required for optimization methods to work efficiently if a short response time is required. Sometimes methods of re-optimization can be used for lowering the response time.

From the point of view of the *quality of solution,* since agent-based approaches are distributed, they do not have a global view of the state of the system, which unfortunately often is necessary in order to find a truly good solution. Therefore, the quality of the solution suggested by an optimization method often will be of a higher quality. Moreover, it may be very difficult (and sometimes even impossible) to estimate the quality of the solution provided by an agent-based approach, as it can be difficult to retrieve reference values (i.e., a bound of the optimal solution values).

In terms of computational times, agent-based approaches can provide some advantages thanks to their ability to divide problems in several sub-problems; however, computational advantages can be offset by the need for frequent interaction in order to coordinate activities and decisions among agents; centralized approach present higher computational complexity, but no communication costs, as they are characterized by high centralization.

According to this comparison, agent-based approaches tend to be preferable when:

- the size of the problem is large;
- the domain is modular in nature;
- the structure of the domain changes frequently (i.e., high changeability).

Classical optimization techniques may outperform MASs-based ones when:

- decomposing the problem in sub-problems can be costly in terms of computational requirements and times;
- the domain is monolithic in nature;
- the quality of the solution is very important.

This analysis indicates that agent-based approaches and classical optimization techniques complement each other. This can explain the increasing interests towards approaches embedding optimization techniques within a MAS schema. There are several ways to integrate MASs-based approaches and classical optimization techniques. The most relevant seems to be:

- Utilizing an optimization technique for strategic planning and MASs for operational and tactic re-planning, i.e., for performing local adjustments of the initial plan;
- Embedding optimization in an agent, by translating search algorithms in agents' behavior.

The reminder of the chapter will be devoted to verify the impact of MASs based methodologies to cope with optimization problems in the OR/MS literature. To this aim, an extensive survey of the state-of-the-art will be provided.

3.3 A Literature Review

To the aim of verifying the presence of a real interest in the literature about MASs-based techniques for optimization problems, an extensive State-of-the-Art survey has been performed.

Through the web-based tool *Google Scholar* (including the most widespread academic search engines), international referred journals in the time interval 2000-2008 have been scrutinized, looking for the words *agent-based optimization* within title, key-words and abstract of the papers.

As it emerges from Table 3.3, 49 papers have been retrieved. From Figure 3.1, it's possible to notice the outstanding increase in publications devoted to the topic.

Year	**2000**	**2001**	**2002**	**2003**	**2004**	**2005**	**2006**	**2007**	**2008**	**Totale Papers**
Papers	3	3	1	4	5	5	7	13	8	**49**

Table 3.3 - Historical series of MASs-based optimization papers

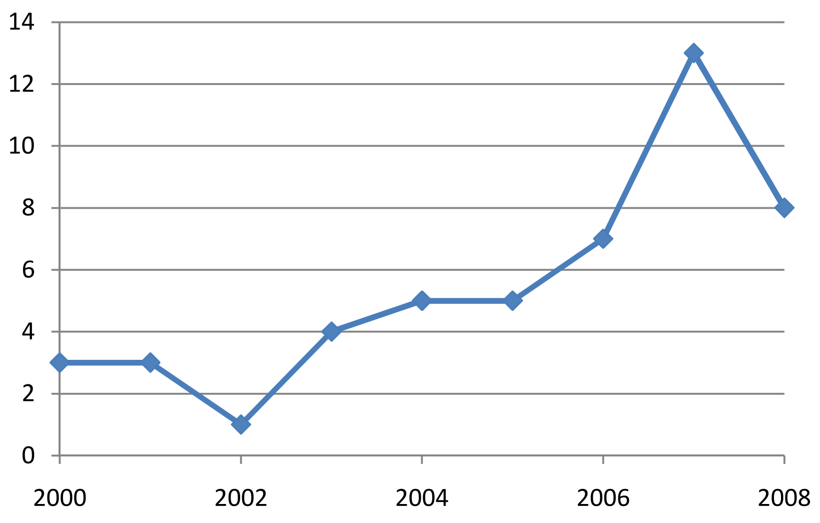


Figure 3.1 - Historical series of MASs-based optimization papers

As regards journals, 30 publications reported at least a paper. Table 3.4 reports journals including at least two papers. They account for 27 total papers out of 51 (53.94% of the total number of papers). The top contributor is the journal *Engineering Applications of Artificial Intelligence*, as it has been explained as MASs fall in the field of Artificial Intelligence.

However, it is noticeable that one of the most prominent journal in OR/MS field, *European Journal of Operational Research*, has hosted 7 contributions. This testifies that MASs approaches for optimization problems are becoming an accepted tool in the OR/MS community.

Journal	**Papers**
Engineering Applications of Artificial Intelligence	**8**
European Journal of Operational Research	**7**
IEEE Transactions on Systems, Men and Cybernetics	**5**
Robotics and Computer-Integrated Manufacturing	**3**
International Journal of Advanced Manufacturing Technology	**2**
International Journal of Production Economics	**2**

Table 3.4 - Journals accounting for at least two papers

The analysis of the journals provides valuable insights: it emerges that MASs applications to optimization problems is a multidisciplinary field of study, as papers on the topic have been retrieved on journals belonging to different disciplinary areas: the above cited Artificial Intelligence and OR/MS, but also Manufacturing (*Robotics and Computer-Integrated Manufacturing*, *International Journal of Advanced Manufacturing Technology*), Logistics (*International Journal of Production Economics*) and others.
Another interesting perspective is offered by the geographical analysis of the papers. Table 3.5 classifies papers according to the country where the institution of the first author is based. The top contributor is China (6 papers), followed by Canada (5).

Country	**Papers**
China	6
Canada	5
Japan	4
UK	4
France	4
Germany	4
USA	4
Taiwan	4
Spain	2
India	2
Singapore	2
Netherlands	2

Table 3.5 - Papers classified by countries of origin

Table 3.6 reports key-words retrieved in the surveyed papers, and the number of occurrences for each key-word. It emerges that the words multi-agent system, agent, and agent-based system, are the most cited. Very often key-words underline the application field of the developed MASs: for example, several papers refer to Supply Chain Management, other to transportation problems and to scheduling applications.
The occurrence of the key-word Ant Colony Optimization can be explained as this

technique can be considered as an extension of the MASs paradigm, embedding optimizing rules in the definition of agents' behaviors. The presence of other meta-heuristics (Tabu Search, Simulated Annealing, Genetic Algorithms) testifies the possibility of integrating search algorithms and MASs.

A further classification is provided in Table 3.7, based on the application field of the papers. MASs approaches seem to be particularly suitable to tackle scheduling and Supply Chain problems.

KeyWord	**Frequency**
Multi-agent system	20
Supply chain management	6
Agents	6
Ant colony optimization	4
Manufacturing scheduling	3
Agent-based systems	3
Scheduling	3
Heuristics	3
Optimization	2
Mobile agent	2
Holonic manufacturing systems	2
Negotiation	2
Integrated process planning and scheduling	2
Intelligent manufacturing	2
Dynamic scheduling	2
Distributed scheduling	2
Artificial intelligence	2
Transportation	2
Simulation	2

Table 3.6 - Keywords

Application Field	Papers
Scheduling	23
Supply Chain	9
Routing	4
Manufacturing	4
Logistics	3
Location	2
Transportation	2
Industrial Planning	1

Table 3.7 - MASs-based approaches application fields

3.3.1 Agent-Based scheduling approaches

It can be useful to focus the attention on MASs-based approaches for scheduling problems, as it appears that this class of problems is particularly suitable to be tackled with MAS-based approaches. Extracting from the previously described sample all the papers published about scheduling problems, the resulting historical series testifies the growing interest towards this application field.

Year	2000	2001	2002	2003	2004	2005	2006	2007	2008	Totale Papers
Papers	1	2	1	2	2	2	3	7	3	**23**

Table 3.8: Historical series of MASs-based scheduling papers

Table 3.9 reports major journal contributors, while table 3.10 classifies papers on the basis of institution nationality.

Journal	Papers
Engineering Applications of Artificial Intelligence	6
IEEE Transactions on Systems, Men and Cybernetics	4
European Journal of Operational Research	4
International Journal of advanced manufacturing technology	2

Table 3.9 - Journals with at least two papers about MASs-based scheduling approaches

Paese	Papers
China	3
USA	3
Canada	3
Japan	2
Germany	2
Taiwan	2
Singapore	2

Table 3.10 - MASs-based scheduling papers classified by countries of origin

As highlighted by Ouelhadj (2008) and shown in the previous section multi-agent systems have found wide application to address complex and dynamic environments related to scheduling problems.

Analyzing the literature, two main multi-agent architectures for dynamic scheduling have been utilized: autonomous architectures and mediator architectures. They are described in more detail in the following sub-sections. For further references, the reader can also refer to Shen et al. (2006).

Autonomous architectures

In autonomous architectures (Figure 3.2), agents representing manufacturing entities such as resources and jobs have the ability to generate their local schedules, react locally to local changes, and cooperate directly with each other to generate global optimal and robust schedules (Ouelhadj, 2008).

One of the earliest MAS-based scheduling architecture was proposed by Parunak (1987) with YAMS (Yet Another Manufacturing System) platform. This system assigns an agent to each node in a control hierarchy (factory, cell, workstation, machine, jobs). The main idea of Yams is that the job agents negotiate with resource agents to assign tasks to the machine agents using the contract net protocol (Smith, 1980). This idea was then utilized also by Shaw (1988) in order to develop a dynamic scheduling system in a cellular manufacturing system. Request for bid messages are broadcast to cells which evaluate operations specification and submit bids. Bids describe the estimation on the earliest finishing time or shortest processing time of the operations. The cell that optimizes a predefined criterion is selected to perform the operation.

Goldsmith and Interrante (1998), and Ouelhadj et al. (1998, 1999, 2000) proposed simple multi-agent architectures for dynamic scheduling in flexible manufacturing systems which involves only resource agents. The resource agents are responsible for dynamic scheduling of the operations and they have no control over each other. They negotiate using the contract net protocol to produce a global schedule. Each resource agent performs the following functions: scheduling, detection, diagnosis, and error handling. Resource agents are also able to react to real-time events (such as machine breakdown), by renegotiating processes.

Sousa and Ramos (1999) proposed some advances by utilizing a MAS-based architecture that involved also jobs agents. The contract net protocol is utilized to model requests coming from jobs to be processed directed to the machines. Resource agents are also able to send fault messages to job agents to signal temporary unavailability.

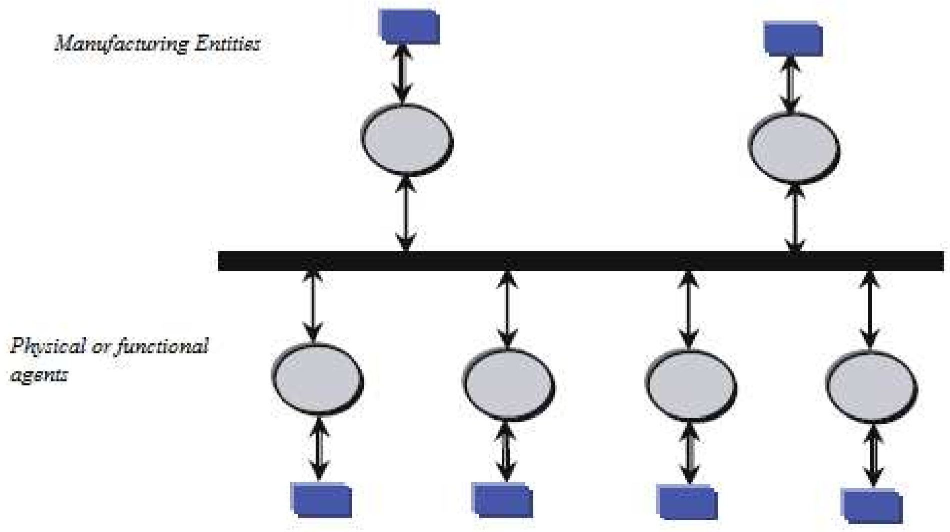


Figure 3.2 - Autonomous MAS-based scheduling architecture (Ouelhadj, 2008)

Cowling et al. (2003, 2004) and Ouelhadj et al. (2003) proposed an adaptation of MAS-based scheduling to dynamic scheduling in steel production process, in which agents representing particular stages of the steel production process are introduced as a further level in the architecture. Several heuristics are implemented in order to obtain robust schedules in the presence of real-time events.

Sandholm (2000) proposed an extension of the contract net protocol in order to deal with negotiation in presence of imprecise information. This extensions allows agents to de-commit from previously negotiated contacts in order to deal with real-time events by simply paying a de-commitment penalty.

This architecture can be traced to Lin and Solberg (1992) that proposed an autonomous multi-agent architecture for shop floor dynamic scheduling based on a currency model that combined the scheduling objectives and price mechanism. Their model represents jobs, resources, and parts by agents. Job agents negotiate with resource agents via a contract net bidding mechanism to optimize a weighted objective that is a function of due date, price, quality, and other user defined factors. The part agent enters the system with a certain currency, and solicits and evaluates bids from several resource agents capable of fulfilling the processing requirements, and selects the one that optimizes its objective. Each resource agent sets its charging price based on its status, then it decides on the basis of the currency offered which of the announced jobs to consider more interesting for a possible bid. The job agent tries to minimize the price paid, but the resource agent's goal is to maximize the price charged. Each deal is completed once the job and resource agents are mutually committed. When a resource agent is in failure, it informs the corresponding job agent, and the latter proceeds to a renegotiation process on the operations in failure with the resource agents.

Other multi-agent based dynamic scheduling systems use learning approaches for dynamic scheduling. Aydin and Öztemel (2000) proposed a dynamic job shop scheduling using reinforcement learning agents. The agent is trained by an improved reinforcement-learning algorithm through the learning stage and then successfully makes the decisions to schedule the operations. The scheduling system consists of two parts: the simulator and the intelligent agent. The agent selects the most appropriate priority rule to select a job to assign to a machine according to the shop conditions, while the simulator performs scheduling activities using the rule selected by the agent. Pendharkar (1999) proposed a multi-agent learning approach for dynamic scheduling. In the multi-agent architecture, the work areas are controlled by agents with a knowledge base containing the dispatching rules. The agents use genetic algorithms-based learning to update the rules in the knowledge-base at periodic intervals of time. The higher frequency of learning may help an agent to quickly adapt to variations on the shop floor.

Knotts et al. (2000) illustrate eight agent-based algorithms for solving the multimode, resource-constrained project scheduling problem, comparing their implementation and their results utilizing purely reactive agents or agents with state.

Chun et al. (2003) developed a architecture for meeting scheduling through performance estimation, by implementing a MAS in which two categories of agents (*meeting* agents, representing forthcoming appointments and *secretary* agents, knowing the schedule and the preferences of the potential participants) are involved and negotiate each other to find a schedule that maximizes a given performance function.

Frey et al. (2003) developed a MAS for Job-Shop problems and compared it to traditional heuristic algorithms using a benchmarking scenario, proving MASs' superiority in a turbulent production environment.
Chen and Wang (2007a, 2007b) proposed a model that concentrates on solving the dynamic scheduling problem of a distributed project for non-cooperative and self-interested participants. In this model, the self-interested activity agents possess various negotiation tactics and strategies. In order to find fitting negotiation tactics and strategies that are optimally adapted for each activity agent, an evolutionary computation approach which encodes the parameters of tactics and strategies of an agent as genes in GAs is also utilized.
Shukla et al. (2006) and Liu et al. (2007) proposed an auction-based MAS in which, like in other approaches, each job and each machine are represented by agent. Each machine agent is also an auctioneer and each job agent is a bidder. Machine agents host combinatorial auctions with proposed prices each time slot of all the machines; bidders construct their bids for the wanted time slots. Coordination of bidders and auctioneer is achieved through an iterative adjustment of prices.
Mes et al. (2007) adapted the auction-based MAS schema to some scheduling problems in transportation contexts, comparing the results obtained thanks to this approach to the ones generated by traditional heuristics.
Leitao and Restivo (2008) proposed a MAS-based holonic approach to manufacturing scheduling, where the scheduling functions are distributed by several entities oriented to fast and dynamic re-scheduling using a scheduling mechanism that evolves dynamically to combine centralized and distributed strategies, improving its responsiveness to emergence.

Mediator architectures
Despite the good performance of autonomous architectures, they usually face problems in providing globally optimized schedules and predictability in the presence of a large number of agents, as also highlighted by Davidsson et al. (2007).
Thus, several approaches based on mediator architectures have been proposed in order to deal with complex scheduling systems aimed at combining robustness, optimality, and predictability of the proposed solution.
The mediator architecture provides computational simplicity, while being quite suitable for developing industrial applications based on MASs.

Brennan and Norrie (2001), Bongaerts et al. (2000), and Cavalieri et al. (2000) showed that mediator MASs architectures allow obtaining significant performance improvements if compared to autonomous architectures.

A mediator-based architecture is composed by a basic structure consisting of autonomous cooperating local agents that are capable to negotiate with each other in order to achieve production targets (Bongaerts et al. 2000; Shen et al. 2001), extended with mediator agents to coordinate the behavior of the local agents to perform global dynamic scheduling (Figure 3.3).

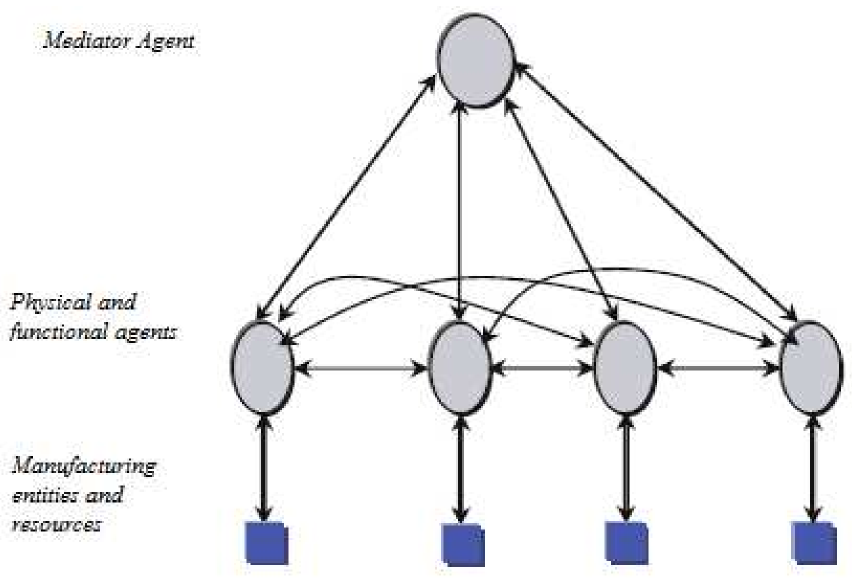

Mediator Agent
Physical and functional agents
Manufacturing entities and resources

Figure 3.3 - Mediator MAS-based scheduling architecture (Ouelhadj, 2008)

The architecture is such that the local agents maintain their autonomous decision making process, but may request advice from the mediator agents. They have the ability to advice, impose or update decisions taken by the resource agents in order to satisfy the global objectives and resolve potential conflict situations. The mediator agent has an overview of the entire system, while the local agents can have a more detailed and up-to-date view of the local situations.

The first basic mediator architecture was proposed by Ramos (1994), composed of task agents, task manager agent, resource agents, and resource mediator agent. Task manager agent creates the task agents. The resource mediator agent negotiates with the resource agents the execution the tasks using the contract net protocol.

For an increased robustness in complex manufacturing systems, some authors proposed the integration of mediator agents to each level of the manufacturing facility (Shen et al., 2000), developing hierarchies of sub-system mediators each responsible for coordinating a part of the manufacturing system.

For instance, Sun and Xue (2001) develope a mediator reactive scheduling architecture for responding to changes in jobs and manufacturing resources. Manufacturing resources including facilities and resources are represented by agents that are coordinated by two mediators, namely a facility mediator and a personnel mediator, using the contract net protocol. Reactive scheduling is conducted to modify the created schedule to respond to changes of jobs such as cancellation of jobs or insertion of urgent jobs, and manufacturing conditions such as machine breakdowns, or a person's sudden sickness during the production process. Match up rescheduling strategy and agent-based collaboration are used to repair only part of the originally created schedule for improving the reactive scheduling efficiency, while maintaining the scheduling quality. A similar approach is also illustrated by Lim and Zhang (2004).

Archimede and Coudert (2001) develop a multi-agent framework, based on four agents categories (Supervisor, Customers, Environment, Producers) with the aim of reaching a high level of co-operation. Its two main interests are the following: first it provides a more efficient control of the consequences generated by the local decisions than usual systems to each agent, then the adopted architecture and behaviour permit an easy co-operation between the different scheduling systems, which can represent different production functions. The MAS-based framework can be adapted to a great variety of scheduling/planning problems.

Zhou et al. (2004) apply a mediator-based MAS architecture to a real-life bus scheduling problem.

Through an extension of the contract net protocol, Wong et al. (2006) establish a comparison between autonomous and mediator architectures.

BenHassine and Ho (2007) propose an extension to Meeting Scheduling problem, in which two categories of agents are introduced: user agents (proposer or participant in a meeting) and interface, that is responsible for the global optimization of the objective function.

Homberger (2007) presents a restart evolution strategy (RES) for the resource-constrained project scheduling problem (RCPSP) integrated in a mediator-based MAS. The approach is tested on problem instances for the RCPSP problem taken from the literature; in particular, it manages to found better solutions than the best ones found so far.

3.3.2 Agent-Based Supply Chain planning approaches

In order to address the planning of manufacturing and supply chain systems, academics have initiated in the middle of the 1980s a new body of approaches and distributed computing techniques drifting away from traditional OR-based solutions. Some of these approaches utilize MAS-based techniques in order to achieve reactive, reliable, and (re)configurable operation management systems.

An agent-based manufacturing system may be defined as a planning and control system made of interdependent software agents designed to:

- individually handle a part of a manufacturing planning and control problem, such as planning a single order or allocating tasks to resources;
- collectively carry out specific higher functionalities such as planning an entire manufacturing system.

For instance, Karageorgos et al. (2003) illustrated the suitability of MASs in a case study concerning optimization of production planning of a virtual manufacturing enterprise in relation to sub-contracted logistic services used to transport materials between the enterprise units.

Caridi and Cavalieri (2004) provided a critical analysis of MAS-based approaches applied to Supply Chain Management, pointing out the lack of real world applications and the low maturity level of agent-based manufacturing technology.

3.3.3 Agent-Based routing approaches

Barbucha and Jedrzejowicz (2007) proposed one of the first MAS-based routing approach through the development of a population based algorithm.

The approach produces solutions to routing combinatorial optimization problems using a set of agents, each representing an improvement algorithm. To escape getting trapped into a local optimum an initial population of solutions called individuals is generated or constructed. Individuals forming an initial population are, at the following computation stages, improved by independently acting agents, thus increasing chances for reaching a global optimum. The steps of the procedure:

- Generating an initial population of solutions;
- Applying solution improvement algorithms which draw individuals from the common memory and store them back after attempted improvement, using some user defined replacement strategy;

- Continuing reading-improving-replacing cycle until a stopping criterion is met.

This functionality is realized mainly by two types of agents:

- *OptiAgents – OA*, optimizing agents (*OptiAgents*), each representing a single optimizing algorithm;
- *SolutionManagers – SMa* , responsible for finding the best solution of a single instance of the problem and maintains a single population of solutions of this problem.

In Hoen and Poutre´ (2004) a MAS is presented for real-time vehicle routing problems. Solutions are obtained through an auction-based mechanism including Sandholm (2000) de-commitment possibility.

3.4 Conclusions

In this chapter, the application of MASs-based approaches to optimization problems has been investigated. Through Davidsson et al. (2007) framework, advantages and risks of MASs-based optimization approaches have been highlighted. Moreover, a broad literature review has been produced, in order to analyze the real impact of these methodologies in the OR/MS literature.

The results of the survey underline that MAS-based approaches are successfully employed to cope with a wide spectrum of optimization problems. In particular, MAS-based approaches are widely utilized to deal with scheduling problems. A relevant number of application is also devoted to Supply Chain planning problems.

In the following of this work, the applicability of multi-agent systems to another particular class of optimization problems will be surveyed. Precisely, a MAS-based approach for modeling and solving Location Problems will be developed.

Chapter 4
Location Problems: an overview

4.1 Introduction

The aim of this chapter is to provide an overview of a relevant class of optimization problems, namely Location Problems. Apart from offering an historical perspective of the development of the field of Locational Studies, the chapter will provide some generalities about the most widespread categories of Location Problems. This introduction will set the stage for the development, in the following of this work, of an agent-based framework for Location Problems.

4.2 A historical perspective

Tracking back the origins of Locational Studies is a controversial issue. An excellent historical perspective is provided by Kuhn (1973), whose work is based on a previous paper published by Zacharias (1913).

The problem of finding the spatial median, namely the *Mini-Sum Euclidean distance point*, was first roughly formulated in a basic version by the French mathematician Pierre de Fermat (1601-1665), who pose the challenge:

"*Let he who does not approve of my method attempt the solution of the following problem: given three points in the plane, find a fourth point such that the sum of the distances to the three given points is a minimum*".

Denote the three given points by $P_1=(a_1, b_1)$, $P_2=(a_2, b_2)$ and $P_3 = (a_3, b_3)$, and let $Q=(x, y)$ be the fourth point to be found. Being $d(Q,P_i)$ the Euclidean distance between Q and a generic point P_i, the sum of the distances from Q to the three given points is given by the following function, to be minimized:

$$f(Q) = d(Q,P1) + d(Q,P2) + d(Q,P3)$$

It is usual to credit the Italian scientist and student of Galileo Galilei, Evangelista Torricelli (1608-1647) with the solution. Other sources (Melzak, 1967) credit the Italian mathematician Battista Cavalieri (1598-1647) with both the formulation and the solution of the problem. Pottage (1983) states that Viviani and Roberval worked on the problem as well.

Actually, Cavalieri (1647) provided a geometrical method to find the solution to the problem. Simpson (1720) provided another graphical/geometrical method for solving this ancestral un-weighted median problem, also proposing some extensions to it, as, for example, the introduction of different weights for the points. Steiner, a geometer from the 19th century, wrote about the problem, without adding significant contribution to the literature. Chrystal (1885) provided the well-known geometrical solution to the un-weighted spatial one-center problem, namely the minimum covering circle for *n* co-planar points.

In the 20th century, engineers and economists started to consider practical applications and implications of the problem. Alfred Weber (1909) considered a three-points weighted version of the problem to locate a single warehouse in order to minimize the total travel distance between the warehouse and a set of spatially distributed customers. A mathematical appendix to the book provided a solution method for more complex cases (more points). Since then, this problem has been known as Weber Problem.

A different early location problem was formulated by Hotelling (1929), an economist who considered the problem of locating two competing vendors along a straight line: it was the first attempt to investigate facility location taking into account competition. This work was later extended by Smithies (1941).

Later on, the seminal work of Weiszfeld (1937) provided the first attempt to develop an iterative algorithm to solve Weber's problem, based on partial derivatives.

Hakimi (1964) introduced the network counterpart of Weber's problem, the *p*-median problem. In his seminal paper, he described the well-known property that for the *p*-median problem on a network, at least one of the alternative optimal solutions will consist entirely of vertices of the network. Hakimi (1964) also introduced the *p*-center problem on a network, consisting on locating facilities in such a way to minimize the maximum distance of a demand node from the closest facility; Kariv and Hakimi (1979) proved this problem to be NP-hard. Drezner and Wesolowsky (1978) developed an ingenious method for the multi-facility minimax *p*-center problem.

Cooper (1963, 1964 and 1967) introduced the location-allocation problem. In its general form, it is similar to the *p*-median problem. The allocation part of this type of problem implies that in addition to being located, new facilities are assigned particular demands that they are asked to satisfy.

Toregas et al. (1971) formulated the set covering problem, consisting in locating the minimum number of facilities required to cover a demand expressed in a plane. Also Minieka (1970) and Moore and ReVelle (1982) provided seminal contributions to the problem.

Nowadays, Location Problems are still a relevant sector in Operational Research/Management Science. A wide community of researchers and scholars is devoted to the development of new models and algorithms. These kinds of problems have gained importance also in real

industrial practice, especially in the area of Supply Chain Management, as highlighted by Melo et al. (2009).
In the following of the chapter, the basic elements of Location Theory will be presented. In addition, a taxonomy of the most common problems and their formulations will be provided.

4.3 Generalities

As suggested by Plastria (2002) a location problem can be characterized by the question "*Where are we going to put things?*" from which two more questions derive:

- *Which places are available?*
- *On what basis do we choose?*

The answer to the first question determines the location space. As location problems are a particular class of optimization problems, the second question requires the definition of the demand space and of an objective function, which can concern, for instance, the minimization of costs, damages or discomfort or the maximization of profits and quality of services. In some contexts, the objective can be defined by a single criterion, while, in more complex situations, more criteria must be monitored simultaneously.
Starting from their first application to industrial systems (Weber, 1909), location problems have received an uprising attention, related to the increasing demand of decision-making support systems in several application fields.
Location Problems find a wide range of possible applications. Indeed, it is possible to reproduce location decisions related to several facilities and services: industrial sites, warehouses, schools, hospitals, supermarkets, transportation facilities (subway stops and stations, parking lots).
The most of the Location Problems fall in the class of *punctual Location Problems*, namely problems in which the aim is to find one or more points to place facilities within a given domain. If facilities or services to be located cannot be represented by a punctual shape because of their extension, *non-punctual Location Problems* have to be considered. Network Design problems are a subset of this last class of problems.
Generally, Location Problems are characterized by the following elements:

- *Location Space;*
- *Demand Space;*
- *Metrics.*
- *Facilities Characteristics;*

- *Objective Function*;

Underlying hypothesis for each of these categories define different sub-classes of location problems, as stated in the following.

4.3.1 Location Space

Available locations can be long to three different kinds of sets. If available locations belong to an enumerable set, the location space can be defined as a discrete one (Figure 4.1).

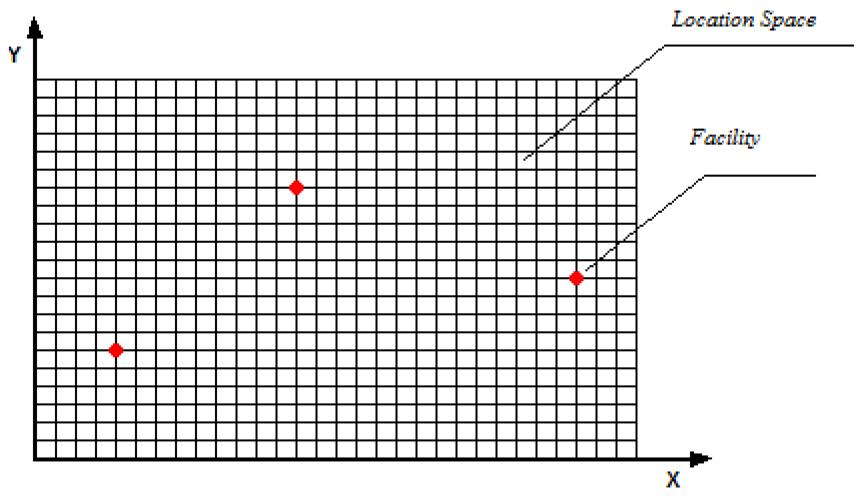


Figure 4.1 - Discrete Location Space

On the other hand, if facilities can be located in every point of a portion of plane, the location space turns out to be a continuous one (Figure 4.2).

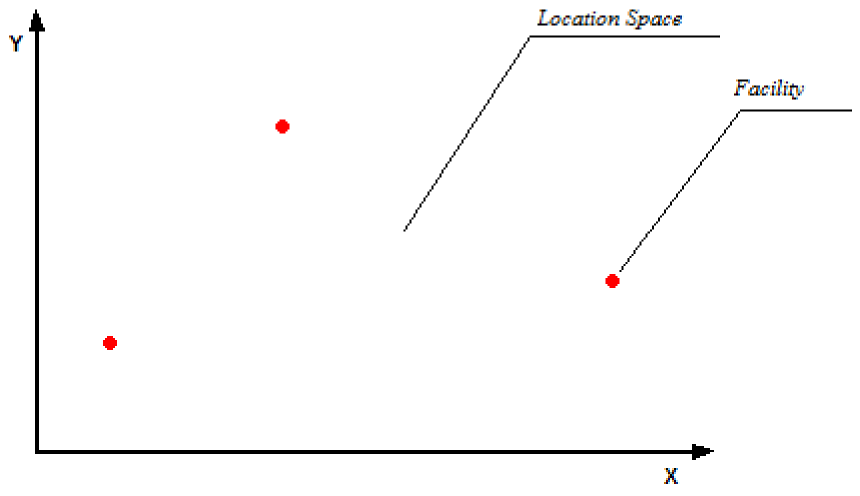


Figure 4.2 - Continuous Location Space

Another case is represented by a situation in which the location space is defined through a network structure (Figure 4.3). Depending on the structure of the problem, location can be allowed just on network nodes or at every point on the edges.

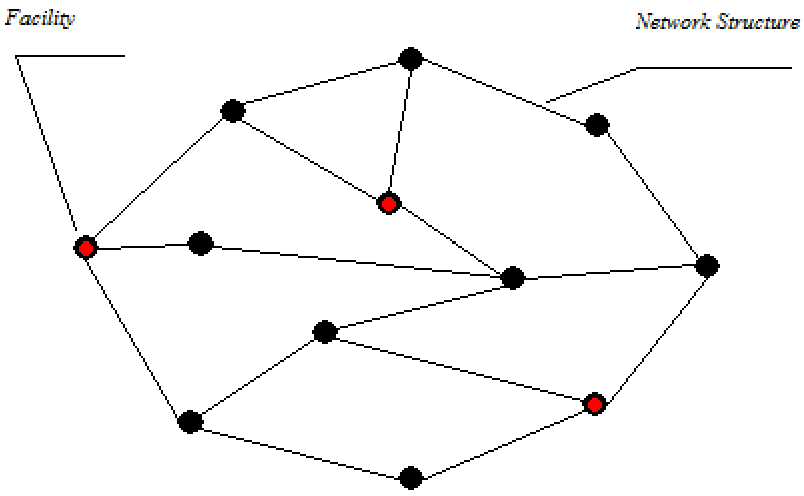


Figure 4.3 - Network Location Space

4.3.2 Demand Space

Demand represents the key element in the location choice, as its distribution has a direct impact on the positioning decision of the facilities.

Demand distribution reproduces the distribution of consumers or users in a given domain; thus, a demand space can be defined as well. Demand can be organized according to the following structures, already introduced for the location space:

- Discrete demand space, in which the demand is concentrated in an enumerable set of points;
- Continuous demand space, in which the demand is distributed over a continuous portion of space;
- Network demand space, in which the demand is distributed in the nodes or over any point in the edges of a graph-structure.

In the most trivial case, demand can have a uniform distribution, namely each element of the demand space is characterized by the same value of service demand. In more complex cases, in the demand space areas with different concentrations are defined.

Generally, demand and location spaces characteristics are independent: thus, it is possible to model location problems in which the location space is a continuous one and the demand space a discrete one, and vice-versa.

4.3.3 Metrics

Location decisions often depend on objective functions that express some form of dependencies on the distances between demand and facilities. Thus, another fundamental aspect is represented by the way of measuring these distances, namely the selected metric.

Given two points $P_i = (x_i,y_i)$ *e* $P_j = (x_j,y_j)$, the distance among them can be expressed as:

$$d_k(P_i,P_j) = ((x_i - x_j)^k + (y_i - y_j)^k)^{1/k}$$

where k=1 defines the *linear* or *Manhattan* metric, while if k=2 the *Euclidean* metrics is defined (examples are shown in Figure 4.4). The choice of the metric depends on the specific problem to be analyzed.

Practical experiments have shown the metric that allows to represent in a better way distances in real contexts is characterized by $1 < k < 2$.

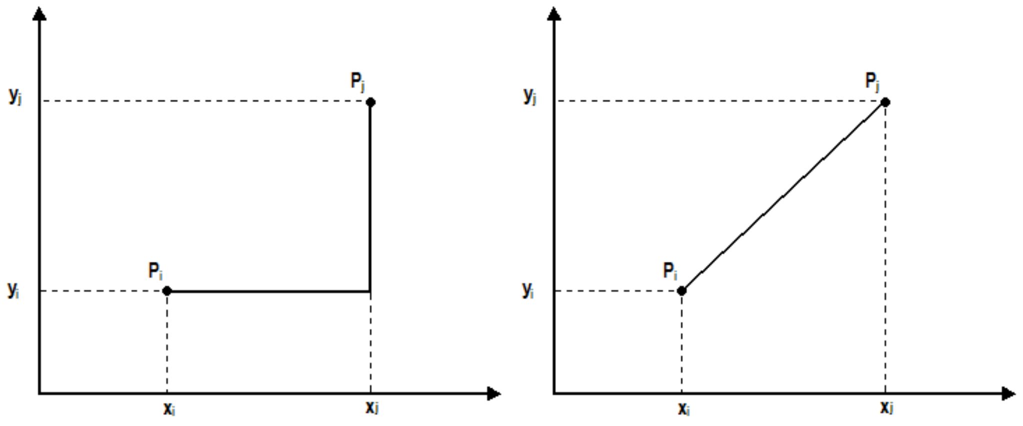


Figure 4.4 - Manhattan metric and Euclidean metric

4.3.4 Facilities Characteristics

Facilities to be located can be defined according to some distinctive features. Among these, we can cite:

- *Number of facilities*; in the simplest case, *single-facility problem* must be faced; otherwise, if more than one facility has to be located, a *multi-facility problem* is defined. In particular, the number of facilities to be located can be pre-defined or to be determined as an output of the problem itself;
- *Type of the facilities*; in the simplest case the facilities to be located are endowed with the same characteristics; a more complex case is defined when the facilities are not homogeneous, namely they present different forms, capacities, dimensions or qualitative characteristics;

- *Capacity of the facilities*, representing the maximum demand amount that can be served at the facility;
- *Costs*, including fixed costs (connected to the opening of the facilities, depending generally on the selected location, namely *location costs*) and variable costs (connected to demand satisfaction, namely *allocation costs*).
- *Covering radius,* representing (once fixed the metrics) the maximum distance that a facility located in a given point can reach. The covering radius defines a covering neighborhood (Figure 4.5), that will assume a different shape according to the selected metrics. Demand points included within the neighborhood will be considered reachable (*covered*) by the facility

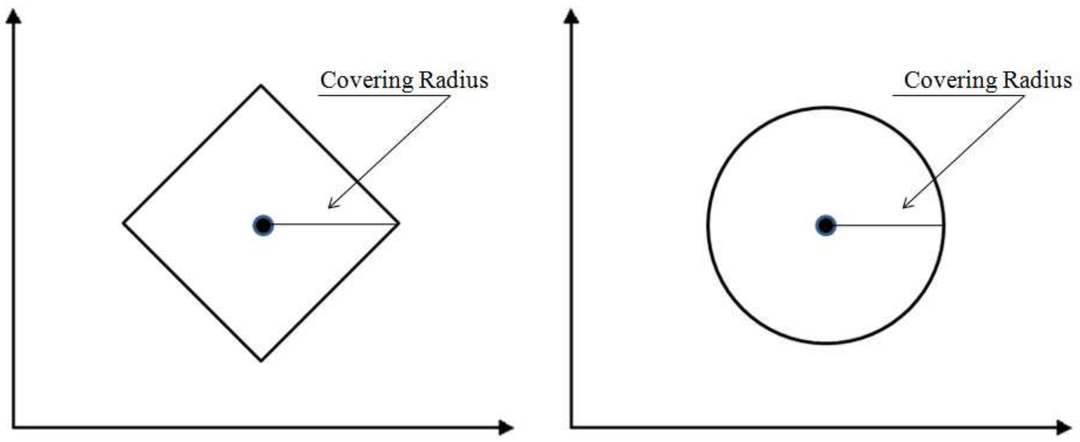


Figure 4.5 – Covering radii and neighborhoods in the case of Manhattan (left) and Euclidean (right) metrics.

4.3.5 Objective Function

Facilities are located according to a given objective function, with the aim of optimizing it. In the most common cases, an objective function can be expressed by the following criteria:

- Minimizing location/allocation costs (Mini-Sum objectives);
- Maximizing the total amount of demand covered by the located facilities (Covering objectives);
- Minimizing the distance of the most disadvantaged customer (Center objectives).

In many cases, it is also possible to optimize:

- a combination of more function;
- a vector function made up of more components (multi-criteria optimization);
- one of the functions and translating the others in constraints for the problems.

4.4 Classification of Location Problems

Like in other branches of optimization problems, also in Location Theory emerged the need for a precise and concise schema for describing problems in a synthetic way, capable of eliminating the ambiguity of verbal model descriptions.

Available classifications are similar to queuing (Kendall, 1951) and scheduling problems (Graham et al., 1979) taxonomies, based on a multiple-position string in which each position is representative of a distinctive characteristic of the problem.

Handler and Mirchandani (1979) suggested a 4-position scheme for network problems with center-type objective; Brandeau and Chiu (1989) give a taxonomy to distinguish location problems with respect to three criteria (objective, decision variables, system parameters) in table format, without providing a formal classification scheme. Eiselt et al. (1993) used a 5-position scheme specialized on competitive location models.

Carrizosa et al. (1995) present a 6-position scheme for classifying planar model where both demand rates and service times are given by a probability distribution.

Hamacher and Nickel (1998) designed a 5-position classification scheme to take into account every class of location problem in a single framework that represent, at the moment, the most detailed attempt to provide a universal classification of Location Problems.

The classification scheme has five positions written as:

$$Pos1/Pos2/Pos3/Pos4/Pos5$$

The meaning of each position can be described as follows:

- *Pos1*: Information about the number and type of new facilities;
- *Pos2*: Type of the location model with respect to the decision space. This information should at least distinguish between continuous, network and discrete models;
- *Pos3*: A description of particulars of the specific location model, such as information about the feasible solutions, capacity restrictions, etc;
- *Pos4*: Relation between new and existing facilities. This relation may be expressed by a distance function or by assigned costs;
- *Pos5*: Description of the objective function.

If no special assumptions are made, the position is filled by a symbol "•". For example, a symbol "•" in the fifth position means that each possible objective function is considered; a symbol "•" in the third position stands for a problem in which no particular restrictions are introduced.

The classification can be applied successfully to each one of the three main categories of location problems (Continuous, Discrete, Network) introduced above, as shown in the following.

4.4.1 Continuous Location Problems

Since continuous location models are the oldest location models and deal with geometrical representations of reality, a broad range of different location model types must be taken into account. We now describe some possible symbols in each position for continuous location models.

Pos1

It is characterized by a number $n \in \{1,\ldots,N\}$ expressing the number of facilities to be located and by a string of characters that specifies the shape of the facilities themselves, as follows:

	(a void string) to indicate punctual facilities;
l	to indicate that n lines have to be located;
p	to indicate that n paths (consisting of one or more lines) have to be located;
A	to indicate that n areas have to be located.

Pos 2

As stated above, it provides information about the location space. In particular:

$\mathbb{R}^d$	the problem has to be solved in a d-dimensional space;
P	the problem has to be solved in a plane (d = 2);
H	the problem has to be solved in a Hilbert Space.

Pos 3

Specifications about particular constraints of the problems can be stated as follows:

$\mathcal{F}$	the problem presents a feasible region; it is necessary that a solution x is such that $x \in \mathcal{F}$.
$\mathcal{B}$	Barrier, i.e., neither placement of new facilities nor trespassing is allowed.

Further specifications can be provided about weights associated with demand points.

Pos 4

This position specifies the distance function, for example:

l_p	the distance is defined by l_p-norm (for example, l_2 is the Euclidean norm);
d_{nonhom}	the distance is not the same in the whole domain of the problem.

Pos 5

As regards the objective function, the default case accounts for an objective function to be minimized. The nature of the objective function also provides, intuitively, information on the demand space, as follows:

$\sum$	ordinary Weber objective function (weighted sum of the distances among demand and facilities). The symbol $\sum$ also signifies that the demand space is a discrete one;
max	objective function that expresses the maximum distance from a demand point to the closest facility (*p*-Center problem);
$\int_d$	Weber objective function within a continuous demand space (the weighted sum is an integral sum);

Examples

Given the adaptation of the 5-positions schema to continuous location problems, the most common problems can be described adopting it as follows:

1/P/●/l_2/$\sum$

Classical Weber problem with Euclidean distance. Just one facility has to be located in a planar space with no other constraints.

1/P/●/●/$\sum$

Class of planar Weber single-facility problems, with any kind of distance.

N/P/(mc)/●/$\sum$

Class of planar Weber problems, with any kind of distance and any number of facilities. Facilities have to satisfy some mutual conditions.

N/P/●/●/max

Class of planar center problems, with any kind of distance and any number of facilities.

4.4.2 Discrete Location Problems

In the following, the meaning of the five positions for Si procederà ora ad illustrare il significato delle 5 posizioni proposte da Hamacher per i problemi di localizzazione discreti.

Pos 1

n	$n \in \{1,\dots,N\}$ states the number of facilities to be located;
#	the number of facilities to be located is unknown and it is part of the problem;
##	two different kind of facilities have to be located; the number of the facilities is unknown for both the kind of facilities.

Pos 2

D the location space is a discrete one; this is the only option for this kind of problems.

Pos 3

cap	facilities have limited capacities;
bdg	there is a budget constraint;
d_{max}	it is assigned a maximum distance constraint between demand and facilities;
d_{min}	it is assigned a minimum distance constraint between demand and facilities.

Pos 4

Any restrictions and particulars of given costs cij can be speci®ed (e.g., triangle inequality, non-negativity, etc.).

Pos 5

Any objective function from the continuous case can be adopted,

$\sum_{\mathbf{comp}}$	Competitive location model;
$\sum_{\mathbf{uncov}}$	Coverage objective function;
$\sum_{cov} + \sum_{uncov}$	Covering objective function;

Examples

After having illustrated the classification schema for discrete problems, some examples of common problems can be provided.

N/D/●/●/∑

Discrete *N*-Median (Weber) problem without restrictions and distance specification.

#/D/●/●/∑

The so-called uncapacitated facility location model, or Simple Plant location problem, in which the number of facilities to be located is unknown *a-priori*.

#/D/d_{max}*,bdg/●/*$\sum_{uncov}$

Coverage model, in which the number of facilities to completely cover the demand is unknown *a-priori*. Facilities have to be placed respecting a maximum distance constraint with respect to demand points; there are also restrictions on available budget.

4.4.3 Network Location Problems

The 5-position schema can be particularized as follows.

Pos 1

As in the other cases, $n \in \{1,\dots,N\}$ states the number of facilities to be located; # indicates that the number is unknown and its determination is part of the problem.

As regards the kind of facilities to be located:

	(a void string) indicates that *n* points have to be located
p	*n* paths have to be located;
T	*n* trees have to be located;
G	*n* subgraphs have to be located.

Pos 2

As the problem is defined on a network, this position clarifies the characteristics of the underlying graph:

G	the problem is defined on an undirected graph;
G_D	the problem is defined on a directed graph;
T	the graph is a tree.

Pos 3

Same options from the continuous case.

Pos 4

In a network, the distance is always measured with reference to the shortest path. Thus, it has to be specified from where to where distances are measured.

In the notation *d(-,-)* the first element determines conditions to be respected for existing and new facilities, as follows:

d(V,V)	both new and existing facilities has to be positioned in graph nodes;
d(V,G)	existing facilities are positioned in graph nodes, while new facilities can be positioned in any point of the graph (also on edges);

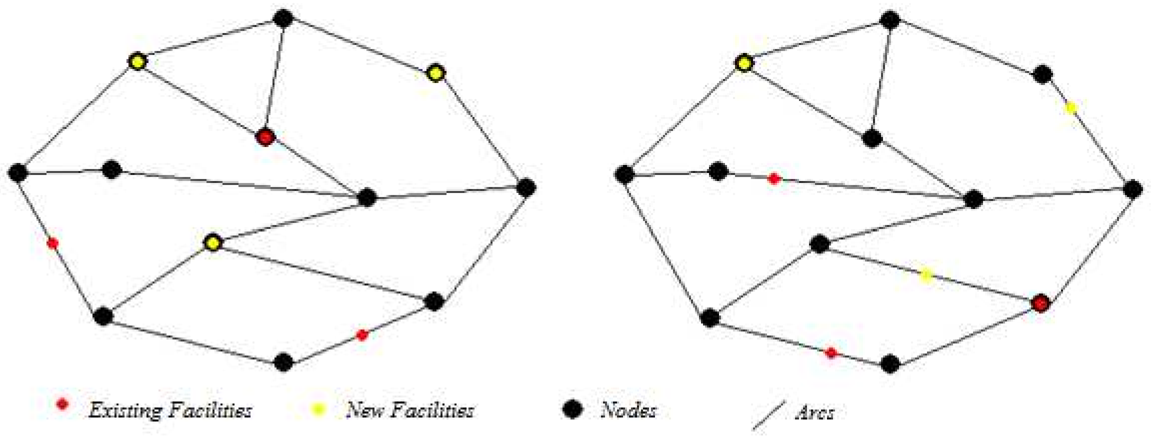


Figure 4.6 - Problems *d(V,V)* and *d(V,G)*

d(G,V)	new facilities can be positioned on graph nodes, while existing facilities are placed on any point on the graph;
d(G,G)	both kind of facilities can be placed everywhere on the graph.

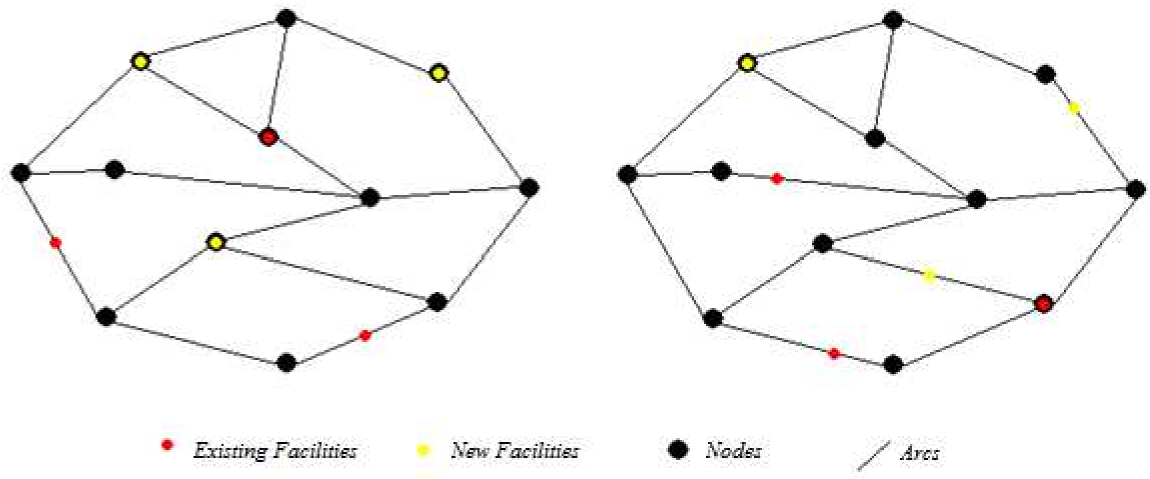


Figure 4.7 - Problems *d(G,V)* and *d(G,G)*

Examples of the four kinds of problems above described can be found in Figures 4.6 and 4.7.

Pos 5

Any of the objective functions listed for the continuous case which are meaningful in the network environment.

Examples

1/G/●/d(V,G)/∑

1-Median network problem, consisting in locating a facility on a non-oriented graph. Existing facilities are positioned on nodes, while facilities to be located can be placed everywhere. The objective function to be optimized is Weber's one.

4.5 Some well-known models of Location Problems

Within the framework of the proposed classification of Location Problems and from an analysis of the literature, it can be stated that there exists two predominant objective functions in location science: the previously introduced Weber objectives and Center-type objectives. These objectives are also known as minisum and minimax problems, respectively.

Other objective functions are also studied within the location science community, especially recently.

The most notable of these are the set covering and maximal covering objective functions. The former of these two objectives attempts to locate the minimum number of new facilities such that a prescribed distance constraint to existing facilities is not violated. In contrast, the latter strives to locate a given number of facilities to best meet the (weighted) demands of the existing facilities subject to a maximum distance between new and existing facility. It should be noted that for the set covering formulation, because all of the demands must be met (covered) regardless, the relative weight of the demands generated by the existing facilities are inconsequential, whereas in the maximal covering objective some existing facility demands may be left unmet (uncovered).

Objectives that involve equity issues are also investigated, like, for example, minimizing the variance or the range of distances between demand points and facilities.

The diametrics of these objective functions also exist (maxisum, maximin, minimal covering), although they are somewhat less studied.

In the following, a brief review on these classes of location problems will be provided.

4.5.1 p-Median like problems

The *p-Median* problem aims at the minimization of the weighted sum of the distances between *p* facilities to be opened and a set of demand points. Several versions of the problem have been defined in the literature.

The version of this problem in which the location space is continuous, often indicated as the Multisource Weber Problem (MWP), belongs to the class of *NP*-hard optimization problems, as shown in Megiddo and Supowit (1984). Given a set of demand points $i \in I$, located in (x_i, y_i) and the coordinates $(x_a, y_a) \in S \subset \Re \; x \; \Re$ for a number p of facilities, a possible formulation for the MWP is the following one (Klose and Drexl, 2005):

$$\min \sum_{i \in I} \sum_{a=1}^{p} w_i d_i (x_a, y_a) z_{ia}$$

subject to

$$\sum_{a=1}^{p} z_{ia} = 1 \qquad \forall i \in I$$

$$z_{ia} \in \{0,1\} \qquad \forall i \in I, a = 1, ..., p$$

$$x, y \in \Re^p$$

being $d_i = \sqrt{(x_a - x_i)^2 + (y_a - y_i)^2}$ in the case of Euclidean metrics. In this model, z_{ia} equals 1 when a demand point i is assigned to a facility a.

Fast heuristic methods to cope with the MWP are considered and compared in Brimberg *et al.* (2000) and Hansen *et al.* (1998), while in Aras *et al.* (2006) the problem is solved using neural networks. As concerns exact algorithms, the first attempt to solve instances of the MWP is proposed by Kuenne and Soland (1972). Later, different approaches are proposed by Rosing (1992), Chen *et al.* (1998) and duMerle *et al.* (1999). A recently developed branch-and-price algorithm (Righini and Zaniboni (2007)) permits to find the optimal solution on instances with some thousands of points and some hundreds of sources in less than three hours on a PC.

The diametrical version of the problem takes into account the necessity of locating *obnoxious* facilities. A facility is called *obnoxious* when it is desired to locate it as far as possible from an inhabited centre. Obnoxious location problems have received significant attention in the last decades, due to the increasing environmental and social impact of facilities such as power plants and dump sites. Thus, in the case of obnoxious facilities, the *p*-Median objective can be turned into an anti-p-Median objective, in which the aim is to maximize the weighted sum of the distances between *p* facilities to be opened and a set of demand points.

4.5.2 p-Center like problems

The p-center problem concerns the location of p facilities (centers) so as to minimise the maximum of the distances from each customer (demand point) to its nearest facility. This problem may address, for instance, the location of public facilities, schools, emergency services, where the objective is to design a system in such a way that no customer has to travel too far (or each customer could be reached in a reasonable amount of time).

The *p*-center problem on a network (Hakimi, 1964) was later addressed by Hakimi (1965), Minieka (1970, 1977), Elzinga and Hearn (1972). Kariv and Hakimi (1979b) proved this problem to be NP-hard.

Francis (1967) provided some insight to the *p*-center problem on a plane. This same problem (with slight variations) was later addressed by Wesolowsky (1972) who investigated the problem with rectilinear distances. Drezner and Wesolowsky (1978a) developed a solution method (which is now often cited as the Drezner-Wesolowsky method) for the multi-facility minimax *p*-center problem. Drezner and Wesolowsky (1978b) researched the problem under an arbitrary l_p distance metric; Chen (1983) looked at the problem with Euclidean distances; Ward and Wendell (1985) formulated the now well-known “block-norm” for the distances involved. Masuyama et al. (1981) and Megiddo and Supowit (1984) were able to show that the Euclidean and rectilinear cases of this problem are NP-complete, respectively.

In the case of a continuous location space, the p-center problem consists in finding a subset A Within a region $S \subset R^2$, such that:

$$\min_{A \subset S} F(A)$$
$$F(A) = \max_{i \in I} w_i \min_{a \in A} d(i,a)$$

Subject to:

$$||A|| = p$$

The objective function represents the minimization of the maximum weighted distance between each demand point i and the closest facility a. The constraint expresses that exactly p facilities are going to be located.

The diametrical version of the problem is represented by the anti-p-center problem. In this problem, the aim is to find p facility locations which will maximize the minimum distance between demand points and their respective nearest facilities. Typically, this problem is used to model the location of the above defined obnoxious facilities such as incinerator plants, hazardous waste sites, unsightly factories.

In the case of a continuous location space, the anti-p-center problem consists in finding a subset A within a region $S \subset R^2$, such that:

$$\max_{A \subset S} F(A)$$
$$F(A) = \min_{i \in I} w_i \min_{a \in A} d(i,a)$$

Subject to:

$$||A|| = p$$

4.5.3 Covering problems

Location Covering Models are another class of problems, in which the objective is to ensure coverage to given demand points. A demand point is said to be covered by a certain facility if the distance between the two points is lower than a certain threshold, or required distance (*RD*). Models of this type generally address the location of urban public facilities, especially emergency facilities. Church and ReVelle (1974) propose the p-Maximal Covering Location Problem (MCLP), which seeks to locate p facilities that can cover the maximum amount of demand. Given a set of demand points $i \in I$, located in (x_i, y_i) and the coordinates $(x_a, y_a) \in S \subset \Re \; x \; \Re$ for a number p of facilities, a possible formulation for the the p-Maximal Covering Location Problem with facility placement on the entire plane can be derived from Mehrez (1983):

$$\max \sum_{i \in I} w_i \zeta_i \tag{4}$$

subject to:

$$\sum_{a=1}^{p} z_{ai} \geq \zeta_i, \ \forall i \in I$$

$$z_{ai} d_i(x_a, y_a) \leq RD \quad \forall i \in I, a = 1,\dots,p$$

$$z_{ia}, \zeta_i \in \{0,1\} \quad \forall i \in I, a = 1,\dots,p$$

$$x, y \in \Re^p$$

The variables ζ_i and z_{ai} are binary. The variable ζ_i is equal to *1* if a demand w_i located in *i* is covered, (*0* otherwise) and the variable z_{ai} is equal to 1 if the demand concentrated in *i* is covered by a facility located in *a*. Constraints (5) ensure that a demand point that is considered to be covered has at least one facility within the required distance; constraint (6) ensures that the variable z_{ai} is equal to *1* if the demand located in *i* can be covered by the service located in *a* within the required distance *RD* (*0* otherwise). The problem is generally complex, and several heuristic methods have been developed to deal with it. A survey on this topic is presented by Galvao *et al.* (2000).

As shown for the previous classes of location problems, also in the case of Covering Location Problems, a diametric version regarding obnoxious facilities can be considered. Berman and Huang (2008) introduced the Minimum Covering Location Problem with Distance Constraints (MCLPDC) that, through locating a fixed number of facilities, aims to minimize the number of covered customers (where, as stated above, a customer is considered covered if her distance to the closest facility is less than a pre-determined radius) by respecting a constraint on the minimum distance among facilities themselves. To motivate the MCLPDC, they consider the problem of locating facilities that may pose a serious danger to the individuals living nearby. The fewer people "covered" the better. The minimum distance constraints express the condition that sometimes, for safety reasons, those facilities should also be separated (e.g., if several reactors are clustered in the same region, the problem would turn out to be trivial, but facilities may all be attacked by an aggressor). Early versions of this objective have been presented by Drezner and Wesolowsky (1994) and by Plastria and Carrizosa (1999).

Moreover, several extensions have been proposed to the basic *p*-Maximal Covering problems. Storbeck (1982) and Benedict (1983) describe a formulation of the problem in which apart from the maximization of the demand covered by the facilities, the demand covered by at

least two facilities has to be maximized. Following the same idea, Daskin et al. (1988) establishes a version of the problem in which the *back-up* coverage has to be maximized.
Chung et al. (1983) introduce capacity constraints for facilities to be located; in this way, the demand concentrated in a given point could also be assigned to a facility other from the closest one. Current and Storbeck (1988) the authors introduce the possibility for demand points to be served by more than a facility according to a given demand distribution model.
Pirkul e Schilling (1991) introduce a multi-criteria objective function aimed at re-assigning demand points out of covering radius of any located facility to the closest facility, respecting capacity constraints.
Karasakal and Karasakal (2004) develop a more sophisticated coverage concept by introducing the concept of partial coverage. Each facility is endowed with two covering radius: a minimum covering radius and a maximum covering radius; demand points within the minimum radius are considered to be totally covered, while the ones falling in the area between the circles described by the two radii are considered to be partially covered. Berman et and Huang (2008) propose the covering radius as a variable associated with a cost to be minimized.

4.5.4 Equity problems

Another kind of problems can be defined when the objective is a measure of "equity" from the demand points to the set of facilities (Eiselt and Laporte (1995)). Equity is sought by minimizing the inequality in the facility-demand points distances. Several objectives have been introduced in order to achieve this goal. Among the others, we can cite: minimizing the variance of distances (Maimon, 1986), minimizing the Gini Coefficient (Maimon, 1988), minimizing the range of distances in the plane (Schöbel, 1999).
As regards the minimum variance objective, being $\mu(x_a,y_a)$ the average distance among the demand points i and the facility a of coordinates (x_a,y_a) and $\sigma^2(x_a,y_a)$ the variance of the distances, the Single Facility Minimum Variance Location Problem in the Euclidean plane aims at defining the position (x_a,y_a) of the facility which minimizes $\sigma^2(x_a,y_a)$. Drezner and Drezner (2007) solve to optimality large instances of this problem using a *Big Triangle Small Triangle* (BTST) approach.

4.6 Conclusions

In this chapter an overview of a relevant class of optimization problems, namely Location Problems, has been proposed. After having offered an historical perspective of the development of the field of Locational Studies, the chapter has provided some generalities about the most widespread categories of Location Problems.

This introduction has set the stage for the development, in the following of this work, of an agent-based framework for Location Problems.

Chapter 5
An agent-based framework for Location Problems

5.1 Introduction

The high relevance of location problems in the operations research literature arises from their wide spectrum of real applications, including decision optimization in industrial management, logistics and territorial planning. Most of these optimization problems fall in the class of *NP*-hard problems, motivating the search for heuristic and approximated algorithms. Currently, a great interest is being devoted to those optimization approaches yielding a concrete integration with spatial analysis instruments (such as Geographical Information Systems), that provide the user with an easy visualization of input data and optimization results.
As shown in Chapter 2, Agent-Based computing was recently proposed as an alternative to mathematical programming in order to deal with problems whose domains are concurrently distributed, complex and heterogeneous, also thanks to the availability of many commercial and open source codes including graphical interfaces for the elements of the problem.
In this Chapter we propose a general Agent-Based framework for modeling various location problems. Together with its description, we present some computational results confirming the suitability and the effectiveness of the proposed approach.

5.2 Theoretical Framework

As it can be derived from Billari *et al.* (2006) and Weiss (1999), and as it has been recalled in Chapter 1, the development of an Agent-Based Model needs a complete description for a set of basic *building blocks*, as follows.

- *The object of the simulation.* It has to be specified what is the phenomenon/problem to be reproduced, defining the space where the simulation takes place.
- *The agents' population.* Agents can be grouped in different categories with common characteristics reproducing the various components of the system.
- *The adaptive capability of each agent category.* Agents of each category present a specific adaptive capability, i.e. the degree of re-activeness and pro-activeness.
- *The interaction paradigm among agents.* Each agent can interact with agents of the same or of other categories. In the literature, several interaction paradigms have been defined, such as cooperation, competition, negotiation (see for instance Weiss (1999)).

On the base of the selected paradigm, the agents evolve in the simulation space in a different way.

Given this peculiarity in dealing with the representation and the simulation of complex systems, ABMs have been recently applied to solve optimization problems whose domains present several inter-related components in a distributed and heterogeneous environment (Weiss (1999), Wooldridge (2002)), sometimes combined to other optimization techniques.
Some of the characteristics of ABMs suggest the possibility to apply this approach to model and solve location problems. The approach appears to be particularly interesting when the location space is a planar region (whose points represent available locations) and the demand can be represented by an enumerable set of discrete points.
Suppose that we have to locate p facilities in a continuous space in which n demand points are positioned. In order to define an Agent-Based framework, in the following we describe how each block previously illustrated can be specified to represent the problem.

The object of the simulation. The object of the simulation is to reproduce all the elements of the problem and to define the appropriate rules that agents should follow. The environment of the simulation is represented by the location space, i.e. a portion of plane (for instance a rectangle of base b and height h) where agents are positioned. We assume that distances between elements are defined by an Euclidean metric. Due to the flexibility of the ABMs, the adaptation of the model to different metrics is straightforward.

The agents' population. We distinguish between two main agent categories (see Figure 5.1):

- a set P of "passive" agents representing the demand points with an associated demand $w_i \, \forall i \in P$;
- a set A of "active" agents representing the facilities to be located.

The adaptive capability of each agent category. The two agent categories present different adaptive capabilities. Passive agents do not change position but they interact with the active agents in an autonomous way. They are neither re-active, as they do not react to any signal, nor pro-active as they do not pursue any objective. On the other hand, the active agents are both re-active, as they answer to the presence of passive agents, and pro-active, as they move in the location space searching for positions according to a given objective.

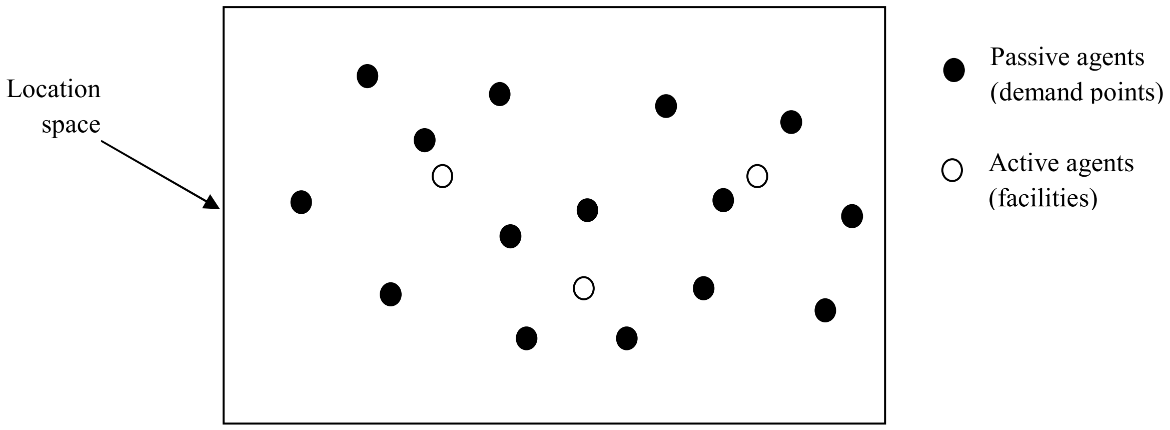


Figure 5.1 – Location space and agent categories

The interaction paradigm among agents. As mentioned before there exist different paradigms to define the interaction among agents. In this context, we adopt the Artificial Potential Fields (APF) paradigm, based on some concepts from physics and biology (see Ferber (1999), Kathib (1986)). The paradigm assumes that the agent behavior is regulated by the action of forces. In this context we suppose that two forces operate on each active agent $a \in A$ (Figure 5.2):

- a *demand-driven* force, $\underline{F}^{d}_{ia}$, due to the presence of a passive agent $i \in P$ which pushes the agent a toward the position of i;
- a *repulsive* force, $\underline{F}^{r}_{ja}$, determined by the presence of an active agent $j \in A$ which pushes the agent a in the opposite direction of j.

The intensity of the two forces is a function of the distance between the agents as widely used in spatial interaction models (see, for instance Fotheringham and O'Kelly (1989), Sen and Smith (1995), Serra and Colomè (2001)).

According to the APF paradigm we suppose that these forces are significant only within a given distance from the agent $a \in A$. In order to define the forces, the paradigm introduces some calibration parameters expressing the width of the neighborhood within which each force is significant.

In this way we can define a *resulting demand-driven* force (Figure 5.3a):

$$\underline{F}^{d}_{Ra} = \frac{\sum_{i \in P_{rd,a}} \underline{F}^{d}_{ia}}{|\, \mathrm{P}_{\mathrm{rd,a}} \,|}$$

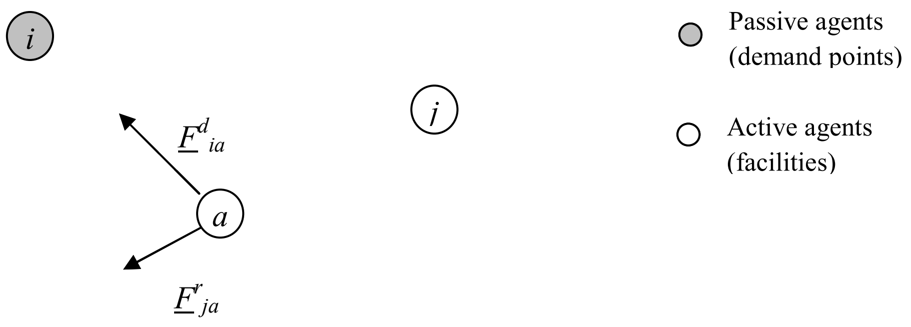


Figure 5.2 - Forces operating on an active agent

where $P_{rd,a}$ is the set of passive agents whose distance from a is within a given radius r_d. In the same way the *resulting repulsive* force is given by (Figure 5.3b):

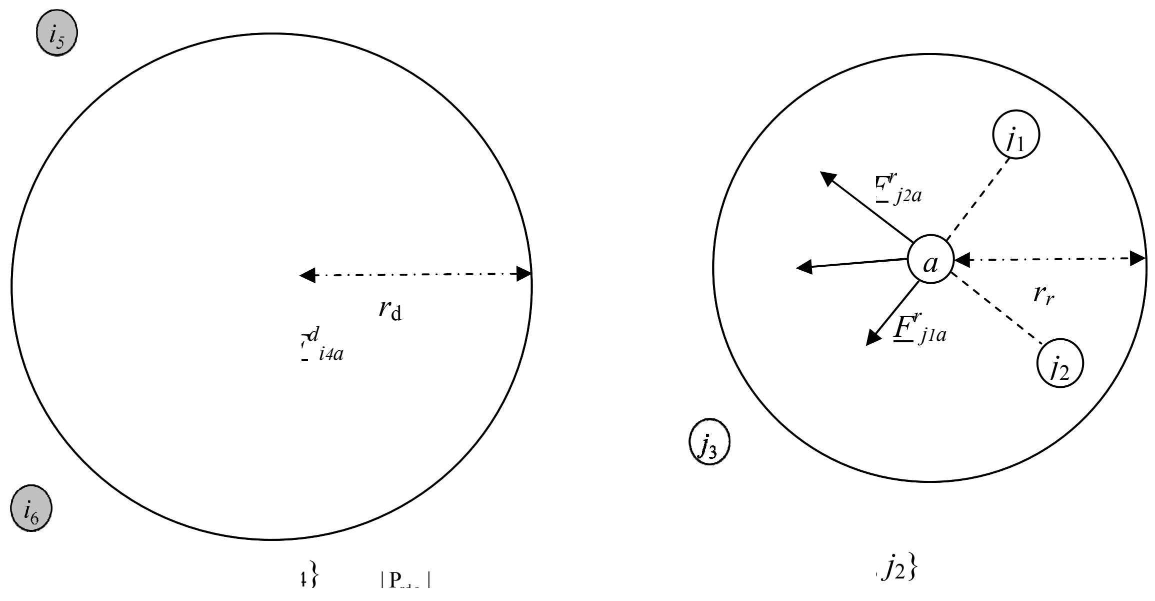
$$\underline{F}^r_{Ra} = \frac{\sum_{j \in A_{rr,a}} \underline{F}^r_{ja}}{|\mathrm{A}_{\mathrm{rr,a}}|}$$

where $A_{rr,a}$ is the set of active agents whose distance from a is within a given radius r_r.

The movement of the agent a is finally determined by the total force $\underline{M}_a$, calculated as a convex combination of the two forces:

$$\underline{M}_a = \alpha \underline{F}^d_{Ra} + (1-\alpha)\underline{F}^r_{Ra} \qquad (7)$$

being α a parameter, $0 \le \alpha \le 1$, expressing the relative weight of each resulting demand-driven and distributive forces.



(a) the resulting demand-driven force (b) the resulting repulsive force

Figure 5.3 – The resulting forces operating on an active agent

5.3 Adaptations of the ABM Framework to several Location Problems

The described framework can be particularized to deal with different location problems and to consequently solve them through the development of proper procedures to be implemented in a given environment. In particular we show how the forces can be specified in relation with the problems illustrated in Section 3, according to the specific objective of the problem. In the following we define a distance vector $\underline{d}_{ba}$ between two agents *b* and *a* as the vector applied to the agent *a* and directed toward the agent *b* with an intensity equal to the distance $||\underline{d}_{ba}||$ between the agents.

5.3.1 The p-Median like problem

We start by describing the adaptation of the proposed ABM framework to solve a class of location problems in which it must be minimized an objective function which includes a weighted sum of the distances between *p* facilities to be opened and a set of demand points. We refer to this as the "*p*-Median like problem".

In this case the demand-driven force can be expressed by

$$\underline{F}^{d}_{ia} = w_i\, \underline{d}_{ia} \qquad \forall i \in P_{rd,a} \qquad (8)$$

where w_i represents the demand associated to *i* and $\underline{d}_{ia}$ the above mentioned distance vector. In practice we suppose that the influence of a demand point *i* on the facility *a* decreases the closer the facility moves towards such a demand point. Indeed, if the active agent reaches exactly the position of the demand point, the demand-driven force becomes zero.

As regards the distributive force we assume that:

$$\underline{F}^{r}_{ja} = -\frac{\underline{d}_{ja}}{||\underline{d}_{ja}||}\frac{1}{||\underline{d}_{ja}||} \qquad \forall j \in A_{rr,a}$$

The influence of another facility *j* on the facility *a* is inversely proportional to the distance $||\underline{d}_{ja}||$: the closer the two facilities, the more intense the force $\underline{F}^{r}_{ja}$ that will tend to push the agent *a* away from the agent *j*.

The proposed adaptation of the ABM framework to the class of *p*-Median like problems can be applied as a heuristic approach to solve instances of the classical *p*-Median problem described in

Section 3, since, given any instance of that problem, it provides a feasible solution in finite computational time, whose quality will be experimentally evaluated in the next Section, through the comparison with the results arising from the related literature for the *p*-Median problem. According to Drezner (1987) we also observe that, even if the *p*-

Median problem does not explicitly consider mutual distances among facilities, the presence of distributive forces allows avoiding facilities overlapping that could yield bad quality objective function values.

The values of the radii r_d and r_r for the determination of P_{rd} and A_{rr} and α are calibration parameters.

The value of r_d can be set as $r_d = \left\lceil \frac{\min(b,h)}{p+1} \right\rceil$ being b and h respectively the base and the height of the location space.

As regards the value of r_r a default value of 1 space unit can be considered. In presence of possible constraints on the minimum distance among the facilities, r_r can be fixed according to this aspect.

The value of α can be set equal to 0.5, so the resulting forces are supposed to have the same relative weight.

5.3.2 The p-Maximal Covering like problem

We consider now the adaptation of the ABM framework to the class of p-Maximal Covering like problems, in which the objective is to ensure the coverage to some demand points under threshold constraints. In this case, the demand-driven force is expressed as follows:

$$\underline{F}^d_{ia} = \frac{w_i}{p_i} \underline{d}_{ia} \qquad \forall i \in P_{rd,a}$$

where p_i is the number of active agents covering the demand point i, i.e. within a distance RD from i. In this way we suppose that if a demand point i is covered by more than one facility, its demand-driven force is equally shared among those facilities.

In this case the values of the radii r_d and r_r can be fixed equal to RD and $\alpha=0.5$.

5.3.3 The Minimum Variance like problem

In the adaptation of the ABM framework to the Single Facility Minimum Variance like problem, since we deal with a single facility location, the repulsive forces are not present. The expression of the demand-driven force is calculated as in (8).

Due to the absence of repulsive forces, $\alpha=1$ and $r_r=0$, the only parameter to be calibrated is r_d whose value can be fixed as already shown for the p-Median like case.

A summary of the adaptations of the Agent-Based framework to the illustrated location problems is reported in Table 1.

Problem	*Demand-driven force*	*Repulsive force*	*Calibration parameters*
p-Median like problem	$\underline{F}^{d}_{ia} = w_i \, \underline{d}_{ia} \quad \forall i \in P_{rd,a}$	$\underline{F}^{r}_{ja} = -\frac{\underline{d}_{ja}}{\lVert \underline{d}_{ja} \rVert} \frac{1}{\lVert \underline{d}_{ja} \rVert} \quad \forall j \in A_{rr,a}$	α, r_d, r_r
p-Maximal Covering like problem	$\underline{F}^{d}_{ia} = \frac{w_i}{p_i} \underline{d}_{ia} \quad \forall i \in P_{rd,a}$	$\underline{F}^{r}_{ja} = -\frac{\underline{d}_{ja}}{\lVert \underline{d}_{ja} \rVert} \frac{1}{\lVert \underline{d}_{ja} \rVert} \quad \forall j \in A_{rr,a}$	α, r_d, r_r
Single Facility Minimum Variance like problem	$\underline{F}^{d}_{ia} = w_i \, \underline{d}_{ia} \quad \forall i \in P_{rd,a}$	--------	r_d

Table 5.1 - Summary of the expressions of the forces for the illustrated problems.

5.4 Implementation of the Framework

The illustrated framework has been implemented within the *NetLogo* Agent-Based simulation environment (http://ccl.northwestern.edu/netlogo, see Appendix I) using the proprietary programming language and its Java architecture. *NetLogo* allows reproducing the two agent categories above introduced. In particular, passive agents are represented by cells in a grid network, being each cell identified by a couple of integer coordinates.

In the implemented procedure (for the detailed code, see Appendix II), whose scheme is represented in Figure 5.4, it is possible to distinguish the following steps.

1. *Initialization*

The parameters of the problem (number of facilities p, values of the radii r_d and r_r, α, objective function, expression of the forces) are defined.

2. *Individuation of the initial solution*

The position of p active agents in the location space is randomly determined according to a uniform distribution with values ranging within the extreme coordinates of the location space.

3. *Evolution of the current solution*

For each active agent a located in the current positions, the total force $\underline{M}_a$ according to (7) is calculated so that the active agents change position on the base of this force and the solution assumes a new objective function value.

4. *Diversification*

If a diversification criterion (defined in terms of number of non-improving iterations, fixed a-priori as a parameter) is satisfied, a diversification move is enacted and the procedure goes back to the step 2; otherwise, it goes to step 5.

5. *Stopping criterion*

If a stopping criterion is satisfied the procedure ends; otherwise it goes back to the step 3.

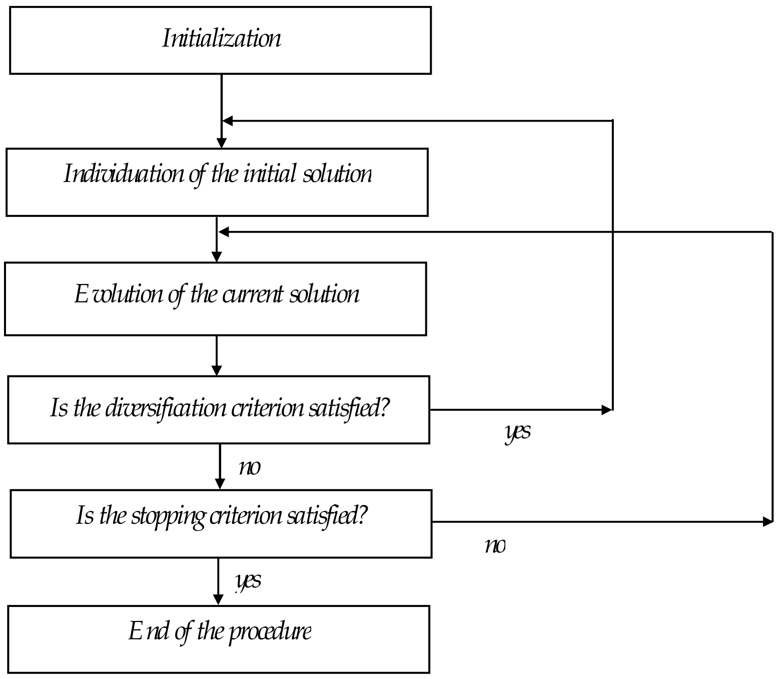

Initialization
Individuation of the initial solution
Evolution of the current solution
Is the diversification criterion satisfied?
yes
no
Is the stopping criterion satisfied?
no
yes
End of the procedure

Figure 5.4 – The scheme of the implemented procedure

The procedure behaves as a metaheuristic searching for better solutions thanks to an evolutionary mechanism which is performed until a diversification or a stopping criterion are satisfied. On the base of a diversification criterion the procedure restarts from a new initial solution.

Possible stopping criteria are represented by a given total number of evolution iterations or a fixed running time, to be defined as parameters.

It is worth to note that in order to store the evolution of the current solution during the run of the procedure, two further categories of agents have been implemented. A first category of agents stores the evolution of the current local best solution found by the algorithm, by following the search process by active agents; another category of agents stores the global best solution found in the search process. Both the agentsets have a cardinality p, thus visually represent on the plane the position of each facility in the current best local or global solution.

5.5 Computational experiences

We illustrate some examples of application of the Agent-Based framework to the location problems introduced and described in Sections 3 and 5, in order to show the capability of the proposed approach to solve these problems and to analyze the provided performances in terms of computational times and quality of the solution. The procedure was run on a PC with a

Dual-Core T2250 2.0 GHz CPU and *2* GB of RAM. In all the experiments the calibration parameters were set according to the criteria illustrated in Section 5.
As stopping criterion we fixed a number of *150* iterations while, to start the diversification,we considered *10* non-improving iterations.
In order to evaluate the quality of the provided solution, we calculated the gap from the known optimal solution as

$$Gap = \left(\frac{ABMBestSolution - OptimalSolution}{OptimalSolution} \right) * 100$$

5.5.1 Solving p-Median problem instances

We applied the proposed framework to solve one of the benchmark problems (*Bongartz287*) available for *p*-Median problem (the data of the instance are available at the website http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/location.html). This test problem is characterized by *287* demand points with variable demand values, whose coordinates assume integer values in the range [*0,50*]. We used a *100x100* grid of passive agents; thus, each grid point can be associated or not to a demand point. We solved the problem for *p* varying from *2* to *10*.
Results reported in Table 5.2 show that ABM finds near optimal solutions in limited computing times.

5.5.2 Solving p-Maximal Covering problem instances

In absence of benchmark instances for this version of the problem, we generated instances in a *100*x*100* location space (for a total number of *10000* demand points) with known optimal solutions according to the following criteria. Once fixed the distance threshold *RD* (we assume *RD=4* space units) and given the number *p* of facilities to be opened, each instance is produced through the random generation of *4* sets of *p* circles of radius $R \geq RD$ in the location space. The coordinates of the center of each circle were chosen according to a uniform distribution. For each set *s* (*s=1..4*), we assigned the same demand value w_s to the points internal to each circle. In particular we fixed $w_1=1$, $w_2=1/2$, $w_3=1/4$, $w_4=1/8$. Points belonging to the intersection of more circles were given the maximum demand value. This way the optimal solution of the *p*-Maximal Covering problem on such instances is known in advance, as it can be obtained locating the *p* facilities exactly in the centre of the *p* circles with unitary demand values.

For some combination of values (*R*,*RD*) and for each value of *p* varying from *1* to *10*, we generated *5* different instances. For each instance, the procedure was run *10* times.
The results indicate the frequency with which the optimal solution is found (calculated as [*number of times*]/*50*) in the case *R=RD* and in the case *R=2RD* (Table 5.3).
As the procedure always finds the optimal solution, the average running times to find the optimal solution are reported for each value of *p*. The *R* value does not seem to affect the results in terms of final solution but there is a slight variation in the computational times.
However, the results appear interesting and the optimal solutions are detected in limited computing times.

5.5.3 Solving Minimum Variance problem instances

The adaptation of the ABM framework to the Single Facility Minimum Variance like problem was applied to some instances contained in Drezner and Drezner (2007). These instances consider a continuous location space and a discrete demand space with demand points of equal demand values distributed on the Euclidean plane.
In order to solve the instances we used a *100x100* grid of agents, i.e. *10000* passive agents. As, in general, the position of an original discrete demand point did not coincide with any grid points, an adaptation of the instances demand data was performed, associating each demand point to the closest grid point; thus, each grid point has been weighted with a demand value equal to the number of associated demand points. The ABM provides the coordinates of the facility to be located with a ten-digit precision in the continuous location space. Then, the objective function value was computed as the variance of the distances of the original demand points from the located facility. This way, the objective function value includes the effects of the aggregation operation and, thus, associated errors (Plastria, 2000).
Table 5.4 shows the capability of the proposed approach to find good results in reasonable computational times.

p	Optimal Solution	ABM Best Solution Gap	ABM Runtime (*sec*)
2	14427.593010	0.00%	0.50
3	12095.442160	0.00%	1.50
4	10661.476590	0.00%	1.50
5	9715.627471	0.10%	2.30
6	8787.556817	0.23%	4.10
7	8160.320284	0.53%	4.20
8	7564.294907	0.22%	5.40
9	7088.128333	0.38%	6.00
10	6705.035556	1.32%	6.00

Table 5.2: Computational Results on the *Bongartz287* instance

	R=RD		*R=2RD*	
p	Optimal Solution Frequency	Average Runtime (*sec*)	Optimal Solution Frequency	Average Runtime (*sec*)
1	100.00%	2.1	100.00%	2.1
2	100.00%	3.2	100.00%	3.4
3	100.00%	4.0	100.00%	4.3
4	100.00%	5.4	100.00%	6.1
5	100.00%	7.2	100.00%	8.3
6	100.00%	10.4	100.00%	11.6
7	100.00%	14.9	100.00%	17.9
8	100.00%	18.3	100.00%	20.8
9	100.00%	22.2	100.00%	23.4
10	100.00%	26.3	100.00%	30.5

Table 5.3: Computational results on the *p*-Maximal Covering randomly generated instances.

Demand Points	Optimal Solution	ABM Best Solution Gap	ABM Runtime (*sec*)
2000	0.0204669774	0.96%	3.1
5000	0.0203239336	0.08%	3.1
10000	0.0205132773	0.19%	4.3

Table 5.4: Computational results on the instances in Drezner and Drezner (2007) for the Single Facility Minimum Variance problem

5.6 Extension of the framework to other classes of Location Problems

Apart from the above considered Location Problems for which the MAS-based approach has been successfully tested against other solution algorithms, several extensions to other classes of problems (introduced in Chapter 4) have been produced, though no computational results are available at the moment.

5.6.1 The Anti-p-Median like problem

As introduced in Chapter 4, the anti-*p*-Median location problem deals with the necessity of locating *p* obnoxious facilities in such a way to minimize the damage they can do to a set of users distributed on an Euclidean plane.

The MAS-based framework can be easily adapted to this problem taking into account the following simple considerations. As the anti-*p*-Median location problem is the *repulsive* counterpart of the *p*-Median one, facilities (represented by active agents in the framework) will be not attracted by demand points, represented as passive agents within the MAS-based framework. On the contrary, active agents will be "pushed" away by demand points: the higher the demand concentration of a demand point, the strongest the force that will push the facility away.

Thus, utilizing the above introduced notation, the demand-driven force will be expressed in the following way:

$$\underline{F}_{ia}^{d} = -\frac{w_i}{\| \underline{d}_{ia} \|} \frac{1}{\| \underline{d}_{ia} \|} \qquad \forall i \in P_{rd,a}$$

where, again, w_i represents the demand associated to *i* and $\underline{d}_{\text{ia}}$ the above distance vector. In practice, we suppose that the influence of a demand point *i* on the facility *a* increases the closer the facility moves towards such a demand point. The demand-driven force becomes zero when the distance between the two agents become infinitive.

As regards the distributive force, it can be assumed equal to the one defined for the *p*-Median like case.

5.6.2 Location Problems with forbidden regions

A forbidden region represents an obstacle to the placement of facilities within a location space. Forbidden regions can be represented by natural or artificial obstacles. For instance, forbidden regions can represent lakes, mountains, national parks. It is worth to note that forbidden regions can also express a demand for the service to be located, but cannot host facilities.

The flexibility of the proposed MAS-based approach allows modeling forbidden regions endowed with a circular shape in a simple and intuitive way.

Indeed, a forbidden region can be reproduced by introducing a new class of agents that has to be placed within the location space. Within a given neighborhood coinciding with the forbidden region (whose radius has to be specified as a parameter), these agents (located at the center of the forbidden region) operate a repulsive force (similar to the one introduced among facilities for the other classes of location problems) that allows keeping away facilities from the neighborhood itself (Figure 5.5). The agents representing the center of the forbidden region can be implemented in any of the location problems previously introduced, independently on the specific objective function.

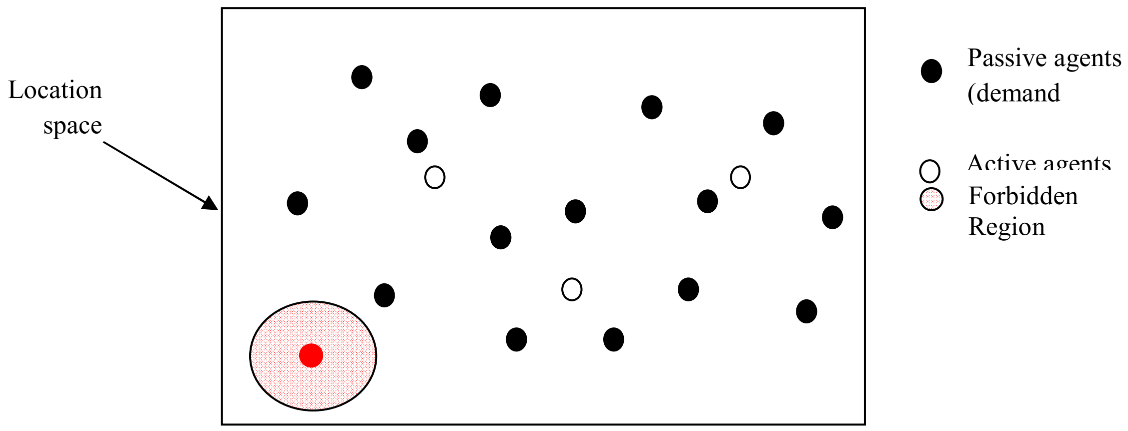


Figure 5.5 – Location space and agent categories

5.6.3 Location Problems with Existing Facilities

Another very common situation in every day real world practice is represented by the presence in the location space of a-priori located facilities. Location Problems presenting this additional characteristic are referred as Existing Facilities Location Problems.

In practice, these problems seek for locating p facilities respecting the additional constraint that a certain number of k facilities are already located.

The MAS-based framework allows simply representing also this case. In practice, providing the number and the coordinates of existing facilities, the MAS generates agents representing them. These agents will operate a repulsive force (similar to the one introduced among facilities to be located for the other classes of location problems) on the active agents representing facilities to be located. Indeed, as already stated, the presence of distributive forces allows avoiding facilities overlapping that could yield bad quality objective function values.

5.7 GIS extensions

Typically, real-life facility location problems can be far more complex than the ones illustrated in theoretical OR literature. Indeed, there are many other variables to be considered (e.g. availability of suitable sites, cost of sites, size of facility, access to and from sites, regulatory issues, planning controls, availability of suitable labor, timing of developments, etc.) and all of these considerations exist within a dynamic environment that affects these and core variables such as customer demand patterns, materials supply and changes in the technological, commercial and political environment.

Thus, a straightforward data import process and an interactive visualization and contextualization of the output and parameter setting of the optimization process is therefore a relevant requirement for a decision support system capable of helping decision makers in the field of location analysis.

Geographic Information Systems (GISs) can be certainly cited as the premiere tool useful in dealing with the analysis of spatial related phenomena. A GIS is a set of computerized tools, including both hardware and software, for collecting, storing, retrieving, transforming and displaying spatial data. GISs are essentially a combination among computerized mapping and data base management systems. Anything that can appear on a map can be encoded into a computer and then compared to anything on any other map, using longitude-latitude coordinates.

In real world location problems, mainly when a geographical database is available, the starting phase of a location decisional process is often the representation and the analysis of the problem by means of maps, datasets and geo-statistical analysis tools of GIS software. GIS may be considered as a support tool for planners using to make decision in site and facility selections at strategic or planning stage in a supply chain. The unique capabilities of GIS make it outstandingly useful as an analysis and visualizing tool because it allows capture both spatial elements and geographical locations of facilities (Raicu et al., 2002). In the literature, several applications have been developed that utilize GIS as a decision tool to support logistic decisions about facility sites (see, for instance, Valchopoulou et al., 2001). GIS is employed to display results of ranked candidate sites in different scenarios for users to select the best sites.

However, while GISs are regularly used to build complex and interesting spatial models that clearly represent the pattern of a phenomenon, these models tend to be static models. Moreover, the majority of GISs packages do not provide capabilities to solve location problems directly, although many offer basic operations, such as locating a weighted mean center in the plane. Attempts to enhance GISs capabilities in order to make them more suitable to different classes of location problems, by integrating them with different kind of algorithms have been developed in the last decade (see, for instance, Bender et al., 2001).

On the other hand, Multi-Agent Systems (MASs) have proved useful in a number of fields. As it has widely shown in this work, applications to decision support systems for optimization problems are a cutting edge application field for MASs, thanks to their ability to deal with complex problems utilizing a decentralized approach. Given the characteristics of both the tools, a wide community of scholars is working at integrating GISs and MASs for obtaining flexible approaches for spatial-related problems (see, for instance, Brown *et al.* (2005), Parker (2005) and Guo *et al.* (2008); further references can be obtained at the website http://gisagents.blogspot.com/).

In this work, starting from the need for flexible tools for coping with real-life location problems, and acknowledging the presence of a wide interest in the literature for GISs based decision support system for location problems, an extension of the above illustrated MASs-based framework was also developed.

In the developed approach, through a procedure implemented in *NetLogo*, it is possible to import from a GIS package (for example, *Microsoft MapPoint*) a map reporting parameters related to a given territory (for example, demand for a given product, or population density); in the visual representation, a different value of the parameter is associated with each color reported on the map (Figure 5.6). Practically, the GIS input can serve as demand space for a location problem. In order to make the GIS input employable by the MAS-based framework that has been shown above, the developed procedure (available in Appendix 1) applies an *agentification* of the imported GIS demand space, as shown in Figure 5.7.

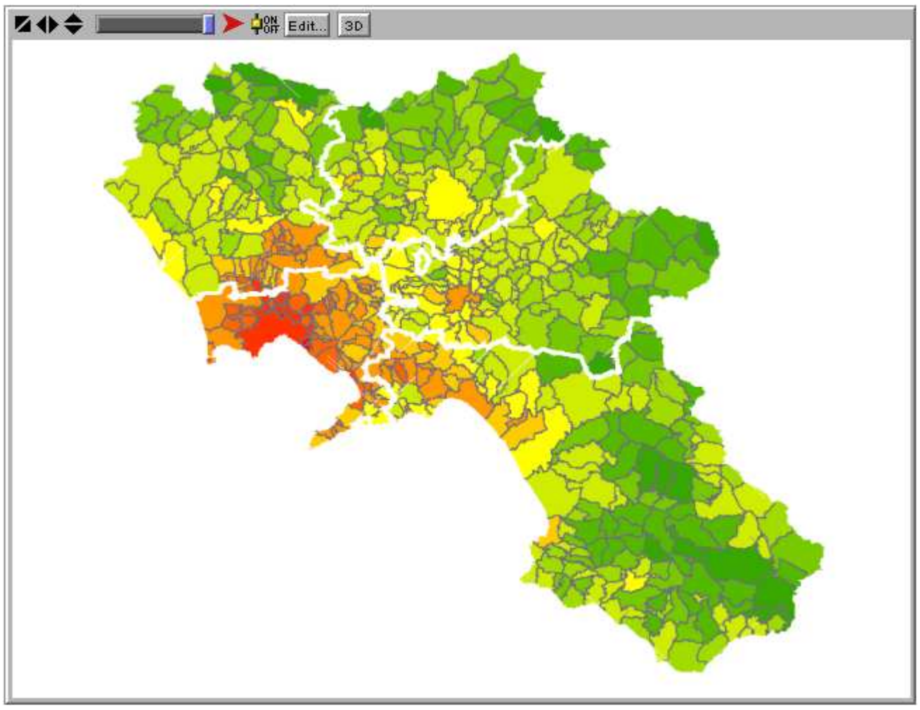


Figure 5.6 - GIS demand space

Practically, a grid schema is overlaid to the GIS map; in this way, the GIS input is partitioned in a finite set of squared portions; to each squared portion, a passive agent is assigned, whose demand value is derived from the map.

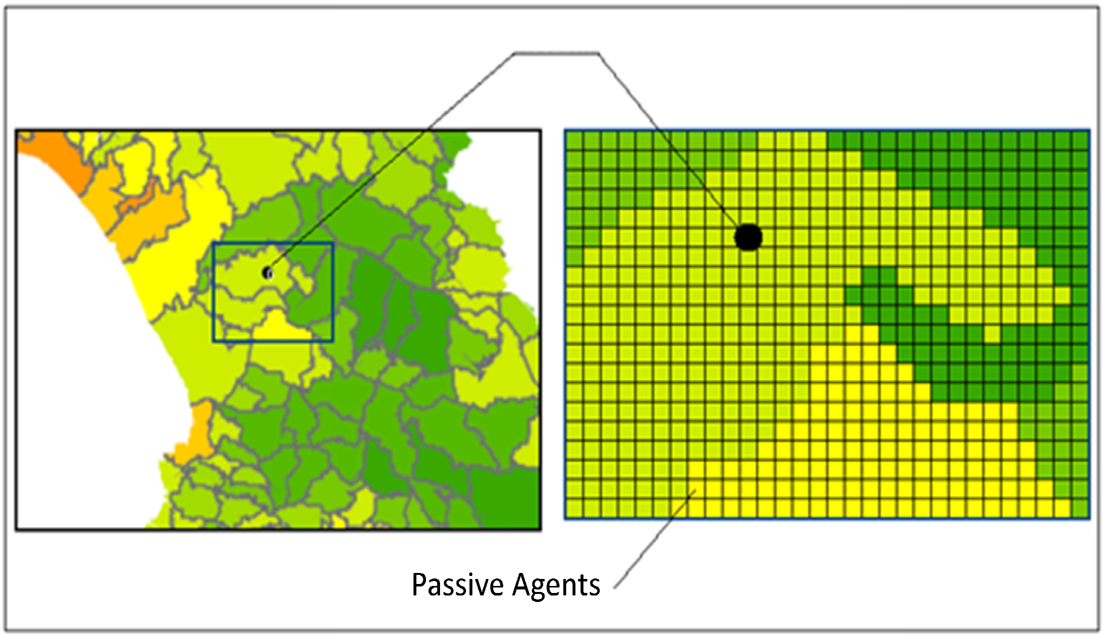


Figure 5.7 - Agentification of the GIS demand space

Obviously, to avoid ambiguity in the assignment of demand values to passive agents, it is possible to increase the resolution of the grid, thus increasing the number of passive agents needed to represent the demand space (Figure 5.8). This, of course, can turn out in a more complex location problem to be solved.

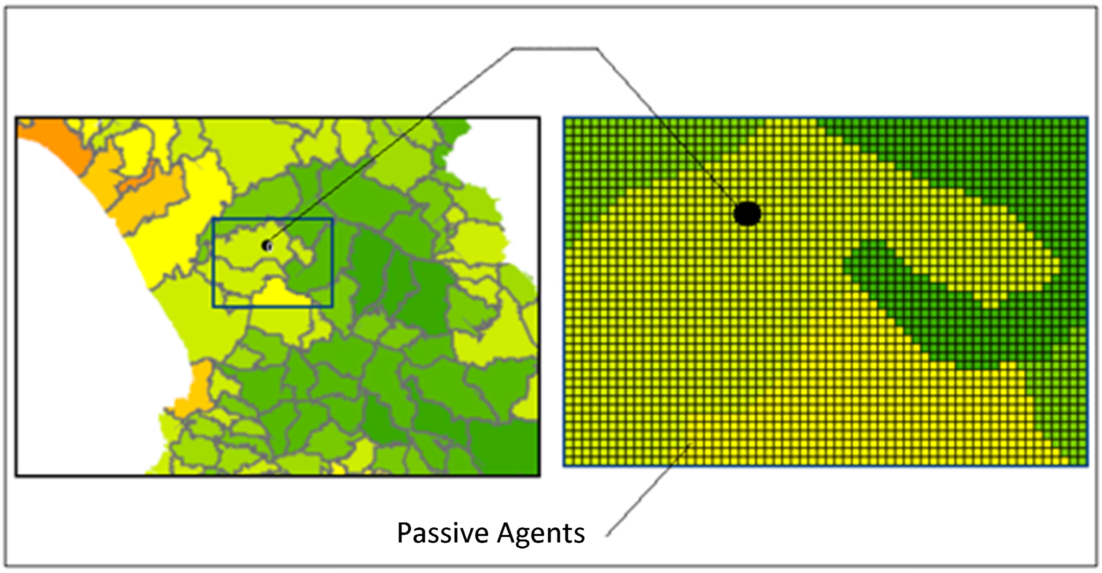


Figure 5.8 - Refinement of the GIS demand space agentification

After the agentification process, any kind of the above described location problems can be solved, providing valuable insights in real life situations.
Currently, these processes are simplified by the presence of a growing number of ABM toolkits that permits a direct access to GIS vector and raster datasets. For instance, the *NetLogo* platform we adopted in this work offers GIS extensions, developed thanks to the contribution of the Center for Connected Learning (CCL) and Computer-Based Modeling of the Northwestern University, Chicago, and available at the internet address http://ccl.northwestern.edu/netlogo/docs/gis.html.

5.8 Benefits of the proposed approach

The computational results presented in this chapter showed the effectiveness of the proposed ABM approach to solve the considered continuous location problems with discrete demand. Even if the approach appears competitive with other heuristics in terms of computational performances, the interest in using ABMs for location problems goes far beyond the computational efficiency.
The proposed approach can be viewed as a sort of metaheuristic in which some steps are performed through agent-based computation. Even if the approach could be implemented in a "traditional" way, the use of an ABM framework provides several additional benefits.
First of all, the current availability of open source environments for ABMs implementation (i.e. *NetLogo*, *JAS*, *SWARM*, *REPAST*) with dedicated libraries let modelling such heuristics easy to perform even for non-specialist users. The presence, within each of the cited toolkits, of integrated Graphical User Interfaces (GUIs) allows the immediate graphical representation of the elements of the problem together with a visual indication of the evolution of the solution.
These aspects could help users, even not particularly skilled in implementation aspects, in the search of adoptable practical solutions, especially in presence of constraints which cannot be easily modelled in a mathematical way, such as forbidden regions, obstacles, and minimum distance constraints among facilities.
The framework presents a significant flexibility that matches the huge variety of problems arising in the context of location studies. As previously illustrated, versions of the problems with variations in the objective function and/or constraints can be tackled through proper modifications of the elements of the framework (i.e. expressions of forces, calibration parameters) or introducing new characteristics in the paradigm of the model.

Among the benefits it should be also mentioned the possibility of an easy and effective integration of ABM tools with Geographical Information Systems (GIS), as deeply shown by Brown *et al.* (2005), Parker (2005) and Guo *et al.* (2008). As suggested above, the use of ABM tools allows users to solve problems through continuous interactions between optimization framework and GIS applications..

5.9 Conclusions

In recent years Agent-Based modeling is becoming more and more frequently used as approach for solving complex optimization problems. In this work an Agent-Based framework for modeling location problems was proposed and illustrated. The original contribution of the work consists mainly in the proposal of an approach that, compared to the other heuristic methods in the literature, is easy to implement and appears particularly suitable for the integration with GIS based data.

Moreover it presents characteristics of flexibility as the general framework can be applied, with slight modifications, to solve different kind of locations problems (i.e. *p*-Median like problem, *p*-Maximal Covering like problem, Single Facility Minimum Variance like problem). The features of the model suggest also the possibility of immediately adapting the approach to take into account constraints (for instance, minimum distance constraints among facilities, presence of obstacles or forbidden areas in the location space) whose formulation makes the problem hard to solve using mathematical programming based methods.

The preliminary computational experiences appear encouraging and indicate that the approach provides reasonable quality solutions within limited running times.

Future researches will include an extensive computational experimentation to test the scalability of the proposed ABM approach on very large scale instances of the considered problems. Moreover, it will be studied the adaptation of the framework to other classes of location problems such as the anti-*p*-Median problem (Erkut and Neuman (1989), Cappanera *et al.* (2003)) and the *p*-Minimal Covering problem with distance constraints (Berman and Huang (2008)).

Conclusions

The daily work of professionals involves making a series of *decisions*. In fact, the world relies on systems designed by engineers and business people. Thus, the quality of decisions made by these two categories of professionals is of critical importance.

Decisions are made by looking at the relevant data and making judgments. Making decisions on issues with important consequences has become a highly complex problem due to the many competing forces under which the world is operating today. Anyone who holds a technical, managerial, or administrative job these days is faced with making decisions daily at work and, thus, is called to solve Decision Making Problems. Today it is essential to make decisions on a rational basis: the most rational way for solving decision making problems is through quantitative analysis.

For a long time, classical optimization techniques (exact methods, heuristic methods) have represented the only available approach to solve different types of decision-making problem, both at strategic and tactical levels.

In the last decade, agent-based computing has been suggested as a promising technique for problem whose domains are distributed, complex and heterogeneous (Weiss, 1999; Wooldridge, 2002), also thanks to the availability of many commercial and open source codes including graphical interfaces for the elements of the problem. Application to several classes of optimization problems, ranging from scheduling and supply chain planning to routing, have been developed, as shown in Chapter 3.

After having verified the presence, in the literature, of a broad set of agent-based approaches for optimization problems, in this work an Agent-Based framework for modeling location problems was proposed and illustrated. The original contribution of the work consists mainly in the proposal of an approach that, compared to the other heuristic methods in the literature, is easy to implement and appears particularly suitable for the integration with GIS based data.

Moreover it presents characteristics of flexibility as the general framework can be applied, with slight modifications, to solve different kind of locations problems (i.e. p-Median like problem, p-Maximal Covering like problem, Single Facility Minimum Variance like problem). The features of the model suggest also the possibility of immediately adapting the approach to take into account constraints (for instance, minimum distance constraints among facilities, presence of obstacles or forbidden areas in the location space) whose formulation makes the problem hard to solve using mathematical programming based methods.

The preliminary computational experiences appear encouraging and indicate that the approach provides reasonable quality solutions within limited running times.
Future researches will include an extensive computational experimentation to test the scalability of the proposed ABM approach on very large scale instances of the considered problems. Moreover, it will be studied the adaptation of the framework to other classes of location problems such as the anti-*p*-Median problem (Erkut and Neuman (1989), Cappanera *et al.* (2003)) and the *p*-Minimal Covering problem with distance constraints (Berman and Huang (2008)).

Appendix I
The NetLogo platform

Generalities

NetLogo is an open-source software platform that allows modeling complex and dynamics systems. It has been developed at the Center for Connected Learning and Computer-Based Modeling at Northwestern University. It is a continuously evolving project supported by a wide users and developers community.

NetLogo architecture is based on a Java platform. Thus, it can be run on any compatible architecture. NetLogo fundamental characteristics is an intuitive programming language that allows even the non-specialist user implementing models. This programming is inspired to the well-famous Logo; NetLogo codes are then translated into Java ones through a compiler embedded in the architecture.

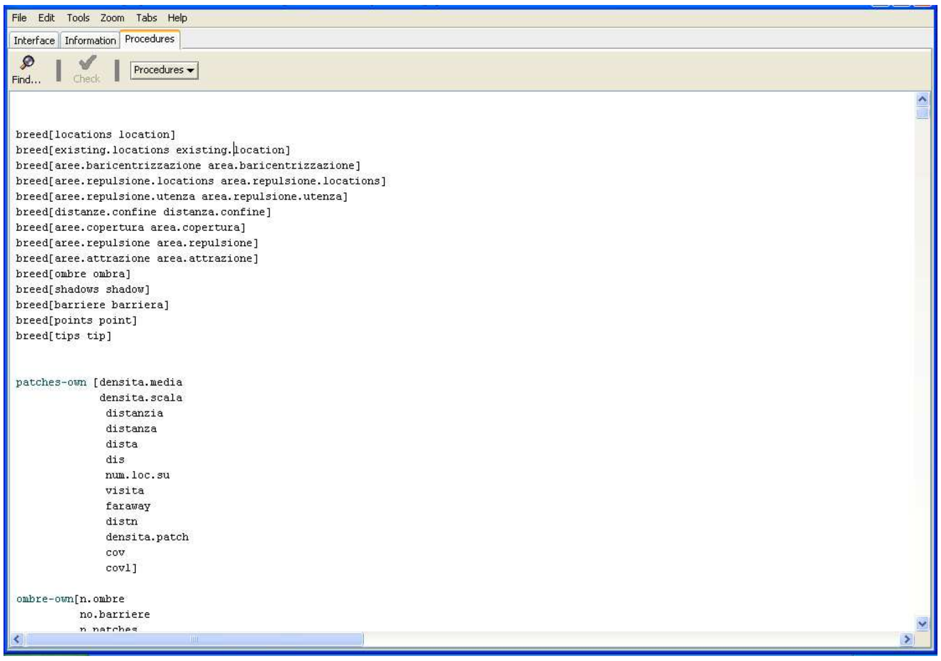
Within the simulation environment it is possible to implement three main agents categories:

- *Turtles*, representing proactive and reactive agents, as they perform evolutive actions among themselves and with the environment. It is also possible to implement different turtles categories (or classes), each endowed with particular characteristics.
- *Patches*, modeling passive agents. They represent square sections of a continuous and bi-dimensional space; they are identified by means of specific coordinates.
- *Observer* representing a mediator architecture that inizializes active agents and assigns them tasks.

NetLogo allows creating agentsets NetLogo that are aimed at defining classes of agents sharing the same characteristics and behaviors. An agentset can be made up of a set of turtles or patches, and it is identified in NetLogo code by using the primitive *breed*.

The NetLogo development kit presents three main fields:

- *Procedures* (Figure I.1) in which commands describing agents behavior can be specified, by utilizing NetLogo primitives;
- *Interface* (Figure I.2) where the *world* in which the simulation takes place can be observed. *Plots* and *monitors* can be inserted, in order to control the evolution of relevant variables. *Sliders* and *switches* are designed to change parameter values without modifying the underlying code;
- *Information* (Figure I.3) in which the main characteristic of the developed model can be explained.


File Edit Tools Zoom Tabs Help
Interface Information Procedures
Find...
Check
Procedures

```
breed[locations location]
breed[existing.locations existing.location]
breed[aree.baricentrizzazione area.baricentrizzazione]
breed[aree.repulsione.locations area.repulsione.locations]
breed[aree.repulsione.utenza area.repulsione.utenza]
breed[distanze.confine distanza.confine]
breed[aree.copertura area.copertura]
breed[aree.repulsione area.repulsione]
breed[aree.attrazione area.attrazione]
breed[ombre ombra]
breed[shadows shadow]
breed[barriere barriera]
breed[points point]
breed[tips tip]


patches-own [densita.media
             densita.scala
              distanzia
              distanza
              dista
              dis
              num.loc.su
              visita
              faraway
              distn
              densita.patch
              cov
              cov1]

ombre-own[n.ombre
          no.barriere
          n.patches
```

Figure I.1 - The procedures field

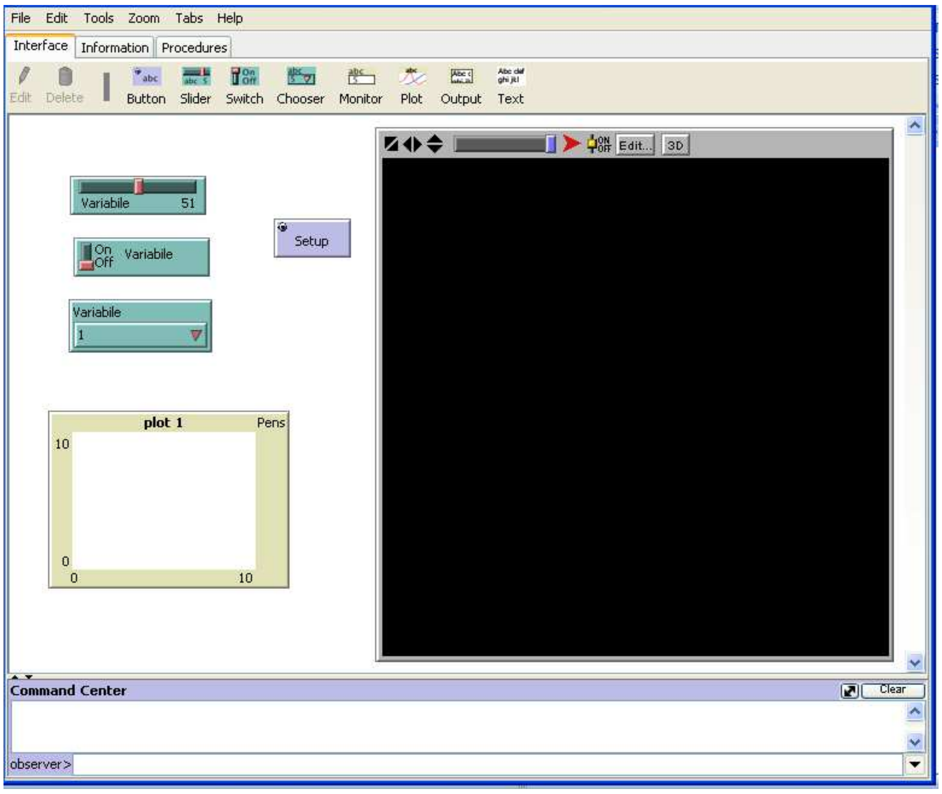
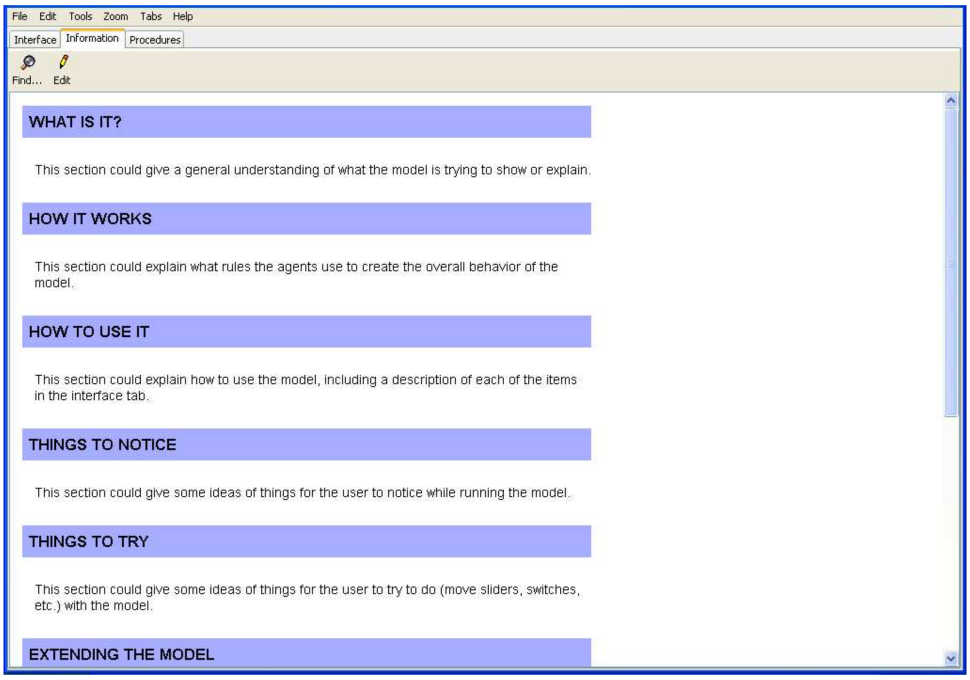

File Edit Tools Zoom Tabs Help
Interface Information Procedures
Edit Delete
Button Slider Switch Chooser Monitor Plot Output Text
Variabile 51
On Off Variabile
Setup
Variabile
1
plot 1
Pens
10
0
0
10
Edit...
3D
Command Center
Clear
observer>

Figure I.2 - The interface field



WHAT IS IT?

This section could give a general understanding of what the model is trying to show or explain.

HOW IT WORKS

This section could explain what rules the agents use to create the overall behavior of the model.

HOW TO USE IT

This section could explain how to use the model, including a description of each of the items in the interface tab.

THINGS TO NOTICE

This section could give some ideas of things for the user to notice while running the model.

THINGS TO TRY

This section could give some ideas of things for the user to try to do (move sliders, switches, etc.) with the model.

EXTENDING THE MODEL

Figure I.3 - The information field

NetLogo also provides the user with a broad library of already developed models that can be used as starting point for new modeling attempts. The *BehaviorSpace* tool is aimed at the development of experimental plans for the validation of the models.

Appendix 2

NetLogo codes of implemented ABM for Location Problems

Introduction

In the following section, the codes of implemented ABM for Location Problems (described in Chapter 5) are listed. All the codes are written in NetLogo programming language.

P-Median model code

```
breed[locations location]
breed[existing.locations existing.location]
breed[aree.baricentrizzazione area.baricentrizzazione]
breed[aree.repulsione.locations area.repulsione.locations]
breed[aree.repulsione.utenza area.repulsione.utenza]
breed[distanze.confine distanza.confine]
breed[aree.copertura area.copertura]
breed[aree.repulsione area.repulsione]
breed[aree.attrazione area.attrazione]
breed[ombre ombra]
breed[shadows shadow]
breed[barriere barriera]
breed[points point]
breed[tips tip]

patches-own [densita.media
            densita.scala
             distanzia
             distanza
             dista
             dis
             num.loc.su
             visita
             faraway
             distn
             densita.patch
             cov
             cov1]

ombre-own[n.ombre
        no.barriere
        n.patches
        no.ombre
        n.ex]

existing.locations-own[val.e]

tips-own [Tx
        Ty
        Ux
        Uy
        direction
```

```
         step]

locations-own[Fx
             Fy
             Rx
             Ry
             Bx
             By
             direzione
             passo
             moto
             val
             val2
             val.b
             num.pat
             vicini ]

globals[filename
        fitness
        contatore
        iterazioni
        non.migl
        corrente
        globale
        locale
        tip.corrente
        velocita.corrente
        velocita.locations
        velocita.tips
        barriere.vicine
        number.barriere
        number.locations
        number.ombre
        number.ombre2
        number.existing
        number.existing.ombre
        d
        covering.loc%
        covering.el%
        fac
        pat1
        ritorna
        total
        differenziazioni
        number.patches
        num.pat.loc
        totale.scala
        k
        vicini.glob ]

to startup
 ca
 let known-paths
 [ "./"
   "./models/"
   "./images/"
   "../models/"
   "../images/" ]
 let basename "north40thmap.png"
 let paths-to-try length known-paths
 set filename false
 let index 0
 while [ index < paths-to-try  ]
 [ if file-exists? (word (item index known-paths) basename)
   [ set filename (word (item index known-paths) basename)
     set index paths-to-try   ]
   set index index + 1 ]
 if filename = false
 [ set filename user-file ]
 if filename = false
 [ stop ]
 import-pcolors filename
 ask patches [
 if pcolor = black [set pcolor white]]
 migliora
end

to migliora
  import-drawing filename
end

to setup
clear-turtles
clear-plot
if mappa = "avellino" [ask patches[
if (pcolor <= 47.5)  and (pcolor >= 46.5)   [set densita.media  2]
if (pcolor <= 69.5)  and (pcolor >= 67.0)   [set densita.media 10]
if (pcolor <= 99.0)  and (pcolor >= 97.0)   [set densita.media 15]
if (pcolor <= 79.0)  and (pcolor >= 77.0)   [set densita.media 15]
if (pcolor <= 76.9)  and (pcolor > 74.0)    [set densita.media 20]
if (pcolor <= 85.0)  and (pcolor > 83.0)   [set densita.media 25]
if (pcolor <= 83.0) and (pcolor >= 81.0)  [set densita.media 50]
if (pcolor <= 73.5)  and (pcolor >= 72.0)   [set densita.media 50]
if (pcolor <= 7.0)  and (pcolor >= 4.0)   [set densita.media 30]
if (pcolor <= 3.8)  and (pcolor >= 2.4)   [set densita.media 40]
if pcolor = white                          [set densita.media 0.0]
set densita.scala 1]]
if mappa = "campania" [ask patches[
if (pcolor <= 90.0)  and (pcolor >= 80.0)   [set densita.media 0.0]
if (pcolor <= 66.9)  and (pcolor >= 63.0)   [set densita.media 2.8]
if (pcolor <= 56.9)  and (pcolor >= 54.0)   [set densita.media 3.8]
if (pcolor <= 44.4)  and (pcolor >= 42.0)   [set densita.media 7.6]
if (pcolor <= 46.9)  and (pcolor > 44.4)    [set densita.media 27.6]
if (pcolor <= 28.5)  and (pcolor >= 24.0)   [set densita.media 125.1]
if (pcolor <= 133.5) and (pcolor >= 132.5)  [set densita.media 760.3]
if (pcolor <= 16.9)  and (pcolor >= 14.0)   [set densita.media 760.3]
if pcolor = white [set densita.media 0]
if (pcolor <= 7.5)  and (pcolor >= 7.0)  [set densita.media 0]
set densita.scala 1]]
if mappa = "sicilia" [ask patches[
if (pcolor <= 47.5)  and (pcolor >= 46.5)   [set densita.media 10]
if (pcolor <= 19.4)  and (pcolor >= 19.0)    [set densita.media 20]
if (pcolor <= 18.9)  and (pcolor >= 18.6)   [set densita.media 35]
if (pcolor <= 18.5) and (pcolor >= 18.1)  [set densita.media 75]
if (pcolor <= 18.0)  and (pcolor >= 17.4)   [set densita.media 100]
if (pcolor <= 17.3)  and (pcolor >= 16.5)   [set densita.media 200]
if (pcolor <= 16.4)  and (pcolor >= 16.1)   [set densita.media 300]
if (pcolor <= 16.0)  and (pcolor >= 15.5)   [set densita.media 750]
if (pcolor <= 99.0)  and (pcolor >= 98.5)   [set densita.media 0.0]
if pcolor = white    [set densita.media 0.0]
set densita.scala 1]]
if mappa = "sardegna" [ask patches[
if (pcolor <= 47.5)  and (pcolor >= 46.5)   [set densita.media 10]
if (pcolor <= 19.4)  and (pcolor >= 19.0)    [set densita.media 25]
if (pcolor <= 18.9)  and (pcolor >= 18.6)   [set densita.media 50]
if (pcolor <= 18.5) and (pcolor >= 18.1)  [set densita.media 75]
if (pcolor <= 18.0)  and (pcolor >= 17.4)   [set densita.media 100]
if (pcolor <= 17.3)  and (pcolor >= 16.9)   [set densita.media 150]
if (pcolor <= 16.8)  and (pcolor >= 16.4)   [set densita.media 200]
if (pcolor <= 16.3)  and (pcolor >= 15.7)   [set densita.media 500]
if (pcolor <= 99.0)  and (pcolor >= 98.5)   [set densita.media 0.0]
if pcolor = white [set densita.media 0.0]
set densita.scala 1]]
if mappa = "basilicata" [ask patches[
if (pcolor <= 47.0)  and (pcolor >= 46.0)   [set densita.media 10]
if (pcolor <= 58)  and (pcolor >= 57.5)    [set densita.media 20]
if (pcolor <= 57.4)  and (pcolor >= 57)    [set densita.media 35]
if (pcolor <= 6.5)  and (pcolor >= 5.5)   [set densita.media 50]
if (pcolor <= 5.4) and (pcolor >= 5)  [set densita.media 75]
if (pcolor <= 106)  and (pcolor >= 105.5)   [set densita.media 150]
if (pcolor <= 99.0)  and (pcolor >= 98.5)   [set densita.media 0]
if (pcolor <= 7.5)  and (pcolor >= 7.0)  [set densita.media 0]
set densita.scala 1]]
if mappa = "nuova.mappa" [ask patches[
if (pcolor <= a.01)  and (pcolor >= da.01)   [set densita.media densita.01]
if (pcolor <= a.02)  and (pcolor >= da.02)    [set densita.media densita.02]
if (pcolor <= a.03)  and (pcolor >= da.03)    [set densita.media densita.03]
if (pcolor <= a.04)  and (pcolor >= da.04)   [set densita.media densita.04]
if (pcolor <= a.05) and (pcolor >= da.05)  [set densita.media densita.05]
if (pcolor <= a.06)  and (pcolor >= da.06)   [set densita.media densita.06]
if (pcolor <= a.07)  and (pcolor >= da.07)   [set densita.media densita.07]
if (pcolor <= a.08)  and (pcolor >= da.08)  [set densita.media densita.08]
if (pcolor <= a.09)  and (pcolor >= da.09)    [set densita.media densita.09]
if (pcolor <= a.10)  and (pcolor >= da.10)   [set densita.media densita.10]
if (pcolor <= a.11) and (pcolor >= da.11)  [set densita.media densita.11]
if (pcolor <= a.12)  and (pcolor >= da.12)   [set densita.media densita.12]
set densita.scala 1]]
set iterazioni 0
set contatore 0
set non.migl 0
set globale 999999999 set locale 999999999 set corrente 999999999
create-custom-points 1 [set xcor 0 set ycor 0 set shape "dot" set size 0.1 set color white]
create-custom-barriere numero.barriere [posiziona.barriere set shape "location0" set color blue set label who]
create-custom-existing.locations numero.existing.locations [posiziona.existing set shape "house" set size 1.0 set
color blue set label who]
create-custom-locations numero.locations [posiziona set shape "dot" set size 2.2 set color red set label who]
create-custom-aree.repulsione numero.locations [setxy xcor-of location (who - numero.locations) ycor-of
location (who - numero.locations) set shape "location1" set size (((distanza.repulsione) * 2)- 1)set hidden? not
hidden?]
create-custom-aree.attrazione numero.locations [setxy xcor-of location (who - (2 * numero.locations)) ycor-of
location (who - (2 * numero.locations)) set shape "location" set size (((visibilita) * 2)- 1)set hidden? not hidden?]
create-custom-ombre numero.locations [setxy xcor-of location (who - (3 * numero.locations)) ycor-of location
(who - (3 * numero.locations)) set shape "house" set size 1.0 set color white set hidden? not hidden?]
create-custom-shadows numero.locations [setxy xcor-of location (who - (4 * numero.locations)) ycor-of location
(who - (4 * numero.locations)) set shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-tips numero.locations [setxy xcor-of location (who - (5 * numero.locations)) ycor-of location
(who - (5 * numero.locations)) set shape "dot" set size 1.6 set color black ask tips [ht]]
ask aree.repulsione[__tie area.repulsione (who) location (who - numero.locations)]
ask aree.attrazione[__tie area.attrazione (who) location (who - (2 * numero.locations))]
end
```

```
to posiziona
setxy (random-xcor) (random-ycor)
if densita.media-of patch-here = 0 [posiziona]
end

to posiziona.barriere
if numero.barriere = 1 [set xcor b.xcoord set ycor b.ycoord set size (raggio.barriera * 2)]
if numero.barriere = 2 [if who = 1 [set xcor b.xcoord set ycor b.ycoord set size (raggio.barriera * 2)] if who = 2
[set xcor b.xcoord1 set ycor b.ycoord1 set size (raggio.barriera1 * 2)]]
if numero.barriere = 3 [if who = 1 [set xcor b.xcoord set ycor b.ycoord set size (raggio.barriera * 2)] if who = 2
[set xcor b.xcoord1 set ycor b.ycoord1 set size (raggio.barriera1 * 2)] if who = 3 [set xcor b.xcoord2 set ycor
b.ycoord2 set size (raggio.barriera2 * 2)]]
end

to posiziona.existing
if numero.existing.locations = 1 [set xcor xcoord set ycor ycoord]
if numero.existing.locations = 2 [if who = (numero.barriere) [set xcor xcoord set ycor ycoord] if who =
(numero.barriere + 1) [set xcor xcoord1 set ycor ycoord1]]
if numero.existing.locations = 3 [if who = (numero.barriere) [set xcor xcoord set ycor ycoord] if who =
(numero.barriere + 1) [set xcor xcoord1 set ycor ycoord1] if who = (numero.barriere + 2) [set xcor xcoord2 set
ycor ycoord2]]
if numero.existing.locations = 4 [if who = (numero.barriere) [set xcor xcoord set ycor ycoord] if who =
(numero.barriere + 1) [set xcor xcoord1 set ycor ycoord1] if who = (numero.barriere + 2) [set xcor xcoord2 set
ycor ycoord2] if who = (numero.barriere + 3) [set xcor xcoord3 set ycor ycoord3]]
if numero.existing.locations = 5 [if who = (numero.barriere) [set xcor xcoord set ycor ycoord] if who =
(numero.barriere + 1) [set xcor xcoord1 set ycor ycoord1] if who = (numero.barriere + 2) [set xcor xcoord2 set
ycor ycoord2] if who = (numero.barriere + 3) [set xcor xcoord3 set ycor ycoord3] if who = (numero.barriere + 4)
[set xcor xcoord4 set ycor ycoord4]]
end

to go
  attrai1
  attrai1.2
  respingi1
  respingi1.2
  calcola.risultante1
  calcola.risultante1.2
  muovi.tips1
  muovi1
  memorizza
  mostra.aree1
  conteggio1
  conta.barriere1
  distacca.ombre1
  distacca.shadows1
  invisibile1
  evidenzia
  plotta
set iterazioni iterazioni + 1
set contatore contatore + 1
if (iterazioni = num.iterazione) [final.set1 stop]
set totale.scala (sum values-from patches [densita.scala * faraway])
ifelse (iterazioni > 1) [set k (94576.22 / totale.scala)] [set k 2]
set corrente (sum values-from patches [densita.media * distanzia * k])
set locale (sum values-from patches [densita.media * distanza * k])
set tip.corrente (sum values-from patches [densita.media * distn * k])
set fitness (sum values-from patches [densita.media * distanzia]) / (sum values-from patches [densita.media]) * k
ask patches [ ifelse calcolo.f.obiettivo
        [let x pxcor
```

```
            let y pycor
            let distanz values-from (locations)[distancexy x y]
            let lont values-from (existing.locations)[distancexy x y]
            let tot distanz + lont
            set distanzia min tot
            let distan values-from (shadows) [distancexy x y]
            let totl distan + lont
            set distanza min totl
            let far values-from (points) [distancexy x y]
            set faraway min far
            let distnz values-from (tips) [distancexy x y]
            let distnz1 distnz + lont
            set distn min distnz1]
            [let x pxcor
            let y pycor
            let distanz values-from (locations)[distancexy x y]
            set distanzia min distanz
            let distan values-from (shadows) [distancexy x y]
            set distanza min distan
            let far values-from (points) [distancexy x y]
            set faraway min far
            let distnz values-from (tips) [distancexy x y]
            set distn min distnz]]
if diversifica[
torna
if ritorna = true[
ask locations[
posiziona]
ask shadows[posizionamento.shadows1]
ask tips[posizionamento.tips1]
set ritorna false]]
 end

to memorizza
ask patches[
set num.loc.su(num.loc.su + ((count locations-on patch pxcor pycor)))
if (num.loc.su >= num.max)
[set visita 1]]
end

to torna
ask locations[
if (visita-of patch-here = 1)
[set ritorna true]]
end

to attrai1
ask locations [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius visibilita) - 1)
   let lista.patch remove patch-here values-from patches in-radius visibilita [patch pxcor pycor]
   let lista.x map [pxcor-of ?] lista.patch
   let lista.y map [pycor-of ?] lista.patch
   let lista.densita map[(densita.media-of ?) / norma]lista.patch
   let lista.distanze map [(distance ?) / norma]lista.patch
   let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)

   set Fx precision (x.cor) 3
   set Fy precision (y.cor) 3 ]]
end

to attrai1.2
ask tips [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius visibilita) - 1)
   let lista.patch remove patch-here values-from patches in-radius visibilita [patch pxcor pycor]
   let lista.x map [pxcor-of ?] lista.patch
   let lista.y map [pycor-of ?] lista.patch
   let lista.densita map[(densita.media-of ?) / norma]lista.patch
   let lista.distanze map [(distance ?) / norma]lista.patch
   let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   set Tx precision (x.cor) 3
   set Ty precision (y.cor) 3]]
end

to respingi1
ask locations [without-interruption[
let x xcor
let y ycor
let norma count locations in-radius distanza.repulsione
let norma0 count barriere with [who = 1] in-radius raggio.barriera
let norma1 count barriere with [who = 2] in-radius raggio.barriera1
let norma2 count barriere with [who = 3] in-radius raggio.barriera2
let norma.el count existing.locations in-radius distanza.repulsione
let norma.tot (norma + norma0 + norma1 + norma2 + norma.el)
   let lista.locations remove self values-from locations in-radius distanza.repulsione [turtle who]
   let lista.barriere0 remove self values-from barriere with [who = 1] in-radius raggio.barriera [turtle who]
   let lista.barriere1 remove self values-from barriere with [who = 2] in-radius raggio.barriera1 [turtle who]
   let lista.barriere2 remove self values-from barriere with [who = 3] in-radius raggio.barriera2 [turtle who]
   let lista.existing remove self values-from existing.locations in-radius distanza.repulsione [turtle who]
   let lista.repulsione (lista.locations + lista.barriere0 + lista.barriere1 + lista.barriere2 + lista.existing)
   let lista.x map [xcor-of ?] lista.repulsione
   let lista.y map [ycor-of ?] lista.repulsione
   let lista.distanze map [(1 / (distance ?)) / norma.tot ]lista.repulsione
   let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   set Rx precision x.cor 3
   set Ry precision y.cor 3]]
end

to respingi1.2
ask tips [without-interruption[
let x xcor
let y ycor
let norma count tips in-radius distanza.repulsione
   let lista.tips remove self values-from tips in-radius distanza.repulsione [turtle who]
   let lista.x map [xcor-of ?] lista.tips
   let lista.y map [ycor-of ?] lista.tips
   let lista.distanze map [(1 / (distance ?)) / norma ]lista.tips
   let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   set Ux precision x.cor 3
   set Uy precision y.cor 3]]
end

to calcola.risultante1
```

```
ask locations[
let x.ris ((a * Fx) - ((1 - a) * Rx))
let y.ris ((a * Fy) - ((1 - a) * Ry))
ifelse x.ris != 0 or y.ris != 0 [set passo precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3][set passo 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[set direzione precision (towardsxy (x.ris + xcor) (y.ris + ycor))
3]]
 end

to calcola.risultante1.2
ask tips[
let x.ris ((a * Tx) - ((1 - a) * Ux))
let y.ris ((a * Ty) - ((1 - a) * Uy))
ifelse x.ris != 0 or y.ris != 0 [set step precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3][set step 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[set direction precision (towardsxy (x.ris + xcor) (y.ris + ycor))
3]]
 end

to conteggio1
ifelse auto.differenzia [ifelse corrente >= locale [set non.migl non.migl + 1] [set non.migl 0]] [set non.migl 0]
if (non.migl = iteraz.auto.differenzia) [differenzia1]
end

to differenzia1
set contatore 0
ask locations [posizionamento1]
ask shadows [posizionamento.shadows1]
ask tips [posizionamento.tips1]
end

to conta.barriere1
ask locations[
let norma0 count barriere with [who = 1] in-radius raggio.barriera
let norma1 count barriere with [who = 2] in-radius raggio.barriera1
let norma2 count barriere with [who = 3] in-radius raggio.barriera2
ifelse ((norma0 > 0) or (norma1 > 0) or (norma2 > 0)) [set val 1] [set val 0]]
set number.barriere sum values-from locations [val]
end

to posizionamento1
setxy (random-xcor) (random-ycor)
set non.migl 0
if densita.media-of patch-here = 0 [posizionamento1]
end

to posizionamento.shadows1
setxy xcor-of location (who - (4 * numero.locations)) ycor-of location (who - (4 * numero.locations))
end

to posizionamento.tips1
setxy xcor-of location (who - (5 * numero.locations)) ycor-of location (who - (5 * numero.locations))
end

to muovi1
if mappa = "avellino" [ask locations[ifelse pcolor = white [set heading direzione fd 0]
[ifelse gradiente [ifelse iterazioni = 0 or iterazioni = 1 [set velocita.locations speed] [set velocita.locations (speed
+ (speed * (((locale / tip.corrente) - 1) * 3)))
                    ask locations[
                    set heading direzione
                    fd passo + velocita.locations]]]
   [set velocita.locations speed
   ask locations[
   set heading direzione
   fd passo + speed ]]]]]
if mappa = "campania" [ask locations[ifelse ((pcolor <= 7.5) and (pcolor >= 7.0))  or ((pcolor <= 90.0) and
(pcolor >= 80.0))[set heading direzione fd 0]
[ifelse gradiente [ifelse iterazioni = 0 or iterazioni = 1 [set velocita.locations speed] [set velocita.locations (speed
+ (speed * (((locale / tip.corrente) - 1) * 3)))
                  ask locations[
                  set heading direzione
                  fd passo + velocita.locations]]]
   [set velocita.locations speed
   ask locations[
   set heading direzione
   fd passo + speed ]]] ]]
if ((mappa = "sicilia") or (mappa = "sardegna")) [ask locations[ifelse ((pcolor <= 99.0) and (pcolor >= 98.5))[set
heading direzione fd 0]
[ifelse gradiente [ifelse iterazioni = 0 or iterazioni = 1 [set velocita.locations speed] [set velocita.locations (speed
+ (speed * (((locale / tip.corrente) - 1) * 3)))
                  ask locations[
                  set heading direzione
                  fd passo + velocita.locations]]]
   [set velocita.locations speed
   ask locations[
   set heading direzione
   fd passo + speed ]]] ]]
if (mappa = "basilicata") [ask locations[ifelse (((pcolor <= 99.0) and (pcolor >= 98.5)) or ((pcolor <= 7) and
(pcolor >= 7.0)) or (pcolor = white)) [set heading direzione fd 0]
[ifelse gradiente [ifelse iterazioni = 0 or iterazioni = 1 [set velocita.locations speed] [set velocita.locations (speed
+ (speed * (((locale / tip.corrente) - 1) * 3)))
                  ask locations[
                  set heading direzione
                  fd passo + velocita.locations]]]
   [set velocita.locations speed
   ask locations[
   set heading direzione
   fd passo + speed ]]] ]]
if (mappa = "nuova.mappa") [ask locations[ifelse (((pcolor <= a.11)  and (pcolor >= da.11)) or ((pcolor <= a.12)
and (pcolor >= da.12))) [set heading direzione fd 0]
[ifelse gradiente [ifelse iterazioni = 0 or iterazioni = 1 [set velocita.locations speed] [set velocita.locations (speed
+ (speed * (((locale / tip.corrente) - 1) * 3)))
                  ask locations[
                  set heading direzione
                  fd passo + velocita.locations]]]
   [set velocita.locations speed
   ask locations[
   set heading direzione
   fd passo + speed ]]] ]]
end

to muovi.tips1
set velocita.tips velocita.locations * 1.5
ask tips [set heading direction
        fd step + velocita.tips]
end

to distacca.ombre1
ask ombre [
if (iterazioni > 1) and (contatore > 1) and (number.barriere = 0)[if corrente <  globale
[setxy xcor-of location (who - (3 * numero.locations)) ycor-of location (who - (3 * numero.locations)) set
globale corrente]]]
```

```
ask ombre [
let norma0 count barriere with [who = 1] in-radius raggio.barriera
let norma1 count barriere with [who = 2] in-radius raggio.barriera1
let norma2 count barriere with [who = 3] in-radius raggio.barriera2
ifelse ((norma0 > 0) or (norma1 > 0) or (norma2 > 0))[set n.ombre 1] [set n.ombre 0]
set number.ombre sum values-from ombre [n.ombre]]
ifelse number.ombre > 0 [ask ombre [die] set d 1] [set d 0]
if d = 1 [create-custom-ombre numero.locations [setxy xcor-of location (who - (3 * numero.locations)) ycor-of
location (who - (3 * numero.locations)) set shape "house" set size 1.0 set color white set hidden? not hidden?]]
end

to distacca.shadows1
ask shadows [
ifelse (corrente < locale)
[setxy xcor-of location (who - (4 * numero.locations)) ycor-of location (who - (4 * numero.locations))] [set xcor
xcor + 0.0 set ycor ycor + 0.0]]
end

to mostra.aree1
ifelse mostra.aree.influenza [ask aree.repulsione[show-turtle]ask aree.attrazione[show-turtle]]
                        [ask aree.repulsione[ht]ask aree.attrazione[ht]]
end

to invisibile1
ifelse set.opt [ask ombre [show-turtle] ask shadows [show-turtle]]
            [ask ombre [ht] ask shadows [ht]]
end

to final.set1
ask locations [die] ask shadows [die] ask aree.repulsione [die] ask aree.attrazione [die]
ask ombre [set shape "house" set size 1.3 set color violet]
ask existing.locations [set shape "house" set size 1.3 set color blue]
if ((number.barriere > 0) and (globale = 999999999)) [ask ombre [die] set globale "nd"]
end

to plotta
set-current-plot "valore f.o."
if iterazioni > 0 [plot  corrente]
end

to evidenzia
if mostra.aree.influenza = false
[  if mouse-inside? [
  let min-d min values-from locations [distancexy mouse-xcor mouse-ycor]
  let chi one-of aree.repulsione with [distancexy mouse-xcor mouse-ycor = min-d]
  let che one-of aree.attrazione with [distancexy mouse-xcor mouse-ycor = min-d]
  if chi != nobody and che != nobody
  [ask chi[show-turtle]ask che[show-turtle]] ] ]
End
```

Single Facility Minimum Variance model code

```
breed[locations location]
breed[aree.repulsion.locations area.repulsione.locations]
breed[aree.repulsione area.repulsione]
breed[aree.attrazione area.attrazione]
breed[ombre ombra]
breed[tips tip]
breed[ shadows shadow]

patches-own [densita.media
            patch-id
            distanzia
            dista
            visita
            num.loc.su
            densita.scala
            distanza
            dis
            distn
            densita.patch]

locations-own[Fx
            Fy
            Rx
            Ry
            direzione
            passo]

tips-own [Tx
         Ty
         Ux
         Uy
         direction
         step]

globals[filename
       corrente
       fitness
       globale
       locale
       tip.corrente
       velocita.corrente
       velocita.locations
       velocita.tips
       number.locations
       number.ombre
       differenziazioni
       number.patches
       non.migl
       num.pat.loc
       totale.scala
       contatore
       iterazioni
       mediana
       ritorna
       d
```

```
        best.mediana
        special-patches
        varianza
        avgdis]

to setup
ca
ask patches [set patch-id pxcor +","+ pycor]
let patch-list1[
"88,16"
"36,73"
"64,95"
"68,33"
"15,20"
"71,92"
"93,7"
"41,71"
"86,69"
"20,34"
"23,99"
"69,77"
"85,78"
"59,27"
"71,94"
"5,38"
"32,84"
"91,71"
"44,49"
"47,9"]
set special-patches patches with [member? patch-id patch-list1]
 ask special-patches [
            set densita.media 10  set pcolor 55]
ask patches[
            if pcolor = black [set pcolor white]
            if pcolor = white [set densita.media 0]]
set iterazioni 0
set contatore 0
set non.migl 0
set globale 999999999 set locale 999999999 set corrente 999999999
create-custom-locations numero.locations [posiziona set shape "dot" set size 1.0 set color black set label who]
create-custom-aree.repulsione numero.locations [setxy xcor-of location (who - numero.locations) ycor-of
location (who - numero.locations) set shape "location1" set size (((distanza.repulsione) * 2)- 1)set hidden? not
hidden?]
create-custom-aree.attrazione numero.locations [setxy xcor-of location (who - (2 * numero.locations)) ycor-of
location (who - (2 * numero.locations)) set shape "location" set size (((visibilita) * 2)- 1)set hidden? not hidden?]
create-custom-ombre numero.locations [setxy xcor-of location (who - (3 * numero.locations)) ycor-of location
(who - (3 * numero.locations)) set shape "dot" set size 0.7 set color white set hidden? not hidden?]
create-custom-shadows numero.locations [setxy xcor-of location (who - (4 * numero.locations)) ycor-of location
(who - (4 * numero.locations)) set shape "dot" set size 1.6 set color blue set hidden? not hidden?]
create-custom-tips numero.locations [setxy xcor-of location (who - (5 * numero.locations)) ycor-of location
(who - (5 * numero.locations)) set shape "dot" set size 1.6 set color black ask tips [ht]]
ask aree.repulsione[__tie area.repulsione (who) location (who - numero.locations)]
ask aree.attrazione[__tie area.attrazione (who) location (who - (2 * numero.locations))]
end

to posiziona
setxy (random-xcor) (random-ycor)
if densita.media-of patch-here = 0 [posiziona]
end
```

```
to go
  attrai1
  attrai1.2
  respingi1
  respingi1.2
  calcola.risultante1
  calcola.risultante1.2
  muovi.tips1
  muovi1
  memorizza
  mostra.aree1
  conteggio1
  distacca.ombre1
  distacca.shadows1
  invisibile1
  plotta
set iterazioni iterazioni + 1
set contatore contatore + 1
if iterazioni = num.iterazione [stop]
set corrente (sum values-from patches [densita.media * distanzia ])
set locale (sum values-from patches [densita.media * distanza ])
set tip.corrente (sum values-from patches [densita.media * distn ])
set fitness (sum values-from patches [densita.media * distanzia])/(sum values-from patches[densita.media])
ask patches
            [let x pxcor
            let y pycor
            let distanz values-from (locations)[distancexy x y]
            set distanzia min distanz
            let distan values-from (shadows) [distancexy x y]
            set distanza min distan
            let distnz values-from (tips) [distancexy x y]
            set distn min distnz]
             torna
           if ritorna = true[
           ask locations[
           setxy (random-xcor) (random-ycor)]
          ask shadows[posizionamento.shadows1]
          ask tips[posizionamento.tips1]
        set ritorna false           ]
 end

to memorizza
ask patches[
 set  num.loc.su (num.loc.su + ( (count locations-on patch pxcor pycor)))
if(  num.loc.su >= num.max )
 [set visita 1]]
end

to attrai1
ask locations [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius visibilita) - 1)
    let lista.patch remove patch-here values-from patches in-radius visibilita [patch pxcor pycor]
    let lista.x map [pxcor-of ?] lista.patch
    let lista.y map [pycor-of ?] lista.patch
    let lista.densita map[(densita.media-of ?) / norma]lista.patch
    let lista.distanze map [(distance ?) / norma]lista.patch
    let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
    let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
    set Fx precision (x.cor) 3
    set Fy precision (y.cor) 3 ]]
end

to attrai1.2
ask tips [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius visibilita) - 1)
    let lista.patch remove patch-here values-from patches in-radius visibilita [patch pxcor pycor]
    let lista.x map [pxcor-of ?] lista.patch
    let lista.y map [pycor-of ?] lista.patch
    let lista.densita map[(densita.media-of ?) / norma]lista.patch
    let lista.distanze map [(distance ?) / norma]lista.patch
    let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
    let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
    set Tx precision (x.cor) 3
    set Ty precision (y.cor) 3]]
end

to respingi1
ask locations [without-interruption[
let x xcor
let y ycor
let norma count locations in-radius distanza.repulsione
     let lista.locations remove self values-from locations in-radius distanza.repulsione [turtle who]
    let lista.x map [xcor-of ?] lista.locations
    let lista.y map [ycor-of ?] lista.locations
    let lista.distanze map [(1 / (distance ?)) / norma ]lista.locations
    let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
    let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
    set Rx precision x.cor 3
    set Ry precision y.cor 3]]
end

to respingi1.2
ask tips [without-interruption[
let x xcor
let y ycor
let norma count tips in-radius distanza.repulsione
    let lista.tips remove self values-from tips in-radius distanza.repulsione [turtle who]
    let lista.x map [xcor-of ?] lista.tips
    let lista.y map [ycor-of ?] lista.tips
    let lista.distanze map [(1 / (distance ?)) / norma ]lista.tips
    let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
    let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)

    set Ux precision x.cor 3
    set Uy precision y.cor 3]]
end

to calcola.risultante1
ask locations[
let x.ris ((a * Fx) - ((1 - a) * Rx))
let y.ris ((a * Fy) - ((1 - a) * Ry))
ifelse x.ris != 0 or y.ris != 0 [set passo precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3][set passo 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[set direzione precision (towardsxy (x.ris + xcor) (y.ris + ycor))
3]]
end
```

```
to calcola.risultante1.2
ask tips[
let x.ris ((a * Tx) - ((1 - a) * Ux))
let y.ris ((a * Ty) - ((1 - a) * Uy))
ifelse x.ris != 0 or y.ris != 0 [set step precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3][set step 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[set direction precision (towardsxy (x.ris + xcor) (y.ris + ycor))
3]]
end

to conteggio1
ifelse auto.differenzia [ifelse corrente >= locale [set non.migl non.migl + 1] [set non.migl 0]] [set non.migl 0]
if (non.migl = iteraz.auto.differenzia) [differenzia1]
end

to differenzia1
set contatore 0
ask locations [posizionamento1]
ask shadows [posizionamento.shadows1]
ask tips [posizionamento.tips1]
end

to muovi1
ask locations[
ifelse gradiente [ifelse iterazioni = 0 or iterazioni = 1 [set velocita.locations speed] [set velocita.locations (speed
+ (speed * (((locale / tip.corrente) - 1) * 3)))
                    ask locations[
                    set heading direzione
                    fd passo + velocita.locations]]]
   [set velocita.locations speed
   ask locations[
   set heading direzione
   fd passo + speed]]]
end

to muovi.tips1
set velocita.tips velocita.locations * 1.5
ask tips [set heading direction
        fd step + velocita.tips]
end

to torna
ask locations[
if(visita-of patch-here = 1)
[set ritorna true]]
end

to posizionamento1
setxy (random-xcor) (random-ycor)
set non.migl 0
if densita.media-of patch-here = 0 [posizionamento1]
end

to posizionamento.shadows1
setxy xcor-of location (who - (4 * numero.locations)) ycor-of location (who - (4 * numero.locations))
end

to posizionamento.tips1
setxy xcor-of location (who - (5 * numero.locations)) ycor-of location (who - (5 * numero.locations))
end
```

```
to distacca.ombre1
ask ombre [
if (iterazioni > 1) and (contatore > 1) [if corrente <  globale
[setxy xcor-of location (who - (3 * numero.locations)) ycor-of location (who - (3 * numero.locations)) set
globale corrente]]]
end

to distacca.shadows1
ask shadows [
ifelse (corrente < locale)
[setxy xcor-of location (who - (4 * numero.locations)) ycor-of location (who - (4 * numero.locations))] [set xcor
xcor + 0.0 set ycor ycor + 0.0]]
end

to distacca1
ask ombre [
if mediana <  best.mediana
[setxy xcor-of location (who - (3 * numero.locations)) ycor-of location (who - (3 * numero.locations))]]
end

to mostra.aree1
ifelse mostra.aree.influenza [ask aree.repulsione[show-turtle]ask aree.attrazione[show-turtle]]
                             [ask aree.repulsione[ht]ask aree.attrazione[ht]]
end

to evidenzia1
if mostra.aree.influenza = false[
  if mouse-inside? [
  let min-d min values-from locations [distancexy mouse-xcor mouse-ycor]
  let chi one-of aree.repulsione with [distancexy mouse-xcor mouse-ycor = min-d]
  let che one-of aree.attrazione with [distancexy mouse-xcor mouse-ycor = min-d]
  if chi != nobody and che != nobody
  [ask chi[show-turtle]ask che[show-turtle]]] ]
end

to invisibile1
ifelse set.opt [ask ombre [show-turtle]]
               [ask ombre [ht]]
end

to final.set1
ask locations [die] ask shadows [die] ask aree.repulsione [die] ask aree.attrazione [die]
ask ombre [set shape "house" set size 1.3 set color violet]
end

to plotta
set-current-plot "mediana"
plot  locale
if iterazioni > 0 [plot  corrente]
end
```

P-maximal covering model codes

Greedy Algorithm

```
breed[locations location]
breed[aree.repulsione area.repulsione]
breed[aree.repulsione1 area.repulsione1]
breed[aree.attrazione1 area.attrazione1]
breed[aree.attrazione area.attrazione]
breed[ombre ombra]
breed[ombre2 ombra2]
breed[shadows shadow]
breed[espls espl]
breed[shadows2 shadow2]
breed[loca loc]
breed[aree.visibilita area.visibilita]

patches-own [
 densita.media
 densita.patch
 densita.patch2
 densita.patch3
 cov
 cov2
 cov3]

loca-own[
 Kx
 Ky
 Wx
 Wy
 direzionel
 passol
 motol]

espls-own[
 Ex
 Ey
 Gx
 Gy
 direzione1
 passo1
 moto1 ]

locations-own[
 Fx
 Fy
 Rx
 Ry
 direzione
 passo
 moto ]

globals[
 filename
 iterazioni
 contatore
 copertura.aum
 C<=LBC
 contatore.parziale
 covering%
 covering%.locale
 covering%.loca
 best.covering%.loca
 best.covering%.locale
 local.covering%.loca
 local.covering%.locale
 fac
 iter.migliorative
 non.meglio
 num.espl
 num.messe
 pat
 total
 totale
 x2
 y2
 lx
 ly
 num.volte]


to startup
 ca
 let known-paths
 [ "./"
   "./models/"
   "./images/"
   "../models/"
   "../images/" ]
 let basename "north40thmap.png"
 let paths-to-try length known-paths
 set filename false
 let index 0
 while [ index < paths-to-try  ]
 [ if file-exists? (word (item index known-paths) basename)
   [ set filename (word (item index known-paths) basename)
     set index paths-to-try   ]
   set index index + 1 ]
 if filename = false
 [ set filename user-file ]
 if filename = false
 [ stop ]
migliora
 import-pcolors filename
 ask patches [
 if pcolor = black [set pcolor white]]
```

```
end

to migliora
  import-drawing filename
end

to setup
clear-turtles

ask patches[
if (pcolor <= 89.0)  and (pcolor >= 85.0)   [set densita.media 1]
if (pcolor <= 76.0)  and (pcolor >= 72.0)   [set densita.media 10]
if (pcolor <= 27.0)  and (pcolor >= 25.0)   [set densita.media 30]
if (pcolor <= 15.0)  and (pcolor >= 10.0)   [set densita.media 100]
if pcolor = white                          [set densita.media 0.0]]
set iterazioni 0
set contatore 0
set num.espl 1
set contatore.parziale 0
set C<=LBC 0
set covering% 0
set best.covering%.loca 0
set best.covering%.locale 0
set local.covering%.locale 0
set covering%.locale 0
set copertura.aum copertura
set num.messe 0
set num.volte 0

create-custom-espls num.espl[posiziona set shape "dot" set size 3.2 set color red set label who]
create-custom-aree.repulsione  num.espl[setxy  xcor-of  espl(who  -  num.espl)  ycor-of  espl(who  -  num.espl)set
shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione  num.espl[setxy  xcor-of  espl(who  -  2  *  num.espl)  ycor-of  espl  (who  -  2  *
num.espl)set shape "location" set size (((copertura)* 2 )- 1) set hidden? not hidden?]
create-custom-shadows num.espl[setxy xcor-of espl(who - 3 * num.espl)ycor-of espl (who - 3 * num.espl) set
shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre num.espl[setxy xcor-of espl(who - 4 * num.espl)ycor-of espl(who - 4 * num.espl) set shape
"dot" set size 1.2 set color green set hidden? not hidden?]
create-custom-loca num.espl[posiziona set shape "dot" set size 3.3 set color blue set label who]
create-custom-aree.visibilita num.espl[setxy xcor-of loc(who -  num.espl)ycor-of loc( who -  num.espl) set shape
"location3" set size(((copertura) * 2  )- 1) set hidden? not hidden?]
create-custom-ombre2 num.espl[setxy xcor-of loc(who - ( 2 * num.espl)) ycor-of loc(who - (2 * num.espl)) set
shape"dot"set size 1.6 set hidden? not hidden?]
create-custom-shadows2  num.espl[setxy  xcor-of  loc(who  -  3  *  num.espl)ycor-of  loc(who  -  3  *  num.espl)  set
shape "dot" set size 1.3 set color black set hidden? not hidden?]
ask aree.repulsione[__tie area.repulsione (who) espl(who - num.espl)]
ask aree.attrazione [__tie area.attrazione (who) espl(who - (2 * num.espl))]
ask aree.visibilita [__tie area.visibilita(who )loc(who -  num.espl )]
end


to go
if num.messe = numero.locations[
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
mostra.aree1
final.set
stop]
attrai
```

```
attrai1.2
respingi
calcola.risultante
calcola.risultante1.2
muovi
muovi1.2
verifica
verifica1.2
diversifica
ferma
cambia
meme
meme2
meme.ombre2
meme.ombre
copri
calcola
mostra.aree2
set contatore.parziale contatore.parziale + 1
set iterazioni iterazioni + 1
ask patch 0 0 [set total sum values-from patches in-radius 100 [densita.media]]

ask espls [without-interruption[
set pat count patches in-radius copertura
set cov sum values-from patches in-radius copertura [densita.patch2]
set covering%.locale ((( sum values-from espls [cov]) / total)* 100)]]
ask loca [without-interruption[
set pat count patches in-radius copertura
set cov2 sum values-from patches in-radius copertura [densita.patch]
set covering%.loca ((( sum values-from loca [cov2]) / total)* 100)]]
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
end

to final.set
ask locations  [set shape "house" set size 1.3 set color red]
mostra.aree1
end



to calcola
ask patches [
set densita.patch2 densita.media]
ask patches[
let num.fac count locations in-radius (copertura - 1)
ifelse num.fac > 0
[set densita.patch3 (densita.media / num.fac)]
[set densita.patch3 densita.media]]
end

to meme
ask shadows[
if iterazioni > 1[
ifelse covering%.locale > local.covering%.locale
[setxy xcor-of espl (who -( 3 * num.espl)) ycor-of espl (who -( 3 * num.espl)) set local.covering%.locale covering%.locale][set xcor xcor + 0.0  set ycor ycor + 0.0]]]
end
```

```
to meme2
ask shadows2[
if contatore.parziale > 1[
ifelse covering%.loca > local.covering%.loca
[setxy xcor-of loc (who -( 3 * num.espl)) ycor-of loc (who -( 3 * num.espl)) set local.covering%.loca
covering%.loca][set xcor xcor + 0.0  set ycor ycor + 0.0]]]
end

to posiziona
setxy (random-xcor)(random-ycor)
if densita.media-of patch-here = 0 [posiziona]
end

to attrai
ask espls[without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius copertura) - 1)
let lista.patch remove patch-here values-from patches in-radius copertura [patch pxcor pycor]
let lista.x map [pxcor-of ?] lista.patch
let lista.y map [pycor-of ?] lista.patch
let lista.densita map [(densita.patch-of ?) / norma]lista.patch
let lista.distanze map [ (distance ?) / norma]lista.patch
let x.cor sum (map[?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
let y.cor sum (map[?1 * ?2  * cos towardsxy ?3 ?4] lista.densita lista.distanze lista.x lista.y)
set Ex precision (x.cor) 3
set Ey precision (y.cor) 3]]
end

to respingi
ask espls[without-interruption[
let x xcor
let y ycor
let norma count espls in-radius distanza.repulsione
let norma1 count locations in-radius distanza.repulsione
let norma.tot (norma + norma1)
let lista.espls remove self values-from espls in-radius distanza.repulsione [turtle who]
let lista.locations  values-from locations in-radius distanza.repulsione [turtle who]
let lista.repulsi(lista.espls + lista.locations)
let lista.x map[xcor-of ?]lista.repulsi
let lista.y map[ycor-of ?]lista.repulsi
let lista.distanze map[(1 / (distance ?))/ norma.tot]lista.repulsi
let x.cor sum(map[?1 * sin towardsxy ?2 ?3 ]lista.distanze lista.x lista.y)
let y.cor sum(map[?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
set Gx precision x.cor 3
set Gy precision y.cor 3]]
end

to respingi1.2
ask loca[without-interruption[
let x xcor
let y ycor
let norma1 count locations in-radius distanza.repulsione
let lista.locations values-from locations in-radius distanza.repulsione [turtle who]
let lista.x map[xcor-of ?]lista.locations
let lista.y map[ycor-of ?]lista.locations
let lista.distanze map[(1 / (distance ?))/ norma1]lista.locations
let x.cor sum(map[?1 * sin towardsxy ?2 ?3 ]lista.distanze lista.x lista.y)
let y.cor sum(map[?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
set Wx precision x.cor 3
set Wy precision y.cor 3]]
end

to muovi
ask espls[
set heading direzione1
fd passo1 + speed]
end

to verifica
ask espls[
let locat count locations in-radius (copertura - 1)
if locat >= 1 [
posiziona
attrai
respingi
calcola.risultante
muovi
meme]]
end

to verifica1.2
ask loca[
let locat count locations in-radius (copertura - 1)
if locat >= 1[
posizionamento.loca
attrai1.2
calcola.risultante1.2
muovi1.2]]
end

to calcola.risultante
ask espls[
let x.ris((a * Ex) - ((1 - a) * (Gx)))
let y.ris((a * EY) - (( 1 - a) * (Gy )))
ifelse x.ris != 0 or y.ris != 0 [set passo1 precision (sqrt((x.ris ^ 2) +( y.ris ^ 2)))3][set passo1 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor [ set direzione1 precision (towardsxy (x.ris + xcor)(y.ris +
ycor))3]]
end

to ferma
ifelse covering%.locale  <=  local.covering%.locale [set C<=LBC C<=LBC + 1][ set C<=LBC 0]
if (C<=LBC = iteraz.auto.differenzia)[
differenzia1.2
set num.volte num.volte + 1 ]
end

to cambia
if num.volte = valore.num.volte[
ask ombre[
set x2 xcor-of ombra(who)
set y2 ycor-of ombra(who)]
ask ombre2[
set lx xcor-of ombra2(who)
set ly ycor-of ombra2(who)]
ifelse best.covering%.locale <= best.covering%.loca
[set non.meglio 1]
[set non.meglio 0]
ask espls [die]
ask aree.repulsione [die]
ask aree.attrazione [die]
ask shadows [die]
ask ombre[die]
scompare
ifelse ( non.meglio = 0)
[ricrea1]
[ricrea2]
contra]
end

to ricrea1
create-custom-locations 1 [setxy x2 y2 set shape "house" set size 1.6 set color pink set label who]
create-custom-aree.repulsione1 1[setxy xcor-of location (who - 1) ycor-of location (who - 1) set shape "location1" set size (((distanza.repulsione) * 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione1 1[setxy xcor-of location (who - 2)ycor-of location (who - 2) set shape "location" set size ((copertura * 2)- 1)set hidden? not hidden?]
ask aree.repulsione1[__tie area.repulsione1 (who) location(who - 1)]
ask aree.attrazione1 [__tie area.attrazione1 (who) location(who - (2))]
end

to contra
set num.messe num.messe + 1
if ( num.messe < numero.locations )[
ricrea
copri]
end

to mostra.aree1
ifelse mostra.aree.influenza
[ask aree.repulsione1[show-turtle]ask aree.attrazione1[show-turtle]]
[ask aree.repulsione1[ht]ask aree.attrazione1[ht]]
end

to mostra.aree2
ifelse mostra.aree.influenza
[ask aree.repulsione[show-turtle]ask aree.attrazione[show-turtle]ask aree.visibilita[show-turtle]]
[ask aree.repulsione1[ht]ask aree.attrazione1[ht]ask aree.visibilita[ht]]
end

to ricrea2
create-custom-locations 1 [setxy lx ly set shape "house" set size 1.6 set color pink set label who]
create-custom-aree.repulsione1 1[setxy xcor-of location (who - 1) ycor-of location (who - 1) set shape "location1" set size (((distanza.repulsione) * 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione1 1[setxy xcor-of location (who - 2)ycor-of location (who - 2) set shape "location" set size ((copertura * 2)- 1)set hidden? not hidden?]
ask aree.repulsione1[__tie area.repulsione1 (who ) location(who - 1)]
ask aree.attrazione1 [__tie area.attrazione1 (who ) location(who - (2))]
end

to copri
ask patches[
let locat count locations in-radius (copertura - 1)
ifelse( locat >= 1 )
[ set densita.patch 0]
[set densita.patch densita.media]]
end

to ricrea
create-custom-espls num.espl[posiziona set shape "dot" set size 3.2 set color red set label who]
```

```
create-custom-aree.repulsione num.espl[setxy xcor-of espl(who - num.espl ) ycor-of espl(who - num.espl )set
shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione num.espl[setxy xcor-of espl(who - (2 * num.espl) ) ycor-of espl (who - 2 *
num.espl )set shape "location" set size (((copertura)* 2 )- num.espl) set hidden? not hidden?]
create-custom-shadows num.espl[setxy xcor-of espl(who - (3 * num.espl) )ycor-of espl (who - (3 * num.espl) )
set shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre num.espl[setxy xcor-of espl(who - ( 4 * num.espl)) ycor-of espl(who - (4 * num.espl)) set
shape"dot"set size 1.6 set hidden? not hidden?]
create-custom-loca num.espl[posiziona set shape "dot" set size 3.3 set color blue set label who]
create-custom-aree.visibilita num.espl[setxy xcor-of loc(who -    num.espl)ycor-of loc( who -   num.espl) set
shape "location3" set size(((copertura.aum) * 2  )- 1) set hidden? not hidden?]
create-custom-ombre2 num.espl[setxy xcor-of loc(who - 2 * num.espl)ycor-of loc(who - 2 * num.espl) set shape
"dot" set size 1.2 set color green set hidden? not hidden?]
create-custom-shadows2 num.espl[setxy xcor-of loc(who - 3 * num.espl)ycor-of loc(who - 3 * num.espl) set
shape "dot" set size 1.3 set color black set hidden? not hidden?]
ask aree.repulsione[__tie area.repulsione (who ) espl(who - num.espl )]
ask aree.attrazione [__tie area.attrazione (who ) espl(who - ((2 * num.espl)))]
ask aree.visibilita [__tie area.visibilita(who )loc(who -  num.espl )]
set C<=LBC 0
set local.covering%.locale 0
set local.covering%.loca 0
set covering%.locale 0
set covering%.loca 0
set best.covering%.loca 0
set best.covering%.locale 0
set copertura.aum copertura.aum + copertura * incremento
set num.volte 0
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
end

to attrai1.2
ask loca[without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius copertura.aum) - 1)
let lista.patch remove patch-here values-from patches in-radius copertura.aum [patch pxcor pycor]
let lista.x map [pxcor-of ?] lista.patch
let lista.y map [pycor-of ?] lista.patch
let lista.densita map [(densita.patch-of ?) / norma]lista.patch
let lista.distanze map [ (distance ?) / norma]lista.patch
let x.cor sum (map[?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
let y.cor sum (map[?1 * ?2  * cos towardsxy ?3 ?4] lista.densita lista.distanze lista.x lista.y)
set Kx precision (x.cor) 3
set Ky precision (y.cor) 3]]
end

to muovi1.2
ask loca[
set heading direzionel
fd passol + speed * 1.5]
end

to calcola.risultante1.2
ask loca[
let x.ris((a * Kx) - ((1 - a ) * Wx))
let y.ris((a * KY) - ((1 - a)* Wy))
ifelse x.ris != 0 or y.ris != 0 [set passol precision (sqrt((x.ris ^ 2) +( y.ris ^ 2)))3][set passol 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor [ set direzionel precision (towardsxy (x.ris + xcor)(y.ris +
ycor))3]]
end

to scompare
ask loca[die]
ask aree.visibilita[die]
ask ombre2[die]
ask shadows2[die]
end

to meme.ombre
ask ombre[
if (contatore.parziale > 1)[
if best.covering%.locale < covering%.locale
[setxy xcor-of espl (who - (4 * num.espl))ycor-of espl(who - (4 * num.espl))set best.covering%.locale
covering%.locale]]]
end

to meme.ombre2
ask ombre2[
if (contatore.parziale > 1)[
if best.covering%.loca < covering%.loca
[setxy xcor-of loc (who - (2 * num.espl))ycor-of loc(who - (2 * num.espl))set best.covering%.loca
covering%.loca]]]
end

to diversifica
ifelse covering%.loca <= local.covering%.loca[
set iter.migliorative iter.migliorative + 1][set iter.migliorative 0]
if (iter.migliorative = iter.miglio)[differenzia]
end

to differenzia
set local.covering%.loca 0
set covering%.loca 0
ask loca[ posizionamento.loca]
ask shadows2[posizionamento.shadows2]
end


to posizionamento.loca
setxy(random-xcor)(random-ycor)
set iter.migliorative 0
if densita.media-of patch-here = 0
[posizionamento.loca]
end

to posizionamento.shadows2
setxy xcor-of loc(who - (3 * num.espl)) ycor-of loc(who - (3 * num.espl))
end

to differenzia1.2
set local.covering%.locale 0
set covering%.locale 0
ask espls[posizionamento.espls]
ask shadows[posizionamento.shadows]
set C<=LBC 0
end
```

```
to posizionamento.espls
setxy (random-xcor)(random-ycor)
if densita.media-of patch-here = 0
[posizionamento.espls]
end

to posizionamento.shadows
setxy xcor-of espl(who - (3 * num.espl)) ycor-of espl(who - (3 * num.espl))
end
```

Il codice per l’algoritmo costruttivo-migliorativo

```
breed[locations location]
breed[loca loc]
breed[espls espl]
breed[aree.repulsione area.repulsione]
breed[aree.repulsione1 area.repulsione1]
breed[aree.attrazione area.attrazione]
breed[aree.attrazione1 area.attrazione1]
breed[shadows shadow]
breed[ombre2 ombra2]
breed[shadows3 shadow3]
breed[ombre3 ombra3]


patches-own [
 densita.media
 densita.patch
 densita.patch2
 densita.patch3
 cov
 cov2
 cov3 ]

espls-own[
 Ex
 Ey
 Gx
 Gy
 direzione1
 passo1
 moto1]

loca-own[
 Sx
 Sy
 Hx
 Hy
 direzione4
 passo4
 moto4]

aree.repulsione1-own[
 vero.a]

aree.attrazione1-own[
 vero.b]

locations-own[
 Fx
 Fy
 Rx
 Ry
 direzione
 passo
 moto
 cop.sing.loc
 vero.loc]

globals[
 filename
 iterazioni
 contatore
 C<=LBC
 contatore.parziale
 covering%
 covering%.locale
 fac
 num.espl
 num.messe
 pat
 total
 totale
 x2
 y2
 x4
 y4
 best.covering%
 best.covering%.locale
 local.covering%
 local.covering%.locale
 covering%.loca
 velocita.corrente
 num.volte
 lista.ord
 lista.ord.rid
 num.giri
 x3
 y3
 bynary]

to startup
 ca
 let known-paths
 [ "./"
   "./models/"
   "./images/"
   "../models/"
   "../images/" ]
 let basename "north40thmap.png"
 let paths-to-try length known-paths
 set filename false
 let index 0
 while [ index < paths-to-try  ]
 [ if file-exists? (word (item index known-paths) basename)
   [ set filename (word (item index known-paths) basename)
     set index paths-to-try   ]
   set index index + 1 ]
 if filename = false
 [ set filename user-file ]
 if filename = false
 [ stop ]
 migliora
 import-pcolors filename
 ask patches [
 if pcolor = black [set pcolor white]]
end

to migliora
  import-drawing filename
end

to setup
clear-turtles

ask patches[
if (pcolor <= 89.0)  and (pcolor >= 85.0)   [set densita.media 1]
if (pcolor <= 76.0)  and (pcolor >= 72.0)   [set densita.media 10]
if (pcolor <= 27.0)  and (pcolor >= 25.0)   [set densita.media 30]
if (pcolor <= 15.0)  and (pcolor >= 10.0)   [set densita.media 100]
if pcolor = white                           [set densita.media 0.0]]
set iterazioni 0
set contatore 0
set num.espl 1
set contatore.parziale 0
set C<=LBC 0
set covering% 0
set local.covering% 0
set covering%.locale 0
set best.covering% 0
set local.covering%.locale 0
set best.covering%.locale 0
set num.messe 0
set num.volte 0
set num.giri 0
create-custom-espls num.espl[posiziona set shape "dot" set size 3.2 set color red set label who]
create-custom-aree.repulsione  num.espl[setxy  xcor-of  espl(who  -  num.espl)  ycor-of  espl(who  -  num.espl)set
shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione  num.espl[setxy  xcor-of  espl(who  -  2  *  num.espl)  ycor-of  espl  (who  -  2  *
num.espl)set shape "location" set size (((copertura)* 2 )- 1) set hidden? not hidden?]
create-custom-shadows num.espl[setxy xcor-of espl(who - 3 * num.espl)ycor-of espl (who - 3 * num.espl) set
shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre2 num.espl[setxy xcor-of espl(who - ( 4 * num.espl)) ycor-of espl(who - (4 * num.espl)) set
shape"dot"set size 1.6 set hidden? not hidden?]
ask aree.repulsione[__tie area.repulsione (who) espl(who - num.espl)]
ask aree.attrazione [__tie area.attrazione (who) espl(who - (2 * num.espl))]
end

to go
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
ifelse num.messe = numero.locations[
  if num.giri = 0[
    memorizza
    sposta]
  final.set1.3
  ask locations[without-interruption[
  set cov3 sum values-from patches in-radius copertura [densita.patch3]
  set covering%((( sum values-from locations [cov3]) / total * 100))  ]]
  ask loca[without-interruption[
  set cov2 sum values-from patches in-radius copertura [densita.patch]
  set covering%.loca((( sum values-from loca [cov2]) / total * 100)) ]]
  if num.giri = num.iterazioni[
    ask locations[without-interruption[
    set cov3 sum values-from patches in-radius copertura [densita.patch3]
    set covering%((( sum values-from locations [cov3]) / total * 100)) ]]
    mostra.aree1
    final.set
    stop]
  set num.giri num.giri + 1
  attrai1.3
  respingi1.3
  calcola.risultante1.3
  muovi1.3
  distanziare1.3
  distacca.ombre1.3
  distacca.shadows1.3
  conteggio1.3
  ask locations[without-interruption[
  set cov3 sum values-from patches in-radius copertura [densita.patch3]
  set covering%((( sum values-from locations [cov3]) / total * 100)) ]]]
[ attrai
  respingi
  calcola.risultante
  muovi
  verifica
  ferma
  cambia
  meme
  meme.ombre
  copri
  calcola
  mostra.aree1
  set contatore.parziale contatore.parziale + 1
  set iterazioni iterazioni + 1
  ask patch 0 0 [set total sum values-from patches in-radius 100 [densita.media]]
  ask espls [without-interruption[
  set pat count patches in-radius copertura
  set cov sum values-from patches in-radius copertura [densita.patch2]
  set covering%.locale ((( sum values-from espls [cov]) / total)* 100)]] ]
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
end

to final.set
ask locations  [set shape "house" set size 1.3 set color red]
mostra.aree1
end

to calcola
ask patches [
set densita.patch2 densita.media]
ask patches[
let num.fac count locations in-radius (copertura - 1)
ifelse num.fac > 0
  [set densita.patch3 (densita.media / num.fac)]
  [set densita.patch3 densita.media]]
end
```

```
to meme
ask shadows[
if iterazioni > 1[
  ifelse covering%.locale > local.covering%.locale
  [setxy xcor-of espl (who -( 3 * num.espl)) ycor-of espl (who -( 3 * num.espl)) set local.covering%.locale
covering%.locale]
  [set xcor xcor + 0.0  set ycor ycor + 0.0]]]
end

to posiziona
setxy (random-xcor)(random-ycor)
if densita.media-of patch-here = 0 [posiziona]
end

to attrai
ask espls[without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius copertura) - 1)
let lista.patch remove patch-here values-from patches in-radius copertura [patch pxcor pycor]
let lista.x map [pxcor-of ?] lista.patch
let lista.y map [pycor-of ?] lista.patch
let lista.densita map [(densita.patch-of ?) / norma]lista.patch
let lista.distanze map [ (distance ?) / norma]lista.patch
let x.cor sum (map[?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
let y.cor sum (map[?1 * ?2  * cos towardsxy ?3 ?4] lista.densita lista.distanze lista.x lista.y)
set Ex precision (x.cor) 3
set Ey precision (y.cor) 3]]
end

to respingi
ask espls[without-interruption[
let x xcor
let y ycor
let norma count espls in-radius distanza.repulsione
let norma1 count locations in-radius distanza.repulsione
let norma.tot (norma + norma1)
let lista.espls remove self values-from espls in-radius distanza.repulsione [turtle who]
let lista.locations  values-from locations in-radius distanza.repulsione [turtle who]
let lista.repulsi(lista.espls + lista.locations)
let lista.x map[xcor-of ?]lista.repulsi
let lista.y map[ycor-of ?]lista.repulsi
let lista.distanze map[(1 / (distance ?))/ norma.tot]lista.repulsi
let x.cor sum(map[?1 * sin towardsxy ?2 ?3 ]lista.distanze lista.x lista.y)
let y.cor sum(map[?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
set Gx precision x.cor 3
set Gy precision y.cor 3]]
 end

to muovi
ask espls[
set heading direzione1
fd passo1 + speed]
end

to verifica
ask espls[
let locat count locations in-radius (distanza.diff - 1)
if locat >= 1 [
  posiziona
  set C<=LBC 0]]
end

to calcola.risultante
ask espls[
let x.ris((a * Ex) - ((1 - a) * (Gx)))
let y.ris((a * EY) - (( 1 - a) * (Gy )))
ifelse x.ris != 0 or y.ris != 0 [set passo1 precision (sqrt((x.ris ^ 2) +( y.ris ^ 2)))3][set passo1 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor [ set direzione1 precision (towardsxy (x.ris + xcor)(y.ris +
ycor))3]]
end

to ferma
ifelse covering%.locale  <=  local.covering%.locale [set C<=LBC C<=LBC + 1][ set C<=LBC 0]
if (C<=LBC = iteraz.auto.differenzia)[
differenzia1.2
set num.volte num.volte + 1 ]
end

to cambia
if num.volte = valore.num.volte[
  ask ombre2[
  set x2 xcor-of ombra2(who)
  set y2 ycor-of ombra2(who)  ]
  ask espls [die]
  ask aree.repulsione [die]
  ask aree.attrazione [die]
  ask shadows [die]
  ask ombre2[die]
  ricrea1
  contra]
end

to ricrea1
create-custom-locations 1 [setxy x2 y2 set shape "house" set size 1.6 set color pink set label who]
create-custom-aree.repulsione1  1[setxy  xcor-of  location  (who  -  1)  ycor-of  location  (who  -  1)  set  shape
"location1" set size (((distanza.repulsione) * 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione1 1[setxy xcor-of location (who - 2)ycor-of location (who - 2) set shape "location"
set size ((copertura * 2)- 1)set hidden? not hidden?]
ask aree.attrazione1 [__tie area.attrazione1 (who) location(who - (2))]
end

to contra
set num.messe num.messe + 1
if ( num.messe < numero.locations )[
  ricrea
  copri]
end

to mostra.aree1
ifelse mostra.aree.influenza
  [ask aree.repulsione1[show-turtle]ask aree.attrazione1[show-turtle]]
  [ask aree.repulsione1[ht]ask aree.attrazione1[ht]]
end

to copri
ask patches[
let locat count locations in-radius (copertura - 1)
ifelse( locat >= 1 ) [ set densita.patch 0][set densita.patch densita.media]]
```

```
end

to ricrea
create-custom-espls num.espl[posiziona set shape "dot" set size 3.2 set color red set label who]
create-custom-aree.repulsione num.espl[setxy xcor-of espl(who - num.espl ) ycor-of espl(who - num.espl )set
shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione num.espl[setxy xcor-of espl(who - (2 * num.espl) ) ycor-of espl (who - 2 *
num.espl )set shape "location" set size (((copertura)* 2 )- num.espl) set hidden? not hidden?]
create-custom-shadows num.espl[setxy xcor-of espl(who - (3 * num.espl) )ycor-of espl (who - (3 * num.espl) )
set shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre2 num.espl[setxy xcor-of espl(who - ( 4 * num.espl)) ycor-of espl(who - (4 * num.espl)) set
shape"dot"set size 1.6 set hidden? not hidden?]
ask aree.repulsione[__tie area.repulsione (who ) espl(who - num.espl )]
ask aree.attrazione [__tie area.attrazione (who ) espl(who - ((2 * num.espl)))]
set C<=LBC 0
set local.covering%.locale 0
set covering%.locale 0
set best.covering%.locale 0
set num.volte 0
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
end

to meme.ombre
ask ombre2[
if (contatore.parziale > 1)[
  if best.covering%.locale < covering%.locale[
    setxy xcor-of espl (who - (4 * num.espl))ycor-of espl(who - (4 * num.espl))set best.covering%.locale
covering%.locale]]]
end

to differenzia1.2
set local.covering%.locale 0
set covering%.locale 0
ask espls[posizionamento.espls]
ask shadows[posizionamento.shadows]
set C<=LBC 0
end

to posizionamento.espls
setxy (random-xcor)(random-ycor)
if densita.media-of patch-here = 0
[posizionamento.espls]
end

to posizionamento.shadows
setxy xcor-of espl(who - (3 * num.espl)) ycor-of espl(who - (3 * num.espl))
end

to memorizza
ask locations[without-interruption[
set cop.sing.loc ((sum values-from patches in-radius copertura [densita.patch3] )/ total)* 100]]
set lista.ord sort-by [cop.sing.loc-of ?1 < cop.sing.loc-of ?2] locations
set lista.ord.rid sublist lista.ord 0 1
set vero.loc-of one-of lista.ord.rid 1
end

to sposta
ask locations[without-interruption[
if vero.loc-of location (who) = 1[
  set x3 xcor-of location(who)
  set y3 ycor-of location(who)
  set vero.a-of area.repulsione1 (who + 1) 1
  set vero.b-of area.attrazione1 (who + 2) 1 ]]]
create-custom-loca 1 [posiziona set shape "dot" set size 1.6 set color pink set label who]
create-custom-shadows3 1[setxy xcor-of loc(who - 1 )ycor-of loc (who - (1 ) ) set shape "dot" set size 1.6 set
color yellow set hidden? not hidden?]
create-custom-ombre3 1[setxy xcor-of loc(who - 2 ) ycor-of loc(who - (2)) set shape"dot"set size 1.6 set hidden?
not hidden?]
end

to attrai1.3
ask loca [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius copertura) - 1)
   let lista.patch remove patch-here values-from patches in-radius copertura [patch pxcor pycor]
   let lista.x map [pxcor-of ?] lista.patch
   let lista.y map [pycor-of ?] lista.patch
   let lista.densita map[(densita.patch-of ?) / norma]lista.patch
   let lista.distanze map [(distance ?) / norma]lista.patch
   let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   set Sx precision (x.cor) 3
   set Sy precision (y.cor) 3 ]]
end

to respingi1.3
ask loca [without-interruption[
let x xcor
let y ycor
let norma count locations in-radius distanza.repulsione
   let lista.locations remove self values-from loca in-radius distanza.repulsione [turtle who]
   let lista.x map [xcor-of ?] lista.locations
   let lista.y map [ycor-of ?] lista.locations

   let lista.distanze map [(1 / (distance ?)) / norma ]lista.locations
   let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   set Hx precision x.cor 3
   set Hy precision y.cor 3]]
end

to calcola.risultante1.3
ask loca[
let x.ris ((a * Sx) - ((1 - a) * Hx))
let y.ris ((a * Sy) - ((1 - a) * Hy))
ifelse x.ris != 0 or y.ris != 0
  [set passo4 precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3]
  [set passo4 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[
  set direzione4 precision (towardsxy (x.ris + xcor) (y.ris + ycor)) 3]]
end

to distanziare1.3
ask loca[without-interruption[
set fac count locations in-radius distanza.diff;]
if contatore < 2
  [if fac > 1 [differenzia1.3]]]]
```

```
end

to conteggio1.3
if num.giri > 1[
  ifelse auto.differenzia [
    ifelse covering%.loca <= local.covering%
      [set C<=LBC C<=LBC + 1]
      [set C<=LBC 0]]
    [set C<=LBC 0]
  if (C<=LBC = iteraz.auto.differenzia) [
    differenzia1.3 ]]
end

to differenzia1.3
set contatore 0
set C<=LBC 0
set local.covering% 0
ask loca [posiziona]
ask shadows3 [posizionamento.shadows1.3]
end

to posizionamento.shadows1.3
setxy xcor-of loc (who - (1)) ycor-of loc (who - (1))
end

to muovi1.3
ask loca[
set heading direzione4
fd passo4 + speed]
end

to distacca.ombre1.3
ask ombre3 [
if (num.giri > 1) [if best.covering% < covering%.loca
  [setxy xcor-of loc (who - (2)) ycor-of loc (who - (2)) set best.covering% covering%.loca]]]
end

to distacca.shadows1.3
ask shadows3 [if num.giri > 1[
ifelse covering%.loca > local.covering%
  [setxy xcor-of loc (who - (1)) ycor-of loc (who - (1)) set local.covering% covering%.loca]
  [set xcor xcor + 0.0 set ycor ycor + 0.0]]]
end

to final.set1.3
if num.giri = num.iterazioni - 1[
  ask locations[
  if vero.loc = 1
    [ifelse cop.sing.loc-of location(who) < best.covering% [
       ask ombre3[
       set x4 xcor-of ombra3(who)
       set y4 ycor-of ombra3(who)]
       set bynary 1 ]
   [set bynary 0] ] ]
  if bynary = 1 [
   create-custom-locations 1 [setxy x4 y4 set shape "house" set size 1.6 set vero.loc 0 set color pink set label
who]
   create-custom-aree.repulsione1 num.espl[setxy xcor-of location(who - num.espl) ycor-of location(who -
num.espl)set shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
```

```
  create-custom-aree.attrazione1 num.espl[setxy xcor-of location(who - 2 * num.espl) ycor-of location (who - 2
* num.espl)set shape "location" set size (((copertura)* 2 )- 1) set hidden? not hidden?]
  ask aree.repulsione1[__tie area.repulsione1 (who) location(who - num.espl)]
  ask aree.attrazione1[__tie area.attrazione1 (who) location(who - (2 * num.espl))]
  ask ombre3[die]
  ask locations[
  if vero.loc = 1[
    die ]]
 ask aree.repulsione1[
 if vero.a = 1[
  die ]]
 ask aree.attrazione1[
  if vero.b = 1[
  die ]] ]
 ask locations[without-interruption[
 set cov3 sum values-from patches in-radius copertura [densita.patch3]
 set covering%((( sum values-from locations [cov3]) / total * 100)) ]]]
if num.giri = num.iterazioni
  [ask loca[die]]
 end
```

Greedy Algorithm with two improving stages

```
breed[locations location]
breed[aree.repulsione area.repulsione]
breed[aree.repulsione1 area.repulsione1]
breed[aree.attrazione area.attrazione]
breed[aree.attrazione1 area.attrazione1]
breed[aree.repulsione12 area.repulsione12]
breed[aree.attrazione12 area.attrazione12]
breed[espls espl]
breed[shadows shadow]
breed[ombre2 ombra2]
breed[loca loc]
breed[shadows3 shadow]
breed[ombre3 ombra3]
breed[loca2 loc2]
breed[shadows32 shadow32]
breed[ombre32 ombra32]

patches-own [
 densita.media
 densita.patch
 densita.patch2
 densita.patch3
 cov
 cov2
 cov22
 cov3]

aree.repulsione1-own[
 vero.a]

aree.attrazione1-own[
 vero.b]

espls-own[
 Ex
 Ey
 Gx
 Gy
 direzione1
 passo1
 moto1]

loca-own[
 Sx
 Sy
 Hx
 Hy
 direzione4
 passo4
 moto4]

loca2-own[
 Sx2
 Sy2
 Hx2
 Hy2
 direzione42
 passo42
 moto42]

locations-own[
 Fx
 Fy
 Rx
 Ry
 direzione
 passo
 moto
 cop.sing.loc
 vero.loc
 dead]

globals[
 filename
 iterazioni
 contatore
 C<=LBC
 contatore.parziale
 fac
 num.espl
 num.messe
 pat
 total
 totale
 x2
 y2
 x3
 y3
 x4
 y4
 x42
 y42
 covering%
 covering%.locale
 best.covering%
 best.covering%.locale
 best.covering%2
 local.covering%2
 local.covering%
 local.covering%.locale
 covering%.loca
 covering%.loca2
 num.volte
 lista.ord
 lista.ord.rid
 lista.ord2
 lista.ord.rid2
 num.giri
 num.giri2
 bynary
 bynary2]

to startup
 ca
 let known-paths
 [ "./"
   "./models/"
   "./images/"
   "../models/"
   "../images/" ]
 let basename "north40thmap.png"
 let paths-to-try length known-paths
 set filename false
 let index 0
 while [ index < paths-to-try  ]
 [ if file-exists? (word (item index known-paths) basename)
   [ set filename (word (item index known-paths) basename)
     set index paths-to-try   ]
   set index index + 1 ]
 if filename = false
 [ set filename user-file ]
 if filename = false
 [ stop ]
 migliora
 import-pcolors filename
 ask patches [
 if pcolor = black [set pcolor white]]
end

to migliora
  import-drawing filename
end

to setup
clear-turtles
ask patches[
if (pcolor <= 89.0)  and (pcolor >= 85.0)   [set densita.media 1]
if (pcolor <= 76.0)  and (pcolor >= 72.0)   [set densita.media 10]
if (pcolor <= 27.0)  and (pcolor >= 25.0)   [set densita.media 30]
if (pcolor <= 15.0)  and (pcolor >= 10.0)   [set densita.media 100]
if pcolor = white                           [set densita.media 0.0]]
set iterazioni 0
set contatore 0
set num.espl 1
set contatore.parziale 0
set C<=LBC 0
set covering% 0
set best.covering% 0
set best.covering%2 0
set best.covering%.locale 0
set local.covering% 0
set local.covering%2 0
set local.covering%.locale 0
set covering%.locale 0
set num.messe 0
set num.volte 0
set num.giri 0
set num.giri2 0
create-custom-espls num.espl[posiziona set shape "dot" set size 3.2 set color red set label who]
create-custom-aree.repulsione num.espl[setxy xcor-of espl(who - num.espl) ycor-of espl(who - num.espl)set shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione num.espl[setxy xcor-of espl(who - 2 * num.espl) ycor-of espl (who - 2 * num.espl)set shape "location" set size (((copertura)* 2 )- 1) set hidden? not hidden?]
create-custom-shadows num.espl[setxy xcor-of espl(who - 3 * num.espl)ycor-of espl (who - 3 * num.espl) set shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre2 num.espl[setxy xcor-of espl(who - ( 4 * num.espl)) ycor-of espl(who - (4 * num.espl)) set shape"dot"set size 1.6 set hidden? not hidden?]
ask aree.repulsione[__tie area.repulsione (who) espl(who - num.espl)]
ask aree.attrazione [__tie area.attrazione (who) espl(who - (2 * num.espl))]
end

to go
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
continua
final.set1.3
continua2
final.set1.32
if num.giri2 = (num.iterazioni + num.iterazioni2)
  [ask locations[without-interruption[
  set cov3 sum values-from patches in-radius copertura [densita.patch3]
  set covering%((( sum values-from locations [cov3]) / total * 100)) ]]
  final.set
  stop]
if num.giri2 = (num.iterazioni + num.iterazioni2 - 1)
  [set num.giri2 num.giri2 + 1]
attrai
respingi
calcola.risultante
muovi
verifica
ferma
cambia
meme
meme.ombre
copri
calcola
mostra.aree1
set contatore.parziale contatore.parziale + 1
set iterazioni iterazioni + 1
ask patch 0 0 [set total sum values-from patches in-radius 100 [densita.media]]
ask espls [without-interruption[
set pat count patches in-radius copertura
set cov sum values-from patches in-radius copertura [densita.patch2]
set covering%.locale ((( sum values-from espls [cov]) / total)* 100)]]
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
end

to final.set
ask locations  [set shape "house" set size 1.3 set color red]
mostra.aree1
end

to calcola
ask patches [
set densita.patch2 densita.media]
ask patches[
let num.fac count locations in-radius (copertura - 1)
ifelse num.fac > 0
[set densita.patch3 (densita.media / num.fac)]
[set densita.patch3 densita.media]]
end

to meme
ask shadows[
if iterazioni > 1[
  ifelse covering%.locale > local.covering%.locale
    [setxy xcor-of espl (who -( 3 * num.espl)) ycor-of espl (who -( 3 * num.espl)) set local.covering%.locale
covering%.locale]
    [set xcor xcor + 0.0  set ycor ycor + 0.0] ]]
end

to posiziona
setxy (random-xcor)(random-ycor)
if densita.media-of patch-here = 0 [posiziona]
end

to attrai
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
  ask espls[without-interruption[
  let x xcor
  let y ycor
  let norma ((count patches in-radius copertura) - 1)
  let lista.patch remove patch-here values-from patches in-radius copertura [patch pxcor pycor]
  let lista.x map [pxcor-of ?] lista.patch
  let lista.y map [pycor-of ?] lista.patch
  let lista.densita map [(densita.patch-of ?) / norma]lista.patch
  let lista.distanze map [ (distance ?) / norma]lista.patch

  let x.cor sum (map[?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
  let y.cor sum (map[?1 * ?2  * cos towardsxy ?3 ?4] lista.densita lista.distanze lista.x lista.y)
  set Ex precision (x.cor) 3
  set Ey precision (y.cor) 3 ]]]
end

to distanziare1.32
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
  ask loca2[without-interruption[
  set fac count locations in-radius distanza.diff;]
  if contatore < 2
    [if fac > 1 [differenzia1.32]]]]]
end

to respingi
```

```
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
  ask espls[without-interruption[
  let x xcor
  let y ycor
  let norma count espls in-radius distanza.repulsione
  let norma1 count locations in-radius distanza.repulsione
  let norma.tot (norma + norma1)
  let lista.espls remove self values-from espls in-radius distanza.repulsione [turtle who]
  let lista.locations  values-from locations in-radius distanza.repulsione [turtle who]
  let lista.repulsi(lista.espls + lista.locations)
  let lista.x map[xcor-of ?]lista.repulsi
  let lista.y map[ycor-of ?]lista.repulsi
  let lista.distanze map[(1 / (distance ?))/ norma.tot]lista.repulsi
  let x.cor sum(map[?1 * sin towardsxy ?2 ?3 ]lista.distanze lista.x lista.y)
  let y.cor sum(map[?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
  set Gx precision x.cor 3
  set Gy precision y.cor 3 ]]]
end

to muovi
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
ask espls[
set heading direzione1
fd passo1 + speed]]
end

to verifica
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
 ask espls[
 let locat count locations in-radius (distanza.diff - 1)
 if locat >= 1 [
 posiziona
 set C<=LBC 0 ]]]
end


to calcola.risultante
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
  ask espls[
  let x.ris((a * Ex) - ((1 - a) * (Gx)))
  let y.ris((a * EY) - (( 1 - a) * (Gy )))
  ifelse x.ris != 0 or y.ris != 0 [set passo1 precision (sqrt((x.ris ^ 2) +( y.ris ^ 2)))3][set passo1 0]
  if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor [ set direzione1 precision (towardsxy (x.ris + xcor)(y.ris +
ycor))3]]]
end

to ferma
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
  ifelse covering%.locale  <=  local.covering%.locale
    [set C<=LBC C<=LBC + 1]
    [set C<=LBC 0]
  if (C<=LBC = iteraz.auto.differenzia)[
    differenzia1.2
    set num.volte num.volte + 1 ]]
end

to cambia
if num.giri2 < (num.iterazioni + num.iterazioni2 - 1)[
  if num.volte = valore.num.volte[
    ask ombre2[
    set x2 xcor-of ombra2(who)
    set y2 ycor-of ombra2(who) ]
    ask espls [die]
    ask aree.repulsione [die]
    ask aree.attrazione [die]
    ask shadows [die]
    ask ombre2[die]
    ricrea1
    contra ]]
end

to ricrea1
create-custom-locations 1 [setxy x2 y2 set shape "house" set size 1.6 set color pink set label who set dead 0]
create-custom-aree.repulsione1 1[setxy xcor-of location (who - 1) ycor-of location (who - 1) set shape
"location1" set size (((distanza.repulsione) * 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione1 1[setxy xcor-of location (who - 2)ycor-of location (who - 2) set shape "location"
set size ((copertura * 2)- 1)set hidden? not hidden?]
ask aree.attrazione1 [__tie area.attrazione1 (who) location(who - (2))]
end

to contra
set num.messe num.messe + 1
if ( num.messe < numero.locations )[
  ricrea
  copri]
end

to mostra.aree1
ifelse mostra.aree.influenza [ask aree.repulsione1[show-turtle]ask aree.attrazione1[show-turtle]]
                             [ask aree.repulsione1[ht]ask aree.attrazione1[ht]]
end

to copri
ask patches[
let locat count locations in-radius (copertura - 1)
ifelse( locat >= 1 )
  [ set densita.patch 0]
  [set densita.patch densita.media]]
end

to ricrea
create-custom-espls num.espl[posiziona set shape "dot" set size 3.2 set color red set label who]
create-custom-aree.repulsione num.espl[setxy xcor-of espl(who - num.espl ) ycor-of espl(who - num.espl )set
shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
create-custom-aree.attrazione num.espl[setxy xcor-of espl(who - (2 * num.espl) ) ycor-of espl (who - 2 *
num.espl )set shape "location" set size (((copertura)* 2 )- num.espl) set hidden? not hidden?]
create-custom-shadows num.espl[setxy xcor-of espl(who - (3 * num.espl) )ycor-of espl (who - (3 * num.espl) )
set shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre2 num.espl[setxy xcor-of espl(who - ( 4 * num.espl)) ycor-of espl(who - (4 * num.espl)) set
shape"dot"set size 1.6 set hidden? not hidden?]
ask aree.repulsione[__tie area.repulsione (who ) espl(who - num.espl )]
ask aree.attrazione [__tie area.attrazione (who ) espl(who - ((2 * num.espl)))]
set C<=LBC 0
set local.covering%.locale 0
set covering%.locale 0
set best.covering%.locale 0
set num.volte 0
ask locations[without-interruption[
set cov3 sum values-from patches in-radius copertura [densita.patch3]
set covering%((( sum values-from locations [cov3]) / total * 100))]]
```

```
end

to meme.ombre
ask ombre2[
if (contatore.parziale > 1)[
  if best.covering%.locale < covering%.locale
    [setxy xcor-of espl (who - (4 * num.espl))ycor-of espl(who - (4 * num.espl))set best.covering%.locale
covering%.locale] ]]
end

to differenzia1.2
set local.covering%.locale 0
set covering%.locale 0
ask espls[posizionamento.espls]
ask shadows[posizionamento.shadows]
set C<=LBC 0
end


to posizionamento.espls
setxy (random-xcor)(random-ycor)
if densita.media-of patch-here = 0
  [posizionamento.espls]
end

to posizionamento.shadows
setxy xcor-of espl(who - (3 * num.espl)) ycor-of espl(who - (3 * num.espl))
end

to memorizza
ask locations[without-interruption[
set cop.sing.loc ((sum values-from patches in-radius copertura [densita.patch3] )/ total)* 100]]
set lista.ord sort-by [cop.sing.loc-of ?1 < cop.sing.loc-of ?2] locations
set lista.ord.rid sublist lista.ord 0 1
set vero.loc-of one-of lista.ord.rid 1
end

to memorizza2
ask locations[without-interruption[
set cop.sing.loc ((sum values-from patches in-radius copertura [densita.patch3] )/ total)* 100
if vero.loc = 1
  [set cop.sing.loc 100]]]
set lista.ord2 sort-by [cop.sing.loc-of ?1 < cop.sing.loc-of ?2] locations
set lista.ord.rid2 sublist lista.ord2 0 1
set vero.loc-of one-of lista.ord.rid2 1
end

to sposta
ask locations[without-interruption[
if vero.loc-of location (who) = 1[
  set x3 xcor-of location(who)
  set y3 ycor-of location(who)
  set vero.a-of area.repulsione1 (who + 1) 1
  set vero.b-of area.attrazione1 (who + 2) 1  ]]]
create-custom-loca 1 [posiziona set shape "dot" set size 1.6 set color pink set label who]
create-custom-shadows3 1[setxy xcor-of loc(who - 1 )ycor-of loc (who - (1 ) ) set shape "dot" set size 1.6 set
color yellow set hidden? not hidden?]
create-custom-ombre3 1[setxy xcor-of loc(who - 2 ) ycor-of loc(who - (2)) set shape"dot"set size 1.6 set hidden?
not hidden?]
end
```

```
to sposta2
ask locations[without-interruption[
if vero.loc-of location (who) = 1[
  set x3 xcor-of location(who)
  set y3 ycor-of location(who) ]]]
create-custom-loca2 1 [posiziona set shape "leaf" set size 1.6 set color black set label who]
create-custom-shadows32 1[setxy xcor-of loc2(who - 1 )ycor-of loc2 (who - (1 ) ) set shape "dot" set size 1.6 set color yellow set hidden? not hidden?]
create-custom-ombre32  1[setxy  xcor-of  loc2(who  -  2  )  ycor-of  loc2(who  -  (2))  set  shape"dot"set  size  1.6  set hidden? not hidden?]
end


to attrai1.3
ask loca [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius copertura) - 1)
   let lista.patch remove patch-here values-from patches in-radius copertura [patch pxcor pycor]
   let lista.x map [pxcor-of ?] lista.patch
   let lista.y map [pycor-of ?] lista.patch
   let lista.densita map[(densita.patch-of ?) / norma]lista.patch
   let lista.distanze map [(distance ?) / norma]lista.patch
   let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   set Sx precision (x.cor) 3
   set Sy precision (y.cor) 3 ]]
end

to attrai1.32
ask loca2 [without-interruption[
let x xcor
let y ycor
let norma ((count patches in-radius copertura) - 1)
   let lista.patch remove patch-here values-from patches in-radius copertura [patch pxcor pycor]
   let lista.x map [pxcor-of ?] lista.patch
   let lista.y map [pycor-of ?] lista.patch
   let lista.densita map[(densita.patch-of ?) / norma]lista.patch
   let lista.distanze map [(distance ?) / norma]lista.patch
   let x.cor sum(map [?1 * ?2 * sin towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * ?2 * cos towardsxy ?3 ?4]lista.densita lista.distanze lista.x lista.y)
   set Sx2 precision (x.cor) 3
   set Sy2 precision (y.cor) 3 ]]
end

to respingi1.3
ask loca [without-interruption[
let x xcor
let y ycor
let norma count locations in-radius distanza.repulsione
   let lista.locations remove self values-from loca in-radius distanza.repulsione [turtle who]
   let lista.x map [xcor-of ?] lista.locations
   let lista.y map [ycor-of ?] lista.locations
   let lista.distanze map [(1 / (distance ?)) / norma ]lista.locations
   let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   set Hx precision x.cor 3
   set Hy precision y.cor 3]]
end
```

```
to respingi1.32
ask loca2 [without-interruption[
let x xcor
let y ycor
let norma count locations in-radius distanza.repulsione
   let lista.locations remove self values-from loca in-radius distanza.repulsione [turtle who]
   let lista.x map [xcor-of ?] lista.locations
   let lista.y map [ycor-of ?] lista.locations
   let lista.distanze map [(1 / (distance ?)) / norma ]lista.locations
   let x.cor sum(map [?1 * sin towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   let y.cor sum(map [?1 * cos towardsxy ?2 ?3]lista.distanze lista.x lista.y)
   set Hx2 precision x.cor 3
   set Hy2 precision y.cor 3]]
end

to calcola.risultante1.3
ask loca[
let x.ris ((a * Sx) - ((1 - a) * Hx))
let y.ris ((a * Sy) - ((1 - a) * Hy))
ifelse x.ris != 0 or y.ris != 0
  [set passo4 precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3]
  [set passo4 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[
  set direzione4 precision (towardsxy (x.ris + xcor) (y.ris + ycor)) 3 ]]
end

to calcola.risultante1.32
ask loca2[
let x.ris ((a * Sx2) - ((1 - a) * Hx2))
let y.ris ((a * Sy2) - ((1 - a) * Hy2))
ifelse x.ris != 0 or y.ris != 0
  [set passo42 precision (sqrt ((x.ris ^ 2) + (y.ris ^ 2))) 3]
  [set passo42 0]
if (x.ris + xcor) != xcor or (y.ris + ycor) != ycor[set direzione42 precision (towardsxy (x.ris + xcor) (y.ris +
ycor)) 3] ]
end

to distanziare1.3
ask loca[without-interruption[
set fac count locations in-radius distanza.diff;]
if contatore < 2[
  if fac > 1 [differenzia1.3]]]]
end

to conteggio1.3
if num.giri > 1[
  ifelse auto.differenzia [ifelse covering%.loca <= local.covering%
  [set C<=LBC C<=LBC + 1]
  [set C<=LBC 0]] [set C<=LBC 0]
    if (C<=LBC = iteraz.auto.differenzia) [
      differenzia1.3]]
end

to conteggio1.32
if  num.giri2 > 1[
  ifelse auto.differenzia [ifelse covering%.loca2 <= local.covering%2 [set C<=LBC C<=LBC + 1] [set C<=LBC
0]] [set C<=LBC 0]
      if (C<=LBC = iteraz.auto.differenzia) [
        differenzia1.32]]
```

```
end

to differenzia1.3
set contatore 0
set C<=LBC 0
set local.covering% 0
ask loca [posiziona]
ask shadows3 [posizionamento.shadows1.3]
end

to posizionamento.shadows1.3
setxy xcor-of loc (who - (1)) ycor-of loc (who - (1))
end

to posizionamento.shadows1.32
setxy xcor-of loc2 (who - (1)) ycor-of loc2 (who - (1))
end

to muovi1.3
ask loca[
set heading direzione4
fd passo4 + speed ]
end

to differenzia1.32
set contatore 0
set C<=LBC 0
set local.covering%2 0
ask loca2 [posiziona]
ask shadows32 [posizionamento.shadows1.32]
end

to muovi1.32
ask loca2[
set heading direzione42
fd passo42 + speed ]
end

to distacca.ombre1.3
ask ombre3 [
if (num.giri > 1) [if best.covering% < covering%.loca
[setxy xcor-of loc (who - (2)) ycor-of loc (who - (2)) set best.covering% covering%.loca]]]
end

to distacca.ombre1.32
ask ombre32 [
if (num.giri2 > 1) [if best.covering%2 < covering%.loca2
[setxy xcor-of loc2 (who - (2)) ycor-of loc2 (who - (2)) set best.covering%2 covering%.loca2]]]
end
to distacca.shadows1.32
ask shadows32 [
if num.giri2 > 1[
  ifelse covering%.loca2 > local.covering%2
    [setxy xcor-of loc2 (who - (1)) ycor-of loc2 (who - (1)) set local.covering%2 covering%.loca2]
    [set xcor xcor + 0.0 set ycor ycor + 0.0] ]]
end

to distacca.shadows1.3
ask shadows3 [
if num.giri > 1[
  ifelse covering%.loca > local.covering%
    [setxy xcor-of loc (who - (1)) ycor-of loc (who - (1)) set local.covering% covering%.loca]
    [set xcor xcor + 0.0 set ycor ycor + 0.0] ]]
end

to mostra.aree1.3
ifelse mostra.aree.influenza [ask aree.repulsione1[show-turtle]ask aree.attrazione1[show-turtle]]
                             [ask aree.repulsione1[ht]ask aree.attrazione1[ht]]
end

to final.set1.3
if num.giri = num.iterazioni [
  ask locations[
    if vero.loc = 1[
      ifelse cop.sing.loc-of location(who) < best.covering% [
        ask ombre3[
        set x4 xcor-of ombra3(who)
        set y4 ycor-of ombra3(who)]
        set bynary 1
        ask ombre3 [
        ask patches in-radius copertura[
        set densita.patch 0]]
        ask locations[
        if vero.loc = 1[
          set dead 1 ]]]
      [set bynary 0]]]]
 end

to final.set1.32
if num.giri2 = (num.iterazioni2 + num.iterazioni - 1 )[
  memorizza2
  ask locations[
  if vero.loc = 1 and dead = 0
    [ifelse cop.sing.loc-of location(who) < best.covering%2 [
      ask ombre32[
      set x42 xcor-of ombra32(who)
      set y42 ycor-of ombra32(who)]
      set bynary2 1
       set vero.a-of area.repulsione1 (who + 1) 1
       set vero.b-of area.attrazione1 (who + 2) 1 ]
     [set bynary2 0]]]

 if bynary2 = 1 [
   create-custom-locations 1 [setxy x42 y42 set shape "house" set size 1.6 set vero.loc 0 set color green set label
who set dead 0]
   create-custom-aree.repulsione1 num.espl[setxy xcor-of location(who - num.espl) ycor-of location(who -
num.espl)set shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
   create-custom-aree.attrazione1 num.espl[setxy xcor-of location(who - 2 * num.espl) ycor-of location (who - 2
* num.espl)set shape "location" set size (((copertura)* 2 )- 1) set hidden? not hidden?]
   ask aree.repulsione1[__tie area.repulsione1 (who) location(who - num.espl)]
   ask aree.attrazione1[__tie area.attrazione1 (who) location(who - (2 * num.espl))]]
  ask locations[without-interruption[
 set cov3 sum values-from patches in-radius copertura [densita.patch3]
 set covering%((( sum values-from locations [cov3]) / total * 100)) ]]
    if bynary = 1 [
   create-custom-locations 1 [setxy x4 y4 set shape "house" set size 1.6 set vero.loc 0 set color blue set label who]
   create-custom-aree.repulsione1 num.espl[setxy xcor-of location(who - num.espl) ycor-of location(who -
num.espl)set shape "location1" set size (((distanza.repulsione)* 2) - 1) set hidden? not hidden?]
   create-custom-aree.attrazione1 num.espl[setxy xcor-of location(who - 2 * num.espl) ycor-of location (who - 2
* num.espl)set shape "location" set size (((copertura)* 2 )- 1) set hidden? not hidden?]
```

```
   ask aree.repulsione1[__tie area.repulsione1 (who) location(who - num.espl)]
   ask aree.attrazione1[__tie area.attrazione1 (who) location(who - (2 * num.espl))]
   ask ombre3[die]
   ask locations[
   if dead-of location(who)= 1[
     die]]
   ask aree.repulsione1[
   if vero.a = 1[
     die ]]
 ask aree.attrazione1[
   if vero.b = 1[
     die ]]
  ask locations[without-interruption[
  set cov3 sum values-from patches in-radius copertura [densita.patch3]
  set covering%((( sum values-from locations [cov3]) / total * 100))
  ]]]]
if bynary2 = 1
[ask ombre32[die]
   ask locations[
   if vero.loc = 1[
       die ]]
     ask aree.repulsione1[
   if vero.a = 1[
     die ]]
   ask aree.attrazione1[
   if vero.b = 1[
     die ]]]
  ask loca2[die]
ask shadows3[die]
ask shadows32[die]
ask loca[die]
end
to continua
if num.messe = numero.locations[
  if num.giri = 0[
    memorizza
    sposta ]
while [num.giri < num.iterazioni] [
  ask locations[without-interruption[
  set cov3 sum values-from patches in-radius copertura [densita.patch3]
  set covering%((( sum values-from locations [cov3]) / total * 100)) ]]
  ask loca[without-interruption[
  set cov2 sum values-from patches in-radius copertura [densita.patch]
  set covering%.loca((( sum values-from loca [cov2]) / total * 100)) ]]
  set num.giri num.giri + 1
  set num.giri2 num.giri + 1
  attrai1.3
  respingi1.3
  calcola.risultante1.3
  muovi1.3
  distanziare1.3
  distacca.ombre1.3
  distacca.shadows1.3
  conteggio1.3 ]]
end

to continua2
if num.messe = numero.locations[
  if num.giri2 = num.iterazioni + 1[
    sposta2 ]
  while [num.giri2 < (num.iterazioni2 + num.iterazioni  - 1)] [
    ask locations[without-interruption[
    set cov3 sum values-from patches in-radius copertura [densita.patch3]
    set covering%((( sum values-from locations [cov3]) / total * 100))
    ]]
    ask loca2[without-interruption[
    set cov22 sum values-from patches in-radius copertura [densita.patch]
    set covering%.loca2((( sum values-from loca2 [cov22]) / total * 100))
    ]]
    set num.giri2 num.giri2 + 1
    attrai1.32
    calcola.risultante1.32
    muovi1.32
    distanziare1.32
    distacca.ombre1.32
    distacca.shadows1.32
    conteggio1.32 ]]
end
```

References

Aras N., Ozkisacik K.C., Altinell I.K. (2006). Solving the uncapacitated multi-facility Weber problem by vector quantization and self-organizing maps. *The Journal of the Operational Research Society*, vol. 57(1), pp. 82-93.

Archimede B., Coudert T. (2001). Reactive scheduling using a multi-agent model: the SCEP framework. *Engineering Applications of Artificial Intelligence*, vol.14, pp. 667-683.

Aydin M. E., Öztemel E. (2000). Job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, vol. 33(2–3), pp. 169–178.

Barbucha D., Jędrzejowicz P. (200). An Agent-Based Approach to Vehicle Routing Problem. *International Journal of Applied Mathematics and Computer Science*, vol. 4(2), pp. 538 – 543.

Bender T., Hennes H., Kalcsics J., Melo M.T., Nickel S. (2001). *Location Software and Interface with GIS and Supply Chain Management*. Berichte des Fraunhofer ITWM, Nr. 23.

Benedict J.M. (1983). *Three hierarchical objective models which incorporate the concept of excess coverage to locate EMS vehicles or hospital*. Master's thesis, Department of Civil Engineering, Northestern University, Evanston, Illinois.

BenHassine A., Ho T.B. (2007). An agent-based approach to solve dynamic meeting scheduling problems with preferences. *Engineering Applications of Artificial Intelligence*, vol. 20(6).

Berman O., Huang R. (2008). The minimum weighted covering location problem with distance constraints, *Computers & OR*, vol. 35(2), pp. 356-372.

Billari F.G., Fent T., Prskawetz A., Scheffran J. (eds.) (2006). *Agent-Based Computational Modelling: Applications in Demography, Social, Economic and Environmental Sciences (Contributions to Economics)*. Physica-Verlag, Heidelberg.

Bongaerts L., Monostori L., McFarlane D., Kadar B. (2000). Hierarchy in distributed shop floor control. *Computers in Industry*, vol. *43*(2), pp. 123–137.

Bozkaya B., Zhang J., Erkut E. (2002). An Efficient Genetic Algorithm for the p-Median Problem, in Drezner Z., Hamacher H. (eds.). *Facility Location: Applications and Theory*, pp. 179-205, Springer, Berlin.

Brandeau, M.L., Chiu, S.S. (1989). An overview of representative problems in location research. *Management Science*, 35, 645-674.

Bratman M. E., Israel D. J., Pollack M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, vol. 4(3), pp. 349-355.

Brennan R.W., Norrie D. H. (2001). Evaluating the performance of reactive control architectures for manufacturing production control. *Computers in Industry*, vol. 46(3), 235–245.

Brimberg J., Hansen P., Mladenovic N, Taillard E.D. (2000). Improvements and Comparison of Heuristics for solving the Multisource Weber Problem. *Operations Research*, vol. 48(3), pp. 444-460.

Brown D., Riolo R., Robinson D.T., North M., Rand W. (2005). Spatial Process and Data Models: Toward Integration of Agent-Based Models and GIS. *Journal of Geographical Systems*, vol. (7)1, pp. 25-47.

Cappanera P., Gallo G., Maffioli F. (2003). Discrete facility location and routing of obnoxious activities. *Discrete Applied Mathematics*, vol. 133(1-3), pp. 3 - 28.

Caridi M., Cavalieri S. (2004). Multi-agent systems in production planning and control: an overview. *Production Planning and Control*, vol. 15(2), pp. 106–118, Taylor&Francis.

Carrizosa E.J., Conde E., Munoz M., Puerto J. (1995). *The generalized weber problem with expected distances*. *RAIRO* vol. 29, pp. 35-57.

Cavalieri, B. (1647). *Exercitationes geometricae*. Bologna.

Cavalieri S., Garetti M., Macchi M., Taisch M. (2000). An experimental benchmarking of twomulti-agent architectures for production scheduling and control. *Computers in Industry*, vol. 43(2), 139–152.

Chen R. (1983). Solution of Minisum and Minimax Location–Allocation Problems with Euclidean Distances. *Naval Research Logistics Quarterly*, vol. 30, pp. 449–459.

Chen P., Hansen P., Jaumard B., Tuy, H. (1998). Solution of the multisource Weber and conditional Weber problems by DC programming, *Operations Research*, vol. 46(4), pp.548-562.

Chen Y. M., Wang S. C. (2007a). An agent-based evolutionary strategic negotiation for project dynamic scheduling. *International Journal of advanced manufacturing technology*, vol. 35(3-4).

Chen Y. M., Wang S. C. (2007b) Framework of agent-based intelligence system with two-stage decision-making process for distributed dynamic scheduling. *Applied Soft Computing*, vol. 7(1), pp. 229-245.

Chrystal G. (1885). On the Problem to Construct the Minimum Circle Enclosing *n* Given Points in the Plane. *Proceedings of the Edinburgh Mathematical Society*, 3, 30–33.

Chun A., Wai H., Wong R.Y.M. (2003). Optimizing agent-based meeting scheduling through preference estimation. *Engineering Applications of Artificial Intelligence*, vol. 16(7-8), pp. 727-743.

Chung C.H., Schilling D.A, Carbone R. (1983). The capacitated Maximal Covering Problem: A Heuristic, in *Proceedings of the Fourteenth Annual Pittsburgh Conference on Modeling and Simulation*, pp. 1423-1428, Pittsburgh, Pennsylvania.

Church R.L., ReVelle C.S. (1974). The maximal covering location problem. *Papers of the Regional Science Association*, vol. 32(1), pp. 101-118.

Cooper L. (1963). Location-allocation problems. *Operations Research*, vol. 11, pp. 37-52.

Cooper L. (1964). Heuristic methods for location-allocation problems. *SLAM Review*, vol. 6, pp. 37-53.

Cooper L. (1967). Solutions of generalized locational equilibrium models. *Journal of Regional Science*, vol. 7, pp. 1–18.

Cowling P. I., Ouelhadj D., Petrovic S. (2003). A multi-agent architecture for dynamic scheduling of steel hot rolling. *Journal of Intelligent Manufacturing*, vol. 14, 457–470.

Cowling P. I., Ouelhadj D., Petrovic S. (2004). Dynamic scheduling of steel casting andmilling using multi-agents. *Journal of Production Planning and Control*, vol. 15, 1–11.

Current J., Storbeck J. (1988). Capacitated Covering Models. *Environment and Planning B: Planning and design*, vol. 15(2), pp. 153-163.

Daskin M.S., Hogan K., Revelle C. (1988). Integration of multiple, excess, backup, and expected covering models. *Environment and Planning B: Planning and Design* vol.15(1), pp. 15- 35.

Davidsson P., Holmgren J., Persson J.A. (2007). On the Integration of Agent-Based and Mathematical Optimization Techniques. *Lecture Notes In Artificial Intelligence*, vol. 4496, pp. 1-10.

Dignum V., Weigand H., Xu L. (2002). Agent societies: Toward frameworks-based design. In M.J. Wooldridge M.J., Wei G., Ciancarini P., (ed), Agent-oriented software engineering II, *Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Articial Intelligence, Vol. 2222, Springer-Verlag, Berlin, Germany.

Drezner Z. (1987). A Heuristic Procedure for the Layout of a Large Number of Facilities. *Management Science*, vol. 33(7), pp. 907-915.

Drezner Z., Wesolowsky G.O. (1978). Facility location on a sphere. *Journal of the Operational Research Society,* vol. 29, pp. 997-1004.

Drezner Z., Wesolowsky G.O. (1978a). A New Method for the Multifacility Minimax Location Problem. *Journal of the Operational Research Society*, vol. 29, pp. 1095–1101.

Drezner Z., Wesolowsky G.O. (1978b). A Trajectory Method for the Optimization of the Multifacility Location Problem with l_p Distances. *Management Science*, vol. 24, pp. 1507–1514.

Drezner Z., Wesolowsky G.O. (1994). Finding the circle or rectangle containing the minimum weight of points. *Location Science*, vol. 2, pp. 83-90.

Drezner Z., Hamacher H. W. (2002). *Facility Location: Application and theory*, Springer-Verlag, Berlin.

Drezner T., Drezner Z. (2007). Equity Models in Planar Location. *Computational Management Science*, vol. 4(1), pp. 1-16.

duMerle O., Villeneuve D., Desrosiers J., Hansen P. (1999). Stabilized column generation, *Discrete Mathematics*, vol. 194(1), pp.229-237.

Eiselt, H.A., Laporte, G., Thisse, J-F., 1993. Competitive location models: A framework and bibliography. *Transportation Science*, 27 (1), 44-54.

Elzinga D.J., Hearn D.W. (1972). The Minimum Covering Sphere Problem. *Management Science* vol. 19, pp. 96–104.

Erkut E., Neuman S. (1989). Analytical Models for Locating Undesirable Facilities. *European Journal of Operational Research*, vol. 40(3), pp. 275-291.

Ferber J. (1999). *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman, Redwood City.

Fotheringham A.S., O'Kelly M.E. (1989). *Spatial interaction models: Formulation and applications*. Kluwer Academic, Dordrecht.

Francis R.L. (1964). On the Location of Multiple New Facilities with Respect to Existing Facilities. *The Journal of Industrial Engineering*, vol. 15, pp. 106–107.

Frey D., Nimis J., Worn H., Lockemann P. (2003). Benchmarking and robust multi-agent-based production planning and control. *Engineering Applications of Artificial Intelligence*, vol. 16(4), pp. 307-320.

Galvao R.D., Espejo L.G.A., Boffey B. (2000). A comparison of Lagrangean and surrogate relaxations for the maximal covering location problem. *European Journal of Operational Research*, vol. 124(2), pp. 377-389.

Goldsmith S. Y., Interrante L. D. (1998). An autonomous manufacturing collective for job shop scheduling. In proceedings of *AI & manufacturing research planning workshop,* pp. 69–74, AAAI Press.

Graham R.E., Lawler E.L., Lenstra J.K., Rinnoy Kan A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 4, 287-326.

Guo D., Ren B., Wang C. (2008). Integrated Agent-Based Modeling with GIS for Large Scale Emergency Simulation, *Lecture Notes in Computer Science*, vol.5370, pp.618-625.

Hakimi S.L. (1964). Optimal location of switching centers and the absolute centers and medians of a graph. *Operations Research*, vol.12, pp. 450–459.

Handler G.Y., Mirchandani P.B. (1979). *Location on Networks Theory and Algorithms*. MIT Press, Cambridge.

Hansen P., Mladenovic N., Taillard E. (1998). Heuristic solution of the multisource Weber problem as a p-Median problem, *Operation Research Letters*, vol. 22(2-3), pp. 55-62.

Homberger J. (2007). A multi-agent system for the decentralized resource-constrained multi project scheduling problem. *International transactions in operational research*, vol. 14(6), pp. 565-589.

Hotelling H. (1929). Stability in competition. *The Economic Journal*, vol. 39, pp. 41-57.

Kariv O., Hakimi S.L., (1979). An Algorithmic Approach to Network Location Problems. II: The p-medians. *SIAM – Journal on Applied Mathematics*, vol. 37(3), pp. 593-560.

Kariv O., Hakimi S.L. (1979a). An Algorithmic Approach to Network Location Problems. I: The *p*-centers. *SIAM Journal on Applied Mathematics*, vol. 37, pp. 513–538.

Karageorgos A., Mehandjiev N., Weichhart G., Hammerle A. (2003). Agent-based optimisation of logistics and production planning. *Engineering Applications of Artificial Intelligence*, vol. 16(4), pp. 335-348.

Karasakal O., Karasakal E.K. (2004). A maximal covering location model in the presence of partial coverage. *Computers & Operations Research,* vol. 31(9), pp. 1515-1526.

Kathib O. (1986). Real time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research,* vol. 5(1), pp. 90-99.

Kendall D. (1951). Some problems in the theory of queues. *Journal of the Royal Statistical Society*, 13, 151-153.

Klose A., Drexl A. (2005). Facility location models for distribution system design, *European Journal of Operational Research*, vol. 162(1), pp. 4-29.

Knotts G., Dror M., Hartman B.C. (2000). Agent-based project scheduling, *IIE Transactions*, vol. 32(5), pp. 387-401.

Kuenne R., Soland R. (1972). Exact and approximate solutions to the multisource Weber problem, *Mathematical Programming*, vol. 3(1), pp. 193-209.

Kuhn H. (1973). A note on Fermat's problem. *Mathematical Programming*, vol. 4, pp. 98–107.

Kuhn K., Kuenne R. (1962). An efficient algorithm for the numerical solution of the generalized Weber problem in spatial economics. *Journal of Regional Science*, vol. 4, pp. 21–33.

Leitao P., Restivo F. (2008). A holonic approach to dynamic manufacturing scheduling. *Robotics and Computer-Integrated Manufacturing*, vol. 220, pp. 37-46.

Lim M.K., Zhang D.Z. (2004). An integrated agent-based approach for responsive control of manufacturing resources. *Computers & Industrial Engineering*, vol. 46(2), pp. 221-232.

Lin G. Y., Solberg J. J. (1992). Integrated shop floor control using autonomous agents. *IIE Transactions*, vol. 24(3), pp. 57–71.

Liu N., Abdelrahman M.A., Ramaswamy S. (2007). A complete multiagent framework for robust and adaptable dynamic job shop scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 37(5).

Maimon O. (1986) The variance equity measure in locational decision theory. *Annals of Operations Research*, vol. 6, pp. 147–160.

Maimon O. (1988). An algorithm for the Lorenz measure in locational decisions on trees. *Journal of Algorithms*, vol. 9, pp. 583–596.

Masuyama S., Ibaraki T., Hasegawa T. (1981). The Computational Complexity of the *m*-Center Problems on the Plane. *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, vol. 64E, pp. 57–64.

Megiddo N., Supowit K.J. (1984). On the Complexity of Some Common Geometric Location Problems. *SIAM Journal of Computing*, vol. 13(1), pp.182-196.

Mehrez A. (1983). A note on the linear integer formulation of the maximal covering location problem with facility placement on the entire plane. *Journal of Regional Science*, vol. 23(4), pp.553-555.

Melo M.T., Nickel S., Saldanha-da-Gama F. (2009). Facility location and supply chain management – A review. *European Journal of Operational Research*, vol. 196(2), pp. 401-412.

Melzak Z.A. (1967). On the problem of Steiner. *Canada Mathematics Bullettin*, vol. 10**,** pp. 431-450.

Mes M., van der Heijden M., van Harten A. (2007). Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1), pp. 59-75.

Minieka E. (1970). The m-Center Problem. *SIAM Review*, vol. 12, pp. 138–39.

Moore G.C., ReVelle C.S. (1982). The hierarchical service location problem. *Management Science* vol. 28, pp. 775-780.

Muller J. P., Pischel M., Thiel M. (1995). *Modelling reactive behaviour in vertically layered agent architectures*, in Wooldridge M., Jennings N. R., (eds.), *Intelligent Agents: Theories, Architectures, and Languages, Lecture Notes In Artificial Intelligence*, vol. 890, pp. 261-276, Springer-Verlag, Berlin, Germany.

Nickel S., Hamacher H. (1998). *Classification of Location Models*. Location Science, 6, 229-242.

Ouelhadj D., Hanachi C., Bouzouia B. (1998). Multi-agent system for dynamic scheduling and control in manufacturing cells. In Proceedings of the *IEEE international conference on robotics and automation*, pp. 1256–1262.

Ouelhadj D., Hanachi C., Bouzouia B., Farhi A., Moualek A. (1999). A multi-contract net protocol for dynamic scheduling in flexible manufacturing systems. In Proceedings of the *IEEE international conference on robotics and automation*, pp. 1114–1120.

Ouelhadj D., Hanachi C., Bouzouia B. (2000). Multi-agent architecture for distributed monitoring in flexible manufacturing systems (FMS). In Proceedings of the *IEEE international conference on robotics and automation*, pp. 1120–1126.

Ouelhadj D., Cowling P. I., Petrovic S. (2003). Contract net protocol for cooperative optimisation and dynamic scheduling of steel production. In Ibraham A., Franke K., Koppen M., *Intelligent systems design and applications*, pp. 457–470, Springer, Berlin.

Parker D.C. (2005). Integration of Geographic Information Systems and Agent-Based Models of Land Use: Challenges and Prospects. In: Maguire, D.J., Batty, M., Goodchild, M. (eds.) GIS, *Spatial Analysis and Modelling*, pp. 403–422. ESRI Press, Redlands.

Parunak H.V.D., Kindrick J., Irish B.W. (1987). A Conservative Domain for Neural Connectivity and Propagation. In Huhns M.N. (ed.) *Distributed Artificial Intelligence*, pp. 307-311, Pitman, London.

Parunak H.V.D (1999). Industrial and practical applications of DAI. In: G. Weiss, Editor, *Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge MA (1999), pp. 79–120.

Pendharkar P. C. (1999). A computational study on design and performance issues of multi-agent intelligent systems for dynamic scheduling environments. *Expert Systems with Applications*, vol. 16(2), pp. 121–133.

Pirkul H., Schilling D.A. (1991). The maximal covering location problem with capacities on total workload. *Management Science*, vol. 37(2), pp.233-248.

Plastria F., Carrizosa E. (1999). Undesirable facility location with minimal covering objectives. *European Journal of Operational Research,* vol. 119(1), pp.158-180.

Plastria F. (2002). *Continuous Covering Location Problems*. In Drezner Z. and Hamacher H. W. (eds.) *Facility Location: Application and theory*, pp. 37-80, Springer-Verlag, Berlin, Germany.

Pottage J. (1983). *Geometrical Investigations.* Addison-Wesley, Reading, MA.

Raicu R., Taylor M.A.P., Zito R. (2002). *An evaluation of logistics network modelling tools available to South Australian companies: Literature review and initial evaluation.* Transport Systems Centre Technical Report, University of South Australia, Adelaide.

Ramos C. (1994). An architecture and a negotiation protocol for the dynamic scheduling of manufacturing systems. In *Proceedings of IEEE international conference on robotics and automation*, pp. 8–13.

Righini G., Zaniboni L.(2007). A branch-and-price algorithm for the multi-source Weber problem, *International Journal of Operation Research*, vol. 2(2), pp. 188-207.

Rosenschein J.S., Zlotkin G. (1994) *Rules of Encounter.* The MIT Press, Cambridge, Massachussets.

Rosing K.E. (1992). An optimal method for solving the (generalized) multi-Weber problem. *European Journal of Operation Research*, vol. 58(3), pp. 414-426.

Russell S., Norvig P. (1995). *Artificial Intelligence: A Modern Approach*, Prentice-Hall, New Jersey.

Sandholm T. W. (2000). Automated contracting in distributed manufacturing among independent companies. *Journal of Intelligent Manufacturing*, vol. 11(3), pp. 271–283.

Schöber A. (1999). *Locating lines and hyperplanes: theory and algorithms*, Kluwer, Dordrecht.

Sen A., Smith T.E. (1995). *Gravity models of spatial interaction behaviour*. Springer-Verlag, Berlin.

Serra D., Colomé R. (2001). Consumer choice and optimal location models: formulations and heuristics. *Papers in Regional Science*, vol. 80 (4), pp. 425-438.

Shaw J.M. (1988). Dynamic scheduling in cellular manufacturing systems: a framework for Network decision making. *Journal of Manufacturing Systems*, vol. *7*(2), pp. 83–94.

Shen W., Maturana F., Norrie D. H. (2000). MetaMorph II: an agent-based architecture for distributed intelligent design and manufacturing. *Journal of Intelligent Manufacturing*, 11(3), pp. 237– 251.

Shen W., Norrie D. H., Barthes J. P. A. (2001). *Multi-agent systems for concurrent intelligent design and manufacturing*. London: Taylor & Francis.

Shukla S. K., Tiwari M. K., Son Y. J. (2008). Bidding-based multi-agent system for integrated process planning and scheduling: A data-mining and hybrid Tabu-SA algorithm-oriented approach. *International Journal of Advanced Manufacturing Technology*, vol. 38, 163–175.

Simpson T. (1750). *The Doctrine and Application of Fluxions*. London.

Smith R. (1980). The contract net protocol: high level communicationand control in distributed problem solver. *IEEE Transactions on Computers*, vol. 29(12), pp. 1104–1113.

Smithies A. (1941). Optimal location in spatial competition. *Journal of Political Economy*, vol. 49, pp. 423-439.

Sousa P., Ramos C. (1999). A distributed architecture and negotiation protocol for scheduling inmanufacturing systems. *Computers in Industry*, vol. 38(2), pp. 103–113.

Storbeck J.E. (1982). Slack, natural slack and location covering, *Socioeconomic Planning Sciences,* vol. 16(3), pp. 99-105.

Sun J., Xue D. (2001). A dynamic reactive scheduling mechanism for responding to changes of production orders and manufacturing resources. *Computers in Industry*, vol. 46(2), pp. 189–207.

Toregas C., Swain R., Revelle C., Bergman L. (1971). The Location of Emergency Service Facilities. *Operations Research*, vol. 19, pp. 1363-1373.

Valchopoulou M., Silleos G., Manthou V. (2001). Geographic information systems in warehouse site selection decisions. *International journal of production economics,* vol. 71, pp. 205-212.

Ward J.E., Wendell R.E. (1985). Using Block Norms for Location Modeling. *Operations Research*, vol. 33, pp. 1074–1090.

Weber, A. (1909). *Über Den Standort Der Industrien*, 1. Teil: Reine Theorie Des Standortes, Tübingen, Germany. English Translation: *On the Location of Industries*, University of Chicago Press, Chicago, IL, 1929. (English Translation by C.J. Friedeich (1957), Theory of the Location of Industries, Chicago University Press, Chicago.).

Weiss G. (1999). *Multiagent Systems, a modern approach to distributed artificial intelligence*, MIT Press, Cambridge.

Weiszfeld E. (1937). Sur le point pour lequel la somme des distances de n points donnes est minimum. *Tohoku Mathematical Journal*, vol. 43, pp. 355-386.

Wesolowsky G.O. (1972). Rectangular Distance Location Under the Minimax Optimality Criterion. *Transportation Science,* vol. 6, pp. 103–113.

Wooldridge M. (2002). *An Introduction to Multiagent Systems*, John Wiley and sons, New York.

Zacharias M. (1913). Maxima und minima. Section 28 in Elementargeometrie, in *Encyklopddie der Mathematischen Wissenschaften* (W. F. Meyer & H. Mohrmann, Leipzig, Eds), 3. Band, 1. Teil (1914-1931). Pp. 1118-1137.

Zhou R., Fox B., Lee H.P., Nee A.Y.C. (2004). Bus maintenance scheduling using multi-agent systems, *Engineering Applications of Artificial Intelligence*, vol. 17(6), pp. 623-630.