Edge-Sets:
An Effective Evolutionary Coding of Spanning Trees

Günther R. Raidl
Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
raidl@ads.tuwien.ac.at

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
St. Cloud, MN, U.S.A.
julstrom@eeyore.stcloudstate.edu

Abstract

The fundamental design choices in an evolutionary algorithm are its representation of candidate solutions and the operators that will act on that representation. We propose representing spanning trees in evolutionary algorithms for network design problems directly as sets of their edges, and we describe initialization, recombination, and mutation operators for this representation. The operators offer locality, heritability, and computational efficiency. Initialization and recombination depend on an underlying random spanning tree algorithm; three choices for this algorithm, based on the minimum spanning tree algorithms of Prim and Kruskal and on random walks, respectively, are examined analytically and empirically. We demonstrate the usefulness of the edge-set encoding in an evolutionary algorithm for the NP-hard degree-constrained minimum spanning tree problem. The algorithm's operators are easily extended to generate only feasible spanning trees and to incorporate local, problem-specific heuristics. Comparisons of this algorithm to others that encode candidate spanning trees via the Blob Code, with network random keys, and as strings of weights indicate the superiority of the edge-set encoding, particularly on larger instances.

Keywords: Spanning trees, network design, edge-set representation, One-Max-Tree problem, degree constraints, evolutionary algorithms, hybridization.

1 Introduction

Let G be a weighted, connected, undirected graph with node set V and edges E. A spanning tree on G is a maximal, acyclic subgraph of G; that is, it connects all of G's nodes and contains no cycles. A spanning tree's cost is the sum of the costs of its edges; a spanning tree with the smallest possible cost is a minimum spanning tree (MST) on G. When the graph's edge costs are fixed and the search is unconstrained, the well-known algorithms of Kruskal [1] and Prim [2] identify MSTs in times that are polynomial in the number of nodes, as do more recent algorithms such as that described by Chazelle [3].

However, many variants of the MST problem are computationally difficult. Some seek to minimize objective functions other than the simple sum of a tree's fixed edge weights. For example, in the optimum communication spanning tree problem [4], a tree's cost depends on communication demands between each pair of nodes and on the tree's structure. In another version of this problem, edge costs are time-dependent [5].

Often, only spanning trees that satisfy particular constraints are feasible. Examples of such constraints include the degree constraint [6], which requires that no node in a spanning tree have more than $d \geq 2$ incident edges; leaf constraints [7], which specify or bound the number of leaves a spanning tree may have; the diameter constraint [8], which bounds the longest path in a spanning tree between any two nodes; and capacity constraints [9], which limit the capacity of edges.

Other combinatorial problems can be mapped to spanning tree problems. These include the rectilinear Steiner problem [10], which seeks the shortest tree composed of vertical and horizontal line segments that connects a collection of points in the plane, and the fixed-charge transportation problem [11], which seeks an economical plan for distributing a commodity from sources to destinations. Finally, some spanning tree problems seek to optimize several objective functions simultaneously [12].

Most such problems are NP-hard, and we can apply exact optimization algorithms only to small instances of them. For larger instances, we turn to heuristic techniques, including evolutionary algorithms (EAs).

Researchers have described EAs for all the MST-related problems listed above and others. Most often, these algorithms have used implicit representations to encode spanning trees and have applied decoding algorithms to identify the spanning trees that genotypes represent. They have applied positional operators like k-point crossover and position-by-position mutation to these genotypes; these operators exchange, rearrange, and modify symbols in parent genotypes.

We propose representing spanning trees for evolutionary search directly as sets of their edges. Applied to this representation, traditional initialization algorithms and positional crossover and mutation operators are unlikely to generate edge-sets that form spanning trees, so we describe new operators that are based on random spanning tree algorithms. With these operators, the edge-set representation exhibits significant advantages over previous codings of spanning trees. Evaluating edge-sets is fast, as are both recombination and mutation; under these operators, edge-sets have strong heritability and locality; it is usually easy to incorporate problem-specific heuristics into hybrid algorithms; and the operators can often handle constraints effectively.

Section 2 lists desirable qualities of an evolutionary coding and its operators and examines previous codings of spanning trees in this context. Section 3 describes the edge-set representation and initialization, crossover, and mutation operators for it. Section 3.1 addresses the surprisingly subtle issue of generating random spanning trees and derives the probabilities that various algorithms will generate trees of particular shapes. Section 4 examines the impact of the random spanning tree algorithm in an EA for a simple test problem called One-Max-Tree. Section 5 describes the specialization of the edge-set representation and its operators to the degree-constrained minimum spanning tree problem and demonstrates how edge-cost-based heuristics can be incorporated. In a comparison with other spanning tree representations, we find that in general the EA with edge-sets identifies the best solutions and is fastest for large, hard problem instances. This suggests the general usefulness of the edge-set representation in evolutionary algorithms for computationally hard spanning tree problems.

2 Representing Spanning Trees

An evolutionary algorithm's structure and parameter settings affect its performance, but the primary determinants of an EA's success or failure are the coding by which its genotypes represent candidate solutions and the interaction of the coding with the EA's recombination and mutation operators. Debate continues on when, how, and why EAs work, but most researchers agree on the relevance of the following features of evolutionary codings and their operators.

- *Space:* Chromosomes should not require extravagant amounts of memory.
- *Time:* The time complexities of evaluating, recombining, and mutating genotypes should be small. When genotypes represent spanning trees, evaluation may include decoding a genotype to identify the spanning tree it represents.
- *Feasibility:* All genotypes, particularly those generated by crossover and mutation, should represent feasible solutions.
- *Coverage:* The coding should be able to represent all feasible solutions. The entire search space, or at least one optimal solution, should be reachable via the EA's operators.
- *Bias:* In general, representations of all solutions should be equally likely, though bias may be an advantage if the favored solutions are near-optimal.
- *Locality:* A mutated genotype should usually represent a solution similar to that of its parent. Here, a mutated genotype should represent a tree that consists mostly of edges also found in its parent.
- *Heritability:* Offspring of crossover should represent solutions that combine substructures of their parental solutions. Here, offspring should represent trees consisting mostly of parental edges.
- *Constraints:* Decoding of genotypes and the crossover and mutation operators should be able to enforce problem-specific constraints. Here, such a constraint might bound the degrees of spanning trees' vertices.
- *Hybrids:* The operators should be able to incorporate problem-dependent heuristics. Here, such a heuristic might favor edges of lower cost.

One more consideration is particular to EAs that search spaces of subgraphs.

- *Sparse graphs:* Some codings can represent spanning trees only on complete graphs. Can the coding also be used to represent subgraphs of graphs that do not contain every possible edge?

The following sections describe codings of spanning trees of a graph $G = (V, E)$ on $n = |V|$ nodes with $m = |E|$ edges in terms of these considerations, except hybridization. Table 1 summarizes this discussion and includes entries for the edge-set representation.

Table 1: Properties of commonly used spanning tree encoding techniques and the new edge-set representation: Space, time, feasibility, coverage, bias, locality and heritability, ability to consider constraints, hybridizability, and applicability to sparse graphs; $n = |V|$, $m = |E|$.

Representation	Space	Time	Feasib.	Cover.	Bias	Loc./Herit.	Const.	Hybrid.	Sparse G
Char. vector	$O(m)$	$O(m)$	worst	yes	none	high	avg.	good	good
Predecessor	$O(n)$	$O(n)$	poor	yes	none	high	avg.	good	poor
Prüfer numbers	$O(n)$	$O(n \log n)$	yes	yes	none	low	poor	poor	poor
Blob Code	$O(n)$	$O(n^2)$	yes	yes	none	avg.	poor	poor	poor
Link-&-node biased	$O(m+n)$	$O(m+n \log n)$	yes	yes	high	avg.	good	good	good
Node-only biased	$O(n)$	$O(m+n \log n)$	yes	no	high	avg.	good	good	good
Netw. rand. keys	$O(m)$	$O(m \log m)$	yes	yes	low	high	good	possib.	good
Edge-set	$O(n)$	$O(n)$	yes	yes	depends	highest	good	good	good

2.1 Characteristic Vectors

A characteristic vector is a binary string whose positions correspond to items in a set. Each bit indicates whether or not the corresponding item is included in the structure the string represents. When such a vector represents a spanning tree, the items are the edges of G, and the vector indicates whether each edge is or is not part of the tree. Several researchers, including Davis *et al.* [13] and Piggott and Suraweera [14], have used characteristic vectors to represent spanning trees in evolutionary algorithms. These EAs have applied positional operators like k-point crossover and position-by-position mutation.

A characteristic vector requires space proportional to m. In a complete graph, $m = n(n-1)/2$ and the size of the search space is $2^{n(n-1)/2}$. However, only a tiny fraction of these genotypes represent feasible solutions, since a complete graph G has only n^{n-2} distinct spanning trees [15]. Penalizing genotypes that do not represent spanning trees has not been effective [14]. Repair strategies have been more successful [16], but they require additional computation and weaken the coding's locality and heritability.

2.2 The Predecessor Coding

A more compact representation of spanning trees is the predecessor or determinant coding, in which an arbitrary node in G is designated the root, and a genotype lists each other node's predecessor in the path from the node to the root in the represented spanning tree: if $\mathit{pred}(i)$ is j, then node j is adjacent to node i and nearer the root. Thus a genotype is a string of length $n-1$ over $\{1, 2, \ldots, n\}$, and when such a genotype encodes a spanning tree, its edges can be made explicit in time that is $O(n)$.

Applied to such genotypes, positional crossover and mutation operators will generate infeasible solutions, requiring again penalization or repair [17, 18]. Abuali *et al.* [19] described a repair mechanism that applies to spanning trees on sparse as well as complete graphs. Chu *et al.* [20] used the predecessor coding with both a penalty and repair in a genetic algorithm for the degree-constrained minimum spanning tree problem. Berry *et al.* [21] used it with special variation operators that produced only feasible solutions for the optimum communication spanning tree problem. Chou *et al.* [22] performed further investigations on this encoding with respect to the degree-constrained minimum spanning tree problem.

2.3 Prüfer Numbers

Cayley's Formula identifies as n^{n-2} the number of distinct spanning trees on a complete graph with n nodes [15, 23, pp. 98–106]. Prüfer [24] presented a constructive proof of this result: a pair of inverse mappings between spanning trees on n nodes and vectors of length $n-2$ over integers labeling the nodes. These vectors are called Prüfer numbers, and they encode spanning trees via Prüfer's mappings.

This coding is deceptively appealing. Prüfer numbers can be encoded and decoded in times that are $O(n \log n)$. Because every Prüfer number represents a unique spanning tree, they support positional genetic operators like k-point crossover and position-by-position mutation without requiring repair or penalization. The degree of each node in a spanning tree is one more than the number of times its label appears in the tree's Prüfer number.

However, many researchers have pointed out that Prüfer numbers have poor locality and heritability and are thus unsuitable for evolutionary search [17, 25, 26]. Patterns of values in Prüfer numbers do not represent consistent substructures of spanning trees, so the mutation of a single symbol may change many edges in the represented tree, and crossover often generates offspring whose trees share few edges with their parents' trees. Further, Prüfer numbers cannot be easily used on incomplete graphs, and it is difficult to implement constraints (except on degrees) or local heuristics.

Nonetheless, researchers have encoded spanning trees as Prüfer numbers in evolutionary algorithms for a variety of problems. These include the degree-constrained minimum spanning tree problem [18, 27], the minimum spanning tree problem with time-dependent edge costs [5], the fixed-charge transportation problem [12], and a bicriteria network design problem [28]. A recent comparison of codings in EAs for several spanning tree problems demonstrated the inferiority of Prüfer numbers [29].

There are many other mappings like Prüfer's from strings of $n-2$ node labels to spanning trees. Recently, Picciotto [30] and Deo and Micikevicius [31] described several of them. One, called the Blob Code, exhibits stronger locality and heritability than do Prüfer numbers, and an EA for the One-Max-Tree problem performed significantly better when it encoded spanning trees via the Blob Code than with Prüfer numbers [32]. As in Prüfer numbers, each node's degree in the spanning tree a string represents via the Blob Code is one more than the number of times the node's label appears in the string. The Blob Code's decoding takes time that in the worst case is $O(n^2)$ but is on average significantly faster.

2.4 Link-and-Node Biasing

Palmer and Kershenbaum [17] proposed a versatile coding of spanning trees that they called link-and-node biasing. In this coding, a genotype is a string of numerical weights associated with a graph's nodes and, optionally, with its edges. The tree such a genotype represents is identified by temporarily adding each node's weight to the costs of all the edges to which the node is incident; if present, edge weights are added to their edges' costs, too. Then Prim's algorithm is used to find a minimum spanning tree from the modified edge costs. Because of the application of Prim's algorithm, decoding (when implemented with a Fibonacci heap) requires time that is $\Theta(m + n \log n)$. Decoding can enforce constraints, though at the cost of additional computation. Any string of weights is a valid genotype, so positional crossover and mutation operators can be applied.

In general, there exist spanning trees that cannot be represented by node weights alone; edge weights are necessary to render every spanning tree reachable. Edge weights also reduce the bias of this representation toward star-like structures [33]. However, edge weights increase the size of each genotype on a complete graph from n values to $n+n(n-1)/2 = n(n+1)/2$.

Raidl and Julstrom [34] proposed a variant of this coding, called weight-coding, in an EA for the degree-constrained minimum spanning tree problem. In weight-coding, the weights in each genotype are initially selected from a log-normal distribution, and the biasing scheme is multiplicative rather than additive. The decoding algorithm was modified to yield only trees that satisfy the problem's degree constraint. Krishnamoorthy *et al.* [18] described another variant of link-and-node-biasing, which they called problem search space, for the degree-constrained minimum spanning tree problem.

2.5 Network Random Keys

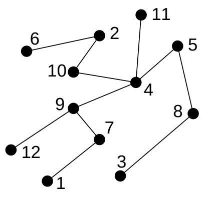
Bean [35] described random keys to encode permutations; Rothlauf *et al.* [36, 37] adapted random keys to represent spanning trees and called them network random keys. In this coding, a genotype is a string of real-valued weights, one for each edge. To identify the tree a genotype represents, the edges are sorted by their weights and Kruskal's minimum spanning tree algorithm considers the edges in sorted order. As with link-and-node-biasing, any string of weights is a valid genotype and positional crossover and mutation operators may be used.

Because they represent trees via Kruskal's algorithm, network random keys are disproportionately likely to encode star-like trees and disproportionately unlikely to encode path-like trees, a phenomenon that Section 3.1.2 below examines. Each genotype requires space that is $O(m)$, and decoding is computationally expensive—$O(m \log m)$—because it requires sorting the edges. Thus, network random keys are effective only for small or sparse problems. Rothlauf *et al.* [36] reported good results with this coding on instances of the optimum communication spanning tree problem of up to 26 nodes. Schindler *et al.* [38] further investigated random network keys in an evolution strategy framework.

2.6 Other Representations

Other representations of spanning trees are less often used. In degree-based permutations [39], a genotype consists of two strings. The first holds a permutation of the node labels, and the second holds the nodes' degrees. The tree this pair represents is obtained by connecting the nodes in the specified order and with the specified degrees. In Prüfer-based permutations [40], a genotype holds indices into a list of multiple copies of the node labels. These indices specify the order in which labels are removed from the list and concatenated to an initially empty Prüfer number, which in turn represents a spanning tree. Not surprisingly, neither of these codings exhibits strong locality or heritability under positional operators.

Knowles and Corne [25] described a coding of spanning trees whose degrees do not exceed a bound $d \geq 2$. In it, a genotype is an array of $n \cdot d$ integers that influence the order in which a variant of Prim's algorithm attaches the nodes to a growing spanning tree. On several hard instances of the degree-constrained minimum spanning tree problem, an EA using this coding outperformed several other heuristics. Section 5 below compares this coding with the edge-set representation.



{ (6,2), (12,9), (2,10), (7,9),
(3,8), (7,1), (4,11), (4,5),
(10,4), (9,4), (5,8) }

Figure 1: A spanning tree on twelve nodes and an edge-set that represents it.

To our knowledge and with the exception of our own recent work [41, 42], only the following publications have considered EAs for spanning tree problems that represent candidate spanning trees directly as sets of their edges. Li and Bouchebaba [43] proposed crossover and mutation operators based on edges, paths, and subtrees of spanning trees in an EA for the optimum communication spanning tree problem. Li [44] described the tree representation in more detail. However, most of the crossover and mutation operators described for it require $O(n^2)$ time; our operators, which the next section describes, are more efficient. Recently, Rothlauf has considered representations for genetic and evolutionary algorithms [37]; he also presents a direct representation of spanning trees.

3 The Edge-Set Representation and its Operators

We propose representing spanning trees directly as sets of their edges; Figure 1 shows a spanning tree on twelve nodes and an edge-set that represents it. This representation can be implemented in an array or a hash table whose entries are the pairs of nodes that define each edge. The latter data structure allows insertion, lookup, and deletion of individual edges in constant time; both require space that is linear in the number n of nodes.

An evolutionary algorithm requires an initial population of diverse genotypes. For the edge-set representation, we therefore need an algorithm that generates spanning trees at random. This algorithm is also the basis for the recombination operator. Recombination of two parent spanning trees merges their edge-sets and derives a new random spanning tree from this union. Recombination can be considered an extension of uniform crossover into the domain of spanning trees.

Since creating random spanning trees is such a fundamental task, Section 3.1 describes several algorithms for this purpose and examines them analytically. Section 3.2 compares these algorithms empirically. Section 3.3 describes recombination of parent edge-sets based on the generation of random spanning trees, and Section 3.4 presents mutation for edge-sets that represent spanning trees.

3.1 Creating Random Spanning Trees

When an evolutionary algorithm searches a space of spanning trees, its initial population consists of genotypes that represent random trees. It is not as simple as it might seem to choose spanning trees of a graph so that all are equally likely. Techniques based on Prim's and Kruskal's minimum spanning tree algorithms do not associate uniform probabilities with spanning trees; techniques that do are limited in their application, computationally tedious, or not guaranteed to terminate.

3.1.1 Extending Prim's Algorithm

Prim's algorithm [2] greedily builds a minimum spanning tree from a start node by repeatedly appending the lowest-cost edge that joins a new node to the growing tree. Choosing each new edge at random rather than according to its cost yields a procedure we call *PrimRST*:

```
procedure PrimRST($V$,$E$):
  $T \leftarrow \emptyset$;
  choose a random starting node $s \in V$;
  $C \leftarrow \{s\}$; (* set of connected nodes *)
  $A \leftarrow \{e \in E \mid e = (s, v), v \in V\}$; (* eligible edges *)
  while $C \neq V$ do
    choose an edge $(u, v) \in A, u \in C$ at random;
    $A \leftarrow A - \{(u, v)\}$;
    if $v \notin C$ then (* connect $v$ to the partial tree *)
      $T \leftarrow T \cup (u, v)$;
      $C \leftarrow C \cup \{v\}$;
      $A \leftarrow A \cup \{e \in E \mid e = (v, w) \wedge w \notin C\}$;
  return $T$.
```

Representing the graph $G = (V, E)$ with adjacency lists [45, pp. 232–233] allows fast identification of the edges adjacent to each newly connected node and the implementation of PrimRST in time that is $O(m)$. If G is complete, it is necessary only to keep track of the set of nodes currently in the spanning tree (C in the sketch above) and its complement $V - C$; each new edge is identified by choosing one node at random from each set, and the algorithm's time and space are both only $O(n)$.

Not only is PrimRST simple and efficient, but it can easily be extended to return only trees that satisfy various constraints. Constraints on a tree's degree, diameter, and capacity, for example, can be honored by accepting only edges whose inclusion does not render the tree invalid.

Unfortunately, PrimRST yields trees of some structures with much higher probabilities than others. The edges adjacent to PrimRST's starting node are in the set A of eligible edges from the beginning and are therefore more likely to be included in the tree than are edges adjacent to later nodes. Thus, the trees that have the highest probability under PrimRST are *stars*, in which one root node connects to all the others.

More formally, consider the probabilities that PrimRST returns a star and a Hamiltonian path on a complete graph G of n nodes; the tendencies these probabilities indicate also hold for incomplete graphs.

The first two edges PrimRST fixes always share a node. A star is created if in each following step $i = 3, \ldots, n-1$, a new node is connected to this common root, which is the case with probability $1/i$. The probability that PrimRST returns a star is therefore

$$1 \cdot 1 \cdot \frac{1}{3} \cdot \frac{1}{4} \cdot \frac{1}{5} \cdots \frac{1}{n-1} \;=\; \frac{2}{(n-1)!}\,.$$

There are n distinct stars on n nodes, so the probability of a *particular* star is

$$p_{\text{star}} \;=\; \frac{1}{n} \cdot \frac{2}{(n-1)!} \;=\; \frac{2}{n!}\,.$$

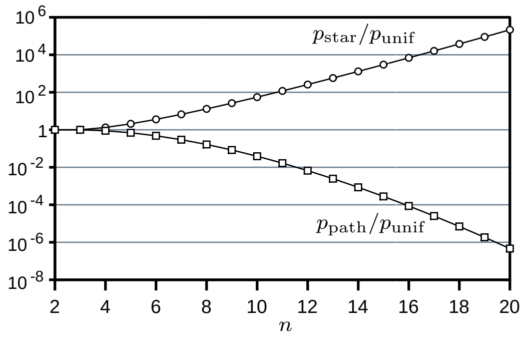


Figure 2: PrimRST: Normalized probabilities of creating a particular path or a particular star for a complete graph on n nodes.

The number of distinct spanning trees on n nodes is n^{n-2} [15]; an unbiased algorithm would assign each the probability

$$p_{\text{unif}} = \frac{1}{n^{n-2}} .$$

The probability p_{star} that PrimRST generates a particular star is greater than this uniform probability for any $n \geq 4$:

$$\frac{2}{n!} > \frac{1}{n^{n-2}} \iff 2 > \frac{n!}{n^{n-2}} .$$

Conversely, the trees that have the smallest probabilities are *(Hamiltonian) paths*, in which all nodes, except two leaves, have degree two. The probability that PrimRST returns *any* path is

$$1 \cdot 1 \cdot \frac{2}{3} \cdot \frac{2}{4} \cdot \frac{2}{5} \cdots \frac{2}{n-1} \;=\; \frac{2^{n-2}}{(n-1)!} .$$

The first two edges chosen will always form part of a path; each subsequent edge must be incident to one of the leaves of the partial path.

A complete, undirected graph on n nodes contains $n!/2$ Hamiltonian paths, so the probability that PrimRST will return a *particular* one of them is

$$p_{\text{path}} \;=\; \frac{2}{n!} \cdot \frac{2^{n-2}}{(n-1)!} \;=\; \frac{2^{n-1}}{(n-1)!\,n!} .$$

This probability is less than the uniform probability $p_{\text{unif}} = 1/n^{n-2}$ for $n \geq 4$; see Appendix A for a proof of this result.

Figure 2 shows the normalized probabilities $p_{\text{star}}/p_{\text{unif}}$ and $p_{\text{path}}/p_{\text{unif}}$ plotted against the number n of nodes in the graph. Even for small n, the probability of a particular path is several orders of magnitude smaller than that of a particular star, and the difference increases dramatically with n. The probability that PrimRST returns a particular general tree always lies between the probabilities of a path and a star.

On the other hand, the probability that PrimRST will return *some* path is greater than the probability that it will return *some* star by a factor of 2^{n-3}, because there are so many more paths than stars.

3.1.2 Extending Kruskal's Algorithm

Kruskal's algorithm [1] also applies a greedy strategy to build a minimum spanning tree of a graph G. It examines G's edges in order of increasing cost and includes in the tree those that connect previously unconnected components. Examining the edges in random order yields a procedure we call *KruskalRST*:

```
procedure KruskalRST($V$,$E$):
  $T \leftarrow \emptyset$;
  $A \leftarrow E$;
  while $|T| < |V| - 1$ do
    choose an edge $(u, v) \in A$ at random;
    $A \leftarrow A - \{(u, v)\}$;
    if $u$ and $v$ are not yet connected in $T$ then
      $T \leftarrow T \cup \{(u, v)\}$;
  return $T$.
```

By using a union-find data structure with weight balancing and path compression [45, pp. 183–189], the determination that two nodes are or are not connected in the developing spanning tree can be carried out in nearly constant time. Thus the total time for KruskalRST is $O(m)$, as with PrimRST in its general case.

Also like PrimRST, KruskalRST is more likely to generate some trees than others. We examine some probabilities in complete graphs, but the tendencies apply generally, as before.

Consider the probability that KruskalRST returns a particular star. There are $n - 1$ edges in the star and $\binom{n}{2}$ edges in G, so the probability that the first edge KruskalRST chooses is in the star is $(n - 1)/\binom{n}{2}$. At each following step, one of the star's remaining unchosen edges must be included in T, and the total number of eligible edges is $\binom{n}{2} - \binom{|T|}{2}$; an edge is eligible unless it joins two nodes already in the star.

Thus the probability that KruskalRST identifies the star is

$$p_{\text{star}} = \frac{n-1}{\binom{n}{2}} \cdot \frac{n-2}{\binom{n}{2} - 1} \cdot \frac{n-3}{\binom{n}{2} - \binom{3}{2}} \cdot \frac{n-4}{\binom{n}{2} - \binom{4}{2}} \cdots \frac{1}{\binom{n}{2} - \binom{n-1}{2}} = \frac{(n-1)!}{\prod_{i=1}^{n-1}\left(\binom{n}{2} - \binom{i}{2}\right)} .$$

Since

$$\begin{aligned}
\prod_{i=1}^{n-1}\left(\binom{n}{2} - \binom{i}{2}\right) &= \frac{1}{2^{n-1}} \cdot \prod_{i=1}^{n-1}(n(n-1) - i(i-1)) \\
&= \frac{1}{2^{n-1}} \cdot \prod_{i=1}^{n-1}(n-i)(n+i-1) \\
&= \frac{1}{2^{n-1}} \cdot (n-1)! \cdot \frac{(2n-2)!}{(n-1)!} = \frac{(2n-2)!}{2^{n-1}} ,
\end{aligned}$$

we can write

$$p_{\text{star}} = \frac{2^{n-1}(n-1)!}{(2n-2)!} .$$

This probability is again larger than p_{unif}, but smaller than the probability that PrimRST returns the same star.

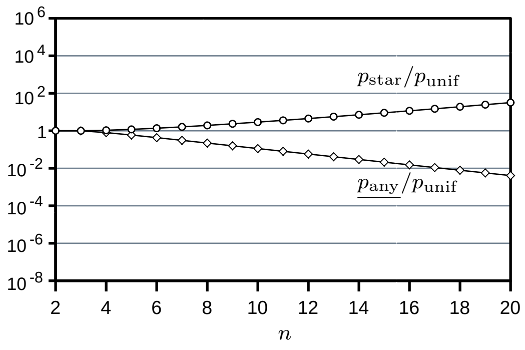


Figure 3: KruskalRST: Normalized probabilities of creating a particular star and a lower bound for the creation of any particular structure for a complete graph on n nodes.

The exact probability that KruskalRST returns a path is much harder to identify. Instead, we derive a lower bound $\underline{p_{\text{any}}}$ for the probability that KruskalRST returns a particular spanning tree of any structure, including a path.

Consider the cases in which KruskalRST identifies the $n-1$ edges of the target tree before any others. The orders in which the target tree's edges and the $\binom{n}{2}-(n-1)$ remaining edges are examined are irrelevant, so the bound we seek is

$$\underline{p_{\text{any}}} = \frac{(n-1)!\,(n(n-1)/2-n+1)!}{(n(n-1)/2)!} = \frac{1}{\binom{n(n-1)/2}{n-1}}\,.$$

This computation ignores the cases in which KruskalRST discards edges before the tree is finished, so the bound is not tight. Still, it shows that KruskalRST returns paths (in particular) with substantially higher probabilities than does PrimRST. Figure 3 plots the ratios of probabilities $p_{\text{star}}/p_{\text{unif}}$ and $\underline{p_{\text{any}}}/p_{\text{unif}}$ against the number n of nodes in G for KruskalRST. Comparing this figure with Figure 2 makes clear that KruskalRST is much less biased than PrimRST.

KruskalRST can honor constraints by rejecting edges that would render partial solutions invalid, but it is not as flexible as PrimRST. Local constraints, as on nodes' degrees (Section 5), can be maintained in this way, but constraints involving large parts of spanning trees, as on trees' diameters or the capacities of edges in a communications network [46], usually cannot. In the latter cases, KruskalRST may generate valid, unconnected subtrees that cannot be connected to form a feasible solution.

3.1.3 A Random Walk through G

Both PrimRST and KruskalRST associate non-uniform probabilities with the spanning trees of a graph G, though the latter is less biased than the former. This section describes three mechanisms that generate spanning trees in an unbiased way.

Section 2.3 described Prüfer numbers: vectors of length $n-2$ over an alphabet of n node labels that are in one-to-one correspondence with the spanning trees on n nodes. It is easy to generate random Prüfer numbers with uniform probabilities; decoding them yields unbiased random spanning trees. This is an effective and efficient mechanism when the underlying graph G is complete.

Guénoche [47] described a random spanning tree algorithm for general graphs. It is based on the fact that the number of distinct spanning trees in a graph can be found by computing a determinant of size $n \times n$, and it requires time that is $O(n^5)$. Colbourne [48] modified the algorithm to require fewer determinant computations and reduced its time to $O(n^3)$. This might be acceptable for the initialization of an EA's population, but is too expensive to be the basis of a recombination operator.

Broder [49] described a probabilistic method based on a random walk in G. A particle begins at an arbitrary node in G. At each step, it moves over a randomly chosen adjacent edge to one of its neighbors. When the particle visits a node for the first time, the edge it traverses joins the spanning tree. The algorithm terminates when the particle has visited every node and thus completed a spanning tree. We call this algorithm *RandWalkRST*:

```
procedure RandWalkRST($V$,$E$):
  $T \leftarrow \emptyset$;
  $v_0 \leftarrow$ a random node of $G$;
  mark $v_0$ as visited;
  while $|T| < n - 1$ do
    $v_1 \leftarrow$ a random neighbor of $v_0$;
    if $v_1$ has not yet been visited then
      $T \leftarrow T \cup \{(v_0, v_1)\}$;
      mark $v_1$ as visited;
    $v_0 \leftarrow v_1$;
  return $T$.
```

Broder showed that this process returns spanning trees with uniform probabilities. While its worst-case time is unbounded—the particle may never visit some nodes—its *expected* time is $O(n \log n)$ for almost all graphs and $O(n^3)$ for a few special cases.

Like PrimRST, RandWalkRST can honor constraints by disallowing edges that would violate them, though such a strategy will in general render non-uniform the probabilities of valid spanning trees.

Broder [49] also described a mechanism that walks randomly not through the graph but through the space of spanning trees on it. It begins with an arbitrary spanning tree, then repeatedly replaces a random edge with a new one that reconnects the tree. The distribution of spanning trees generated by this scheme converges to a uniform distribution even if G is incomplete, and the distribution is nearly uniform after a number of steps that is polynomial in n.

3.2 An Empirical Comparison

To further examine properties of spanning trees created by PrimRST and KruskalRST, we applied these algorithms and RandWalkRST to complete graphs of different sizes and measured the diameters of the resulting general spanning trees. A tree's diameter is the number of edges in a longest path in the tree. This value expresses the similarities of the tree to a star, which has the minimum diameter of two, and to a path, which has the maximum diameter of $n - 1$.

In the experiments, the number n of nodes ranged from three to 1 000, and each random tree algorithm was called 500 times for each size. Figure 4 summarizes the results of these

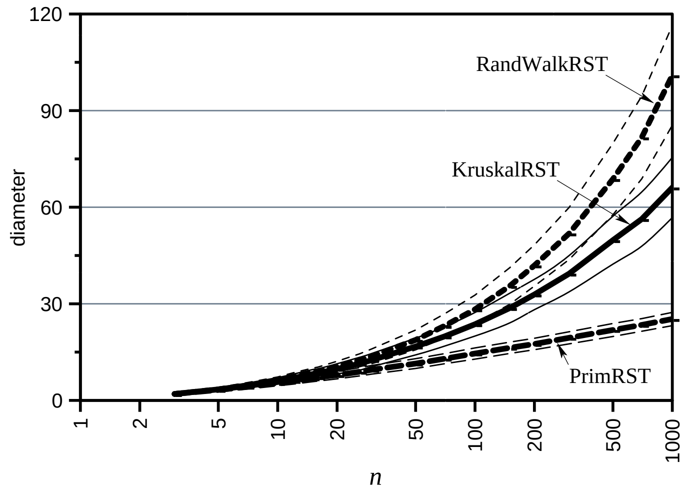


Figure 4: Average diameters (bold lines) with standard deviations added and subtracted (thin lines) of random spanning trees created by PrimRST, KruskalRST, and RandWalkRST on complete graphs of n nodes.

trials; it plots the average diameters of the trees the three algorithms returned against the size of the underlying graph.

The results conform to the analyses in the preceding sections. At each number of nodes greater than three, PrimRST returned trees of the smallest average diameter, thus the most like stars. KruskalRST returned trees of larger average diameter, thus less like stars than those of PrimRST. RandWalkRST returned trees of the largest average diameter, thus the least like stars and the most like paths.

These differences were consistent and increased with n. When $n = 1\,000$, the average diameter of the PrimRST trees was about a quarter of that of RandWalkRST's trees, and the average diameter of KruskalRST's trees was about 2/3 of the latter value. Both KruskalRST and especially PrimRST are biased toward shallow, star-like structures.

3.3 Recombination

To provide good heritability, a recombination operator must build an offspring spanning tree that consists mostly or entirely of edges found in the offspring's parents. This can be done by applying any of the random tree algorithms described in the previous section to the graph $G_{\text{cr}} = (V, T_1 \cup T_2)$, where T_1 and T_2 are the edge-sets of the parental trees. Figure 5 illustrates this operation.

When recombination applies PrimRST, it builds adjacency lists from the two parental edge-sets. In this case, $|T_1 \cup T_2| \leq 2n - 2$, and the time recombination requires is $O(n)$. Recombination based on KruskalRST also requires time that is linear in n.

In the absence of constraints, it is always possible to build an offspring spanning tree using only edges from the two parents. However, when a problem constrains spanning trees, recombination using only parental edges, whether based on PrimRST, KruskalRST, or some other algorithm, may generate infeasible trees. When recombination runs out of

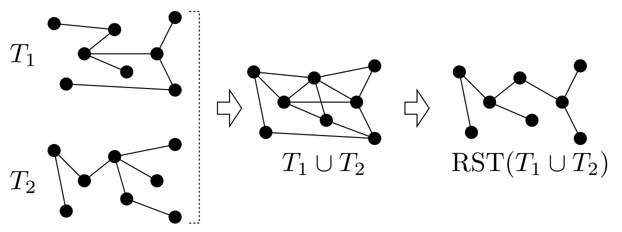


Figure 5: Recombination of spanning trees by applying a random spanning tree algorithm to the union of the parental edge-sets.

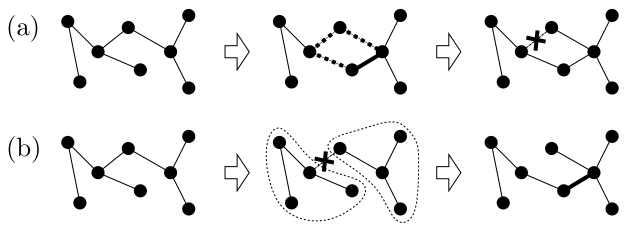


Figure 6: Mutation by either (a) including a new edge and removing another from the introduced cycle or (b) by removing a randomly chosen edge, determining the separated components, and reconnecting them by a new edge.

feasible edges from $T_1 \cup T_2$, it must include feasible edges from $E - (T_1 \cup T_2)$; *i.e.*, edges found in neither parent. Usually, only a few non-parental edges are necessary to build a valid offspring, and this strategy avoids complex repair and penalty mechanisms. Section 5 demonstrates this for degree-constraints on the nodes.

We distinguish variants of the recombination operator by the random spanning tree algorithms they use and their treatment of edges appearing in both parents. Recombination may favor such common edges, always including them in an offspring, or it may treat all parental edges equally. Section 4 below compares these alternatives. As demonstrated in Section 5, recombination may also modify edges' probabilities heuristically, favoring, for example, feasible edges of lower cost.

3.4 Mutation

Mutation (re-)introduces genetic information into the population. To provide high locality, a mutation operator should usually make a small change in a parent solution. Here, the smallest possible change is the replacement of one spanning tree edge with another feasible edge.

This replacement can be implemented in two ways. The first, which Figure 6(a) depicts, chooses an edge at random from $E - T$ and includes it in T. This creates a cycle. The operator then chooses a random edge in the cycle and removes it from T. If T is temporarily represented by adjacency lists, a depth-first search can quickly identify the cycle, and the operator's time is $O(n)$.

The second approach, which Figure 6(b) illustrates, deletes a random edge from T and replaces it with a random new edge that reconnects the tree. On a complete graph and without constraints, this operation's time is also $O(n)$, but it may be computationally more expensive on an incomplete graph or under constraints.

4 The One-Max-Tree Problem

We compare the techniques just described in a steady-state EA for a simple spanning tree problem called *One-Max-Tree*. In the well-known One-Max problem [50], a bit string's fitness is the number of 1's it contains. In the One-Max-Tree problem [51], a target spanning tree on a complete base graph is specified, and the fitness of any other tree is the number of edges that it shares with the target. An EA that supports meaningful building blocks should be able to solve this problem easily.

The evolutionary algorithm for One-Max-Tree is conventional. At each step, it selects two parents in binary tournaments with replacement. It recombines them to generate one offspring, which is then mutated. The resulting solution replaces the worst in the population, except when it duplicates an existing solution. In the latter case, to preserve diversity in the population, the EA discards the offspring.

Initialization of the population and recombination are based on PrimRST, KruskalRST, or RandWalkRST, as the previous section described. In recombination, either all parental edges are treated equally, or edges appearing in both parents are favored and automatically appear in their offspring. We identify the latter variants with the symbol '*'.

The resulting six versions of the EA were applied to instances of One-Max-Tree of 10 to 100 nodes. The target trees were randomly chosen stars, general trees (generated by RandWalkRST), and paths. The EA's population size was 100 throughout, and the algorithm was run 50 independent times for each combination of initialization and recombination technique, problem size, and target tree structure. The EA always identified the target tree, though the number of evaluations it required to do so varied. Table 2 lists the average numbers *evals* of evaluations required to identify the target and their standard deviations s.

The most prominent result is that regardless of the RST-algorithm used for initialization and recombination, the EA usually identified stars more quickly than general trees and much more quickly than paths, and the differences became larger as the number of nodes increased. Figure 7 plots the average numbers of iterations needed to identify the target against the problem size for the EA with RandWalkRST. Since RandWalkRST does not favor stars, we might be surprised that it identifies them most rapidly. The explanation lies in the fact that the space of spanning trees is most sparse near stars; the EA can reach them from nearby structures most easily.

More formally, let G be a complete graph on n nodes and let two spanning trees be *neighbors* if they differ in exactly one edge. If G is complete, a star has only $(n-1)(n-2) = O(n^2)$ neighbors, while a path has $\frac{1}{6}n(n-1)(n+1) - n + 1 = O(n^3)$ neighbors [52].

Due to the large differences in the probabilities of KruskalRST and especially PrimRST for creating star- versus path-like structures, one might expect that the random spanning tree algorithm underlying initialization and recombination would substantially affect the EA's performance. However, Table 2 indicates that this is not the case. Columns $\delta_{\mathrm{P,K}}$ and $\delta_{\mathrm{K,R}}$ list the relative differences $(evals_1 - evals_2)/\min(evals_1, evals_2)$, where $evals_1$ and $evals_2$ are the numbers of evaluations needed to identify the target tree when the EA used PrimRST and KruskalRST, respectively KruskalRST and RandWalkRST. In general, the differences are small and the variances are large. Columns $\alpha_{\mathrm{P,K}}$ and $\alpha_{\mathrm{K,R}}$ list the error probabilities in

Table 2: Average numbers of iterations needed by the EA using PrimRST, KruskalRST, or RandWalkRST during initialization and recombination to solve the One-Max-Tree problem for different target structures; in the $*$-variants, recombination favored edges appearing in both parents.

Struct.	n	PrimRST		KruskalRST		RandWalkRST		$\delta_{\mathrm{P,K}}$	$\alpha_{\mathrm{P,K}}$	$\delta_{\mathrm{K,R}}$	$\alpha_{\mathrm{K,R}}$
		evals	*s*	*evals*	*s*	*evals*	*s*				
star	10	446	113	737	180	797	172	-65%	0.0%	-8%	4.6%
	20	1 062	247	2 512	552	2 906	504	-137%	0.0%	-16%	0.0%
	50	14 742	4 973	27 823	7 759	29 267	7 221	-89%	0.0%	-5%	16.9%
	100	113 654	29 018	192 248	45 193	196 431	41 498	-69%	0.0%	-2%	31.5%
tree	10	836	196	813	187	890	217	3%	27.0%	-10%	2.9%
	20	3 318	796	3 598	1 210	3 626	1 108	-8%	8.7%	-1%	45.2%
	50	66 353	28 546	74 092	25 102	69 994	25 333	-12%	7.7%	6%	20.9%
	100	625 461	138 955	717 727	193 749	792 127	207 169	-15%	0.4%	-10%	3.3%
path	10	1 026	254	821	199	784	236	25%	0.0%	5%	19.8%
	20	5 314	1 445	3 654	574	3 254	732	45%	0.0%	12%	0.1%
	50	127 065	56 306	111 331	52 128	98 801	40 056	14%	7.5%	13%	9.0%
	100	1 188 204	358 886	1 166 542	387 468	1 234 315	296 382	2%	38.6%	-6%	16.4%

Struct.	n	PrimRST*		KruskalRST*		RandWalkRST*		$\delta_{\mathrm{P*,K*}}$	$\alpha_{\mathrm{P*,K*}}$	$\delta_{\mathrm{K*,R*}}$	$\alpha_{\mathrm{K*,R*}}$
		evals	*s*	*evals*	*s*	*evals*	*s*				
star	10	393	69	540	102	488	99	-37%	0.0%	11%	0.6%
	20	1 243	402	1 922	728	1 530	315	-55%	0.0%	26%	0.0%
	50	10 616	3 133	13 232	3 475	11 777	3 292	-25%	0.0%	12%	1.7%
	100	64 444	15 352	64 895	12 427	66 431	15 375	-1%	43.6%	-2%	29.2%
tree	10	507	121	513	131	486	108	-1%	41.2%	5%	13.8%
	20	1 485	395	1 527	325	1 714	545	-3%	28.2%	-12%	2.0%
	50	17 514	5 890	15 514	4 764	16 479	4 458	13%	3.2%	-6%	14.9%
	100	101 661	26 280	108 767	26 427	98 116	25 471	-7%	9.0%	11%	2.1%
path	10	531	109	461	110	471	89	15%	0.1%	-2%	67.9%
	20	1 479	504	1 386	383	1 503	473	7%	15.1%	-8%	8.8%
	50	16 559	4 778	16 841	4 435	19 162	7 327	-2%	38.0%	-14%	2.9%
	100	129 456	35 014	124 792	43 770	110 784	25 648	4%	27.9%	13%	2.7%

t-tests of the hypotheses that differences exist; we may conclude with high confidence only that PrimRST is the best choice when searching for star-like trees.

Thus the EA is robust with respect to the probabilities associated with spanning trees during initialization and recombination. We have investigated this robustness in more detail [53] and concluded that the probabilities associated with edges play a much more significant role than do the probabilities associated with trees. If the underlying graph is complete, edges are equally likely regardless of which of the three RST algorithms is used.

The lower half of Table 2 summarizes the trials in which the EA used the *-variants of recombination, which favor edges that appear in both parents. Differences among these versions of the EA were small and similar to the differences among the non-* recombination operators. However, compared to those operators, the *-variants enabled the EA to identify target trees using substantially fewer evaluations. For example, when targets were general random trees on 100 nodes, the EA required only about 15% as many evaluations to find them with the *-recombinations as it did with the non-* operators.

We conclude that the three random spanning tree algorithms are about equally effective in this EA, and that recombination should conserve edges found in both parents. The bias toward stars of PrimRST and KruskalRST is unimportant compared to the good locality and heritability of the edge-set coding of spanning trees and the operators applied to it.

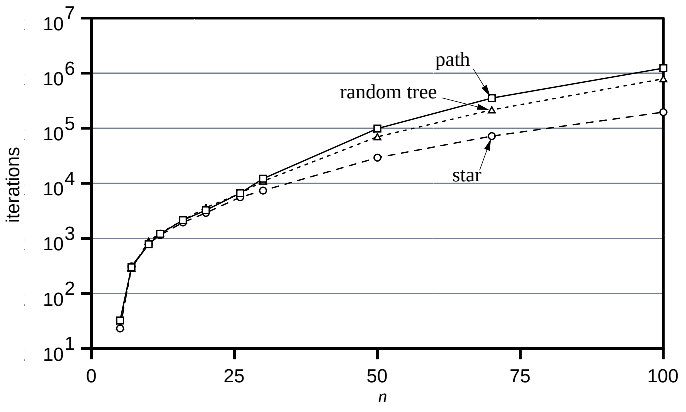


Figure 7: Average numbers of iterations needed by the EA with RandWalkRST to solve the One-Max-Tree problem with stars, random trees, or paths as target.

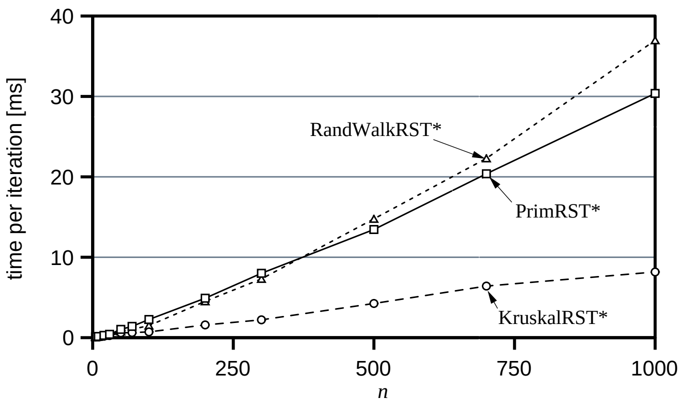


Figure 8: Average CPU times for a single iteration of the EA including selection, crossover based on either PrimRST*, KruskalRST*, or RandWalkRST*, and mutation.

Another important comparison between the three random spanning tree algorithms is the time they require, particularly in recombination. We have seen that PrimRST and KruskalRST can be implemented in time that is essentially linear in the number n of nodes, but RandWalkRST has expected time that is $O(n \log n)$ for most graphs and $O(n^3)$ for some cases, and its time is unbounded in the worst case.

Figure 8 plots the average CPU time per iteration through 10 000 iterations of the EA using PrimRST*, KruskalRST*, and RandWalkRST* in recombination against the number of nodes in the graph. The curves for PrimRST* and KruskalRST* confirm their linear times. They also illustrate the larger constant in the time of PrimRST*, which uses more complex data structures than does KruskalRST*, in particular a temporary array of adjacency lists. The computational effort of RandWalkRST* is slightly greater than linear, but still reasonable. In our implementation, only for $n > 350$ does RandWalkRST*'s time exceed that of PrimRST*, and the difference is still less than 20% when $n = 1\,000$. All three *-variants of the EA scale well to larger problem instances; KruskalRST* holds an advantage due to its simpler data structures.

For problems computationally more challenging than One-Max-Tree, we suggest considering all three random spanning tree algorithms and choosing the one that can best be adapted to a problem's constraints and hybridized with local heuristics, as the following section demonstrates using the degree-constrained minimum spanning tree problem.

5 The Degree-Constrained Minimum Spanning Tree Problem

In a graph, the degree of a node is the number of edges adjacent to it, and the degree of the graph is the maximum degree of its nodes. In a weighted, undirected graph G, consider all the spanning trees whose degrees do not exceed a bound $d \geq 2$. Among these, a tree of minimum cost is a degree-constrained minimum spanning tree (d-MST); the degree-constrained minimum spanning tree problem seeks such a tree. That is, if $c(e)$ is the cost of an edge $e \in E$, it seeks a spanning tree $T \subseteq E$ that minimizes

$$C = \sum_{e \in T} c(e),$$

subject to the constraint

$$\text{degree}(T) \leq d.$$

This problem is clearly NP-hard: When $d = 2$, it becomes the search for a Hamiltonian path of minimum cost [54, p. 206].

Nevertheless, when the nodes are points in the plane and the edge costs are the Euclidean distances between them, the problem is relatively easy. In this case, there always exists an unconstrained minimum spanning tree of degree no more than five [55]. Finding a d-MST in the plane is NP-hard when $d = 3$ and is conjectured to remain so when $d = 4$ [56]. Branch-and-bound techniques can find exact solutions for problem instances of several hundred points in reasonable computing times [6, 18], and several authors have described effective polynomial-time approximation schemes and heuristics [57, 58].

In general, however, edge costs need not satisfy the triangle inequality. An unconstrained minimum spanning tree may have degree up to $n - 1$, and when a graph has a high-degree unconstrained MST, identifying a d-MST for it is usually hard. Even for graphs of moderate size, exact algorithms and many heuristics are slow, and evolutionary algorithms are useful.

As in most previous work on the d-MST problem, we consider the case in which the underlying graph G is complete. However, our approach can be modified for incomplete graphs.

5.1 Satisfying the Degree Constraint

An EA employing the edge-set representation can guarantee that every tree it generates satisfies the degree constraint if we appropriately modify the initialization, recombination, and mutation operators.

To create spanning trees of degree no more than d for the algorithm's initial population, each of PrimRST, KruskalRST, and RandWalkRST accepts a newly-chosen edge only if its inclusion does not violate the degree constraint. Since the underlying graph is complete, this does not prevent each algorithm from always returning a spanning tree. Checking the

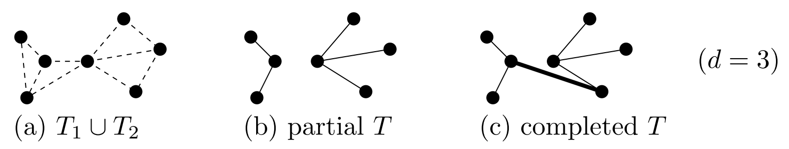


Figure 9: Recombination with KruskalRST for the 3-MST problem: The partial solution in (b) cannot feasibly be completed with parental edges shown in (a). Therefore, an edge from $E - (T_1 \cup T_2)$ must be included (c).

feasibility of each new edge can be done quickly; it does not increase the time complexity of the random spanning tree algorithms.

For recombination, the random spanning tree algorithms are modified in the same way, but because of the degree constraint, it may become impossible to complete an offspring spanning tree using only parental edges. Figure 9 shows an example of this under KruskalRST.

In such a case, recombination must include non-parental edges. How this is done depends on the random spanning tree algorithm.

- Under PrimRST, two nodes are chosen at random, one from the nodes in the spanning tree T whose degrees are less than d and the other from the unconnected vertices. The edge connecting these two nodes is included in the set of eligible edges. With this modification, recombination can still be implemented in time that is linear in n.
- Under KruskalRST, after the parental edges have been exhausted, it is simple to examine the non-parental edges in random order until the offspring tree has been completed. The resulting algorithm's time is, however, $O(m) = O(n^2)$.

  A more efficient technique identifies the unconnected components of the incomplete solution. These are connected by repeatedly adding edges between random nodes from different components and with degrees less than the bound. This process requires time that is only $O(n)$.
- Under RandWalkRST, when the tree cannot be expanded with a parental edge, the walk proceeds from a random connected node of degree less than d to any unconnected node. This scheme requires keeping track of the unconnected nodes that may yet be connected to the tree via a feasible parental edge; depending on the implementation, maintaining this information can increase the algorithm's expected time considerably.

During mutation, it is straightforward to ensure that the new edge does not violate the degree constraint. If mutation inserts a new random edge before deleting one from the cycle it completes, the following two special cases must be considered.

- If the new edge violates the degree constraint at both of its end-nodes, the edge is not suitable for insertion and is therefore discarded; the selection of a new edge is repeated. The probability of this case is small, so it does not much affect performance.
- If the new edge violates the degree constraint at one of its end-nodes, the edge to be removed must be the other edge adjacent to this node in the cycle.

5.2 Heuristics Based on Edge-Costs

Including problem-specific knowledge in an EA often improves its performance. We consider here a technique that is useful for many spanning tree problems: Choose edges to be included in candidate spanning trees according to probabilities that are higher for edges of lower cost.

We have previously investigated empirically the probability that an edge appears in a near-optimal degree-constrained spanning tree as a function of the edge's cost-based rank [59]. On a complete graph of 100 nodes with several different cost structures, for example, it was consistently the case that about 98% of the edges in near-optimal trees with $d = 3$ had ranks less than $300 = 3n$; that is, they were among the cheapest 6.1% of all the edges in the graph. These results support the intuition that candidate solutions should favor low-cost edges. The following sections describe computationally efficient implementations of this favoritism in KruskalRST initialization, KruskalRST* recombination, and mutation via insertion-before-deletion.

Heuristic Initialization. To favor low-cost edges when generating the initial population, KruskalRST begins by sorting all the edges by their costs. It builds the population's first spanning tree by examining edges in sorted order; that is, in order of increasing costs. The remaining trees are created with less heuristic bias by randomly permuting the cheapest k edges in the sorted list, before KruskalRST scans it. The number of edges permuted increases with each tree according to the formula

$$k = \alpha(i-1)n/P,$$

where P is the population size, i is the index of the next tree ($i = 1, \ldots, P$), and α is a parameter that controls the heuristic bias, thus the diversity of the initial population.

Heuristic Recombination. To build an offspring spanning tree, KruskalRST* recombination includes edges common to both parents, examines in random order the remaining parental edges $E' = (T_1 \cup T_2) - (T_1 \cap T_2)$, and completes the offspring with non-parental edges if necessary. To favor low-cost edges, the operator chooses each next edge from E' in a 2-tournament with replacement; the contestant with lower cost is examined next. This technique is simple and efficient, it favors low-cost edges without excluding more expensive ones, and it does not increase the time complexity of the operator.

We have compared several similar heuristic recombination operators in EAs for the traveling salesman and d-MST problems [59]. While an absolutely greedy strategy that always selects the cheapest edge in E' performed slightly better on problem instances that were simple or small, tournament-based selection was better on large and misleading instances.

Heuristic Mutation. Mutation via insertion-before-deletion is also extended to favor the insertion of low-cost edges. The edge to be inserted can be selected via a tournament on $E - T$, but this set is large, so such a tournament must also be large to reliably return edges of low cost. In several experiments, the following strategy was more effective.

Sort the edges in E by their costs; then each edge has a rank, with ties broken randomly. Identify a rank, thus an edge, by sampling the random variable

$$R = \lfloor |\mathcal{N}(0, \beta\, n)| \rfloor \bmod m + 1\,,$$

where $\mathcal{N}(0, \beta\, n)$ is a normally distributed random variable with mean 0 and standard deviation $\beta\, n$. Note that the values of R are integers between 1 and $m = |E|$. The parameter

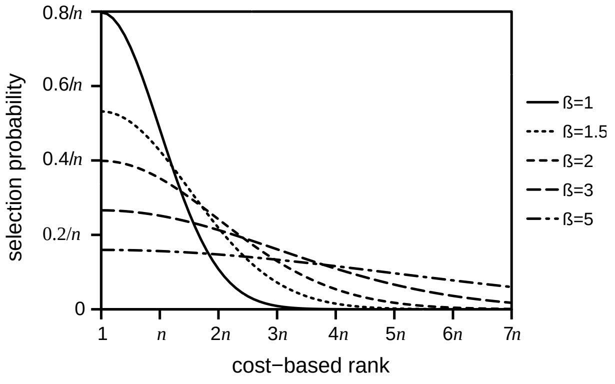


Figure 10: Heuristic mutation: Probability that an edge is selected for insertion as a function of its cost-based rank in E for β=1, 1.5, 2, 3, and 5.

β controls the strength of the scheme's bias towards low-cost edges; Figure 10 shows the probability density function of R for various values of β, assuming that $n \gg 1$ and $m \gg n$.

This scheme may select an edge already in T or an otherwise infeasible edge, but the probability that this will happen is small; such an edge is discarded and the selection repeated. If the graph's edges are sorted in a pre-processing step, this scheme can select each edge in constant expected time, and the expected time of mutation remains $O(n)$.

5.3 Empirical Comparisons

The edge-set representation of spanning trees, with and without the cost-based heuristics just described, was compared to three competing codings in a steady-state EA for the d-MST problem. When the algorithm represents candidate spanning trees by their edge-sets and non-heuristic operators are used, we identify it as ES-EA. The variant that includes edge-cost-based heuristics is called HES-EA. The competing codings represent spanning trees as strings via the Blob Code [32] (Section 2.3), as network random keys [36] (Section 2.5), and as strings of real-valued weights that influence a decoding algorithm [34] (Section 2.4). We identify the variants of the EA that use these codings as BC-EA, NRK-EA, and WE-EA, respectively.

All five variants initialize their populations with random solutions, according to their codings of spanning trees. They select parents from the population in binary tournaments with replacement. They create offspring via crossover with a probability of 80% and by always applying one mutation. Each offspring, if it does not duplicate an existing solution, replaces the worst solution in the population.

The algorithms' populations always contain 500 candidate trees. In HES-EA, the parameters α and β, which control the degree of heuristic bias in initialization and mutation, are both 1.5. BC-EA applies uniform crossover, position-by-position mutation, and to comply with the degree constraint, the initialization and repair strategies described by Zhou and Gen [7] for Prüfer numbers. NRK-EA uses uniform crossover and position-by-position mutation as suggested in [51]; the degree constraint is enforced by decoding solutions via a variant of Kruskal's algorithm that accepts only feasible edges. WE-EA selects the values in its initial solutions from a normal distribution, and generates offspring via uniform

Table 3: Average results on structured-hard problem instances for problem-space-search (PSS) and EAs employing the Blob Code (BC-EA), network random keys (NRK-EA), the weighted coding (WE-EA), and the edge-set representation (ES-EA), optionally including edge-cost-based heuristics (HES-EA).

Instance				PSS	BC-EA		NRK-EA		WE-EA		ES-EA			HES-EA		
name	n	d	C_{opt}	%-*gap*	%-*gap*	s	%-*gap*	s	%-*gap*	s	%-*gap*	s	*Opt*	%-*gap*	s	*Opt*
SHRD150	15	3	582	1.72	7.15	2.50	0.03	0.10	3.44	2.39	0.02	0.08	47	0.00	0.00	50
		4	430	2.79	12.07	4.26	0.00	0.00	4.64	0.07	0.00	0.00	50	0.00	0.00	50
		5	339	0.00	15.55	7.68	0.00	0.00	2.27	3.69	0.00	0.00	50	0.00	0.00	50
SHRD200	20	3	1 088	2.02	8.85	3.67	0.18	0.15	0.26	0.17	0.06	0.10	34	0.12	0.12	20
		4	802	2.00	11.96	4.67	0.17	0.35	0.18	0.13	0.04	0.11	39	0.03	0.05	39
		5	627	1.28	16.95	7.76	0.26	0.33	0.48	0.03	0.05	0.16	43	0.00	0.00	50
SHRD250	25	3	1 745	2.35	9.44	3.47	0.44	0.88	0.65	0.18	0.01	0.02	40	0.17	0.17	5
		4	1 276	1.10	15.20	7.14	0.42	0.57	0.62	0.33	0.05	0.08	31	0.05	0.09	35
		5	999	2.50	17.40	6.71	0.47	0.50	0.00	0.00	0.00	0.00	50	0.00	0.00	50
SHRD300	30	3	2 592	2.04	8.67	2.50	0.74	1.17	0.13	0.08	0.07	0.07	14	0.29	0.18	4
		4	1 905	1.63	14.57	6.32	1.02	1.45	0.10	0.05	0.12	0.15	20	0.09	0.22	33
		5	1 504	1.46	19.42	8.55	1.19	1.32	0.22	0.11	0.14	0.18	19	0.42	0.45	18

crossover and a mutation operator that resets weights to new normal random values, as described by Raidl and Julstrom [34]. All five variants stop when the best solution has not improved for 100 000 iterations.

Knowles and Corne [25] have observed that simple greedy heuristics can usually find near-optimal solutions to instances of the d-MST problem with Euclidean or randomly distributed edge costs. Similarly, HES-EA and WE-EA can in most cases identify optimal solutions to such problems with few evaluations. We compare the EA variants on more challenging instances that have high-degree unconstrained minimum spanning trees and low-cost edges chosen to mislead greedy heuristics.

From Krishnamoorthy *et al.* [18] we adopt the "structured hard" instance set, which contains four d-MST instances from 15 to 30 nodes with unconstrained MSTs of high degree. Here, the maximum degree d was set in turn to 3, 4, and 5, and each EA variant was run 50 times on each of the resulting twelve instances.

Table 3 shows the results of these trials, along with optimal values generated by a computationally expensive branch-and-cut algorithm, not yet published, and the previously published results of the problem-space-search EA (PSS) of Krishnamoorthy *et al.* [18].

A trial's percentage gap %-*gap* is the percentage by which the cost C of the trial's best tree exceeds the optimum cost C_{opt} of the problem instance:

$$\%\text{-}gap = \frac{C - C_{\text{opt}}}{C_{\text{opt}}} \cdot 100\%.$$

For each EA-variant on each problem instance, Table 3 displays the average %-*gap* over the corresponding trials and the standard deviation s of the gaps. For ES-EA and HES-EA, columns *Opt* show the number of trials that found an optimal spanning tree (out of 50 trials per instance).

Table 3 reveals clear differences in performance among the EA variants. The edge-set representation with and without heuristics (HES-EA and ES-EA) gave in general the best

Table 4: Average results on hard misleading instances with maximum degree $d = 5$ for Knowles and Corne's EA (KC-EA) and EAs using the Blob Code (BC-EA), network random keys (NRK-EA), the weighted coding (WE-EA), and the edge-set representation (ES-EA), optionally including edge-cost-based heuristics (HES-EA).

Instance			KC-EA	BC-EA		NRK-EA		WE-EA		ES-EA		HES-EA		
name	n	C_{opt}	*%-gap*	*%-gap*	s	*%-gap*	s	*%-gap*	s	*%-gap*	s	*%-gap*	s	*Opt*
M1	50	6.60	28.37	136.56	27.12	7.85	4.16	0.41	0.20	1.19	2.02	0.00	0.00	50
M2	50	5.78	35.18	163.28	33.08	11.75	5.90	9.01	3.46	1.45	2.97	<0.01	0.01	49
M3	50	5.50	9.54	133.75	25.94	5.40	2.80	16.20	5.88	0.92	1.65	0.00	0.00	50
M4	100	11.08	23.48	217.16	33.06	53.88	12.26	3.60	2.78	6.58	5.93	<0.01	0.01	49
M5	100	11.33	36.59	224.50	33.29	58.62	13.05	4.59	4.61	10.78	6.18	<0.01	0.01	49
M6	100	10.19	44.71	253.88	33.39	71.33	13.45	6.89	1.68	14.70	8.93	0.01	0.05	46
M7	200	18.33	13.96	296.69	31.97	146.60	13.54	0.34	0.14	17.64	6.36	0.02	0.04	31
M8	200	19.16	37.60	313.29	30.33	166.17	14.13	0.87	0.95	26.30	7.55	0.02	0.03	26
M9	200	16.13	2.06	313.17	32.13	153.89	13.24	0.36	0.57	9.78	3.76	0.01	0.01	15
M10	300	40.69*	–	175.48	15.26	89.67	5.07	13.71	2.93	19.99	3.09	0.06	0.04	1
M11	400	54.69*	–	212.72	22.78	137.93	4.74	20.34	2.73	37.84	3.84	1.00	0.93	1
M12	500	79.34*	–	171.53	12.90	117.38	3.82	29.09	1.23	34.04	3.02	1.09	0.74	1

results, followed by the weighted coding (WE-EA), network random keys (NRK-EA), problem space search (PSS), and the Blob Code (BC-ES). The latter's performance was decisively the worst. ES-EA and HES-EA returned similarly good results, which were the best on all the instances except SHRD300, where WE-EA sometimes performed slightly better. On SHRD250 with $d = 5$, all three, HES-EA, ES-EA, and WE-EA, found optimum solutions in all trials. On the remaining instances, of 15, 20, and 25 nodes, t-tests indicate the superiority of ES-EA and HES-EA to WE-EA at a significance level much smaller than 1%. NRK-EA performed as well as (H)ES-EA only on the smallest instance SHRD150, where it also identified optimum solutions most of the time. HES-EA was usually the fastest of the competing algorithms, though all terminated on average within 15 seconds on a Pentium-III/800MHz PC.

Because operations on edge-sets are fast, (H)ES-EA scales well to larger problem instances. We demonstrate this, and further compare the five EA-variants, on a set of larger "hard and misleading" d-MST instances. Nine of these instances, of 50, 100, and 200 nodes, were developed by Knowles and Corne [25]; we created three more instances with similar structure of 300, 400, and 500 nodes. These instances are hard for simple greedy algorithms because they contain low-cost edges that do not appear in optimum solutions. These edges mislead greedy heuristics—*e.g.*, Prim's MST algorithm modified to accept only edges that satisfy the degree constraint [6]—so that they produce only poor solutions. Knowles and Corne [25] describe the creation of such instances in detail. In all twelve instances, the degree constraint was set to $d = 5$. Each EA was run 50 times on each instance.

Table 4 reports the results of these trials and reprints the results of the EA of Knowles and Corne [25] (Section 2.6) on the first nine instances. For these instances, the costs of optimal trees were again determined by branch-and-cut and were used to compute the percentage gaps of the EA-results. For the remaining three instances, percentage gaps were computed in terms of the lowest costs observed in all the trials on them; these costs are marked by '*' in the table.

Differences in the algorithms' performances are readily apparent and without exception significant at a level far less than 1%. HES-EA always returned the best results; on instances

Table 5: Average numbers of evaluations and CPU times until the best solutions for runs on hard misleading problem instances. The runs of NRK-EA and WE-EA on M11 and M12 were terminated after 100 000 evaluations.

Instance		BC-EA		NRK-EA		WE-EA		ES-EA		HES-EA	
name	n	*evals*	t [s]	*evals*	t [s]	*evals*	t [s]	*evals*	t [s]	*evals*	t [s]
M1	50	122 332	17	59 133	74	26 732	36	68 534	17	5 331	2
M4	100	221 378	58	113 504	633	78 824	408	202 678	97	12 958	7
M7	200	386 840	222	200 625	5 188	98 551	2 373	475 084	466	16 415	20
M10	300	531 862	506	249 008	15 723	265 684	16 793	969 885	1 396	36 124	69
M11	400	665 009	905	99 166	14 179	99 519	12 631	1 321 372	2 585	47 290	140
M12	500	848 315	1 525	99 268	27 341	99 801	19 026	1 793 420	8 466	60 701	240

of up to 200 nodes it often found optimal trees, as column *Opt* documents. It is followed by WE-EA, ES-EA, the algorithm of Knowles and Corne (KC-EA) and, trailing far behind, NRK-EA and finally BC-EA. The superiority of HES-EA becomes more decisive as the instances become larger. Here, local heuristics based on edge-costs, as they appear in HES-EA and implicitly in WE-EA, are an obvious advantage.

In the experiments Table 4 summarizes, the trials of NRK-EA and WE-EA on the 400- and 500-node instances exceeded their allotted run-times and were terminated after a total of 100 000 evaluations. Table 5 shows the EAs' average numbers of evaluations and average CPU times until the best solutions have been identified on instances of six sizes. While the times of HES-EA, BC-EA, and ES-EA increase relatively slowly with n, the times WE-EA and NRK-EA require grow rapidly. Decoding strings of weights to degree-constrained spanning trees requires modifying all the edge costs and then applying a variant of Prim's algorithm that considers the degree-constraint; its time is $O(n^2 \log n)$. Similarly, sorting edges according to network random keys and applying Kruskal's algorithm requires $O(m \log m)$ time.

6 Conclusion

The most important factor in an evolutionary algorithm's performance is the interaction of its coding of candidate solutions with the operators it applies to them. We have proposed that spanning trees be represented directly as sets of their edges, and we have described initialization, recombination, and mutation operators for the edge-set encoding. These operators offer strong locality and heritability and computational efficiency.

For the initialization and recombination operators, three algorithms that generate random spanning trees were investigated. PrimRST and, to a lesser degree, KruskalRST are disproportionately likely to yield star-like trees, but RandWalkRST generates spanning trees with uniform probabilities. Nonetheless, tests using the One-Max-Tree problem indicate that an EA's performance depends little on this choice. Operators that use KruskalRST are slightly faster because of its simpler data structures, but the choice should be based on how easily operators can accommodate problem-specific constraints and be hybridized with other heuristics.

Recombination of spanning trees should always include in the offspring edges that are common to both parents. Our studies indicate the clear benefits of this strategy, regardless of the underlying random spanning tree algorithm, target tree structure, or problem size.

The edge-set representation was applied in an EA for the degree-constrained minimum spanning tree problem. With small modifications, the initialization, recombination, and mutation operators all efficiently maintained the degree constraint, and they were easily extended to accommodate heuristics that probabilistically prefer low- to high-cost edges. These heuristics substantially improve the EA's performance.

Tests on two sets of hard d-MST problem instances indicate the superiority of edge-sets, particularly when the variation operators implement edge-cost-based heuristics, to several other codings of spanning trees—the Blob Code, network random keys, and strings of weights—in evolutionary search. The edge-set-coded EA identified better solutions on most instances, and it scaled up more efficiently to larger problem sizes.

Acknowledgments

The authors would like to thank the anonymous reviewers and David Fogel for their their valueable comments on a previous version of this article. This work has been supported by the Austrian Science Fund (FWF) under the grant P13602–INF.

References

[1] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematics Society*, vol. 7, no. 1, pp. 48–50, 1956.

[2] R. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.

[3] Bernard Chazelle, "A minimum spanning tree algorithm with inverse-Ackermann type complexity," *Journal of the Association for Computing Machinery*, vol. 47, no. 6, pp. 1028–1047, 2000.

[4] T. C. Hu, "Optimum communication spanning trees," *SIAM Journal of Computing*, vol. 3, pp. 188–195, 1974.

[5] M. L. Gargano, W. Edelson, and O. Koval, "A genetic algorithm with feasible search space for minimal spanning trees with time-dependent edge costs," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Hitoshi Iba, and Rick L. Riolo, Eds. 1998, p. 495, Morgan Kaufmann.

[6] S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning trees," *Computers and Operations Research*, vol. 7, pp. 239–249, 1980.

[7] G. Zhou and M. Gen, "Genetic algorithms on leaf-constrained minimum spanning tree problem," *Beijing Mathematics*, vol. 7, no. 2, pp. 50–62, 1998.

[8] L. R. Esau and K. C. Williams, "On teleprocessing system design," *IBM Systems Journal*, vol. 5, pp. 142–147, 1966.

[9] C. H. Papadimitriou, "The complexity of the capacitated tree problem," *Networks*, vol. 8, pp. 217–230, 1978.

[10] M. Hanan, “On Steiner’s problem with rectilinear distance,” *SIAM Journal of Applied Mathematics*, vol. 14, no. 2, pp. 255–265, 1966.

[11] G. M. Guisewite and P. M. Pardalos, “Minimum concave-cost network flow problems: Applications, complexity and algorithms,” *Annals of Operations Research*, vol. 25, pp. 75–100, 1990.

[12] M. Gen and Yinzhen Li, “Spanning tree-based genetic algorithm for the bicriteria fixed charge transportation problem,” In Angeline et al. [60], pp. 2265–2271.

[13] L. Davis, D. Orvosh, A. Cox, and Y. Qiu, “A genetic algorithm for survivable network design,” in *Proceedings of the Fifth International Conference on Genetic Algorithms*, Stephanie Forrest, Ed. 1993, pp. 408–415, Morgan Kaufmann.

[14] P. Piggott and F. Suraweera, “Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem,” in *Progress in Evolutionary Computation*, Xin Yao, Ed., LNAI 956, pp. 305–314. Springer, 1995.

[15] A. Cayley, “A theorem on trees,” *Quarterly Journal of Mathematics*, vol. 23, pp. 376–378, 1889.

[16] Les Berry, Bruce Murtagh, and Steve Sugden, “A genetic-based approach to tree network synthesis with cost constraints,” in *Proceedings of the Second European Congress on Intelligent Techniques and Soft Computing*, H. Zimmermann, Ed., Aachen, Germany, 1994, pp. 626–629.

[17] C. C. Palmer and A. Kershenbaum, “Representing trees in genetic algorithms,” in *Proceedings of the First IEEE Conference on Evolutionary Computation*, David Schaffer, Hans-Paul Schwefel, and David B. Fogel, Eds. 1994, pp. 379–384, IEEE Press.

[18] Mohan Krishnamoorthy and Andreas T. Ernst, “Comparison of algorithms for the degree constrained minimum spanning tree,” *Journal of Heuristics*, vol. 7, pp. 587–611, 2001.

[19] F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld, “Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem,” in *Proceedings of the Sixth International Conference on Genetic Algorithms*, Larry J. Eshelman, Ed. 1995, pp. 470–477, Morgan Kaufmann.

[20] C.-H. Chu, G. Premkumar, C. Chou, and J. Sun, “Dynamic degree constrained network design: A genetic algorithm approach,” in *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, Eds. 1999, pp. 141–148, Morgan Kaufmann.

[21] Les Berry, Bruce Murtagh, Steve Sugden, and Graham McMahon, “Application of a genetic-based algorithm for optimal design of tree-structured communication networks,” in *Proceedings of the Regional Teletraffic Engineering Conference of the International Teletraffic Congress*, South Africa, 1995, pp. 361–370.

[22] Hsinghua Chou, G. Premkumar, and Chao-Hsien Chu, “Genetic algorithms for communications network design—An empirical study of the factors that influence performance,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, pp. 236–249, 2001.

[23] S. Even, *Algorithmic Combinatorics*, The Macmillan Company, New York, 1973.

[24] H. Prüfer, "Neuer Beweis eines Satzes über Permutationen," *Archiv für Mathematik und Physik*, vol. 27, pp. 141–144, 1918.

[25] J. Knowles and D. Corne, "A new evolutionary approach to the degree constrained minimum spanning tree problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 2, pp. 125–134, 2000.

[26] Franz Rothlauf and David Goldberg, "Pruefernumbers and genetic algorithms: A lesson how the low locality of an encoding can harm the performance of GAs," in *Parallel Problem Solving from Nature – PPSN VI*, Kalyanmoy Deb, Günther Rodolph, Xin Yao, and Hans-Paul Schwefel, Eds. 2000, vol. 1917 of *LNCS*, pp. 395–404, Springer.

[27] G. Zhou and M. Gen, "Approach to degree-constrained minimum spanning tree problem using genetic algorithm," *Engineering Design & Automation*, vol. 3, no. 2, pp. 157–165, 1997.

[28] J. R. Kim and M. Gen, "Genetic algorithm for solving bicriteria network topology design problem," In Angeline et al. [60], pp. 2272–2279.

[29] Jens Gottlieb, Bryant A. Julstrom, Günther R. Raidl, and Franz Rothlauf, "Prüfer numbers: A poor representation of spanning trees for evolutionary search," in *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, Lee Spector, Erik Goodman, Annie Wu, W. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max Garzon, and Edmund Burke, Eds. 2000, pp. 343–350, Morgan Kaufmann.

[30] Sally Picciotto, *How to Encode a Tree*, Ph.D. thesis, University of California, San Diego, 1999.

[31] Narsingh Deo and Paulius Micikevicius, "Comparison of Prüfer-like codes for labeled trees," in *32. Southeastern International Conference on Combinatorics, Graph Theory, and Computing*, Baton Rouge, LA, 2001.

[32] Bryant A. Julstrom, "The Blob Code: A better string coding of spanning trees for evolutionary search," in *2001 Genetic and Evolutionary Computation Conference Workshop Program*, Robert Heckendorn, Ed., San Francisco, CA, 2001, pp. 256–261.

[33] Thomas Gaube and Franz Rothlauf, "The link and node biased encoding revisited: Bias and adjustment of parameters," In Boers et al. [61], pp. 11–19.

[34] Günther R. Raidl and Bryant A. Julstrom, "A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 ACM Symposium on Applied Computing*, Janice Carroll, Ernesto Damiani, Hisham Haddad, and Dave Oppenheim, Eds. 2000, pp. 440–445, ACM Press.

[35] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[36] Franz Rothlauf, David Goldberg, and Armin Heinzl, "Bad codings and the utility of well-designed genetic algorithms," in *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, Eds. 2000, pp. 355–362, Morgan Kaufmann.

[37] Franz Rothlauf, *Representations for Genetic and Evolutionary Algorithms*, Studies in Fuzziness and Soft Computing. Physica, Heidelberg, 2002.

[38] Barbara Schindler, Franz Rothlauf, and Hans-Josef Pesch, "Evolution strategies, network random keys, and the One-Max Tree problem," in *Applications of Evolutionary Computing: EvoWorkshops 2002*, Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther R. Raidl, Eds. 2002, vol. 2279 of *LNCS*, pp. 143–152, Springer.

[39] M. Gen, G. Zhou, and J. R. Kim, "Genetic algorithms for solving network design problems: State-of-the-art survey," *Evolutionary Optimization*, vol. 1, no. 2, pp. 121–141, 1999.

[40] William Edelson and Michael Gargano, "Feasible encodings for GA solutions of constrained minimal spanning tree problems," In Armstrong [62], pp. 82–89.

[41] Günther R. Raidl, "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, Carlos Fonseca, Jong-Hwan Kim, and Alice Smith, Eds. 2000, pp. 104–111, IEEE Press.

[42] Bryant A. Julstrom, "Encoding rectilinear Steiner trees as lists of edges," In Lamont et al. [63], pp. 356–360.

[43] Y. Li and Y. Bouchebaba, "A new genetic algorithm for the optimal communication spanning tree problem," in *Proceedings of Artificial Evolution: Fourth European Conference*, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Edmund Ronald, and Marc Schoenauer, Eds. 1999, vol. 1829 of *LNCS*, pp. 162–173, Springer.

[44] Y. Li, "An effective implementation of a direct spanning tree representation in GAs," In Boers et al. [61], pp. 11–19.

[45] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

[46] Günther R. Raidl and Christina Drexel, "A predecessor coding in an evolutionary algorithm for the capacitated minimum spanning tree problem," In Armstrong [62], pp. 309–316.

[47] A. Guénoche, "Random spanning tree," *Journal of Algorithms*, vol. 2, pp. 214–220, 1983.

[48] Charles J. Colbourne, Robert P. J. Day, and Louis D. Nel, "Unranking and ranking spanning trees of a graph," *Journal of Algorithms*, vol. 10, no. 2, pp. 249–270, 1989.

[49] Andrei Broder, "Generating random spanning trees," in *IEEE 30th Annual Symposium on Foundations of Computer Science*. 1989, pp. 442–447, IEEE.

[50] D. H. Ackley, *A Connectionist Machine for Genetic Hill Climbing*, Kluwer Academic Press, Boston, 1987.

[51] Franz Rothlauf, David Goldberg, and Armin Heinzl, "Network random keys – a tree network representation scheme for genetic and evolutionary algorithms," *Evolutionary Computation*, vol. 10, no. 1, pp. 75–97, 2002.

[52] Franz Rothlauf and David Goldberg, "Tree network design with genetic algorithms – An investigation in the locality of the Pruefernumber encoding," in *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, Scott Brave and Annie S. Wu, Eds., Orlando, FL, 1999, pp. 238–243.

[53] B. A. Julstrom and G. R. Raidl, "Initialization is robust in evolutionary algorithms that encode spanning trees as sets of edges," in *Proceedings of the 2002 ACM Symposium on Applied Computing*, Gary Lamont et al., Eds. 2002, pp. 547–552, ACM Press.

[54] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[55] C. Monma and S. Suri, "Transitions in geometric minimum spanning trees," *Discrete and Computational Geometry*, vol. 8, no. 3, pp. 265–293, 1992.

[56] C. H. Papadimitriou and U. V. Vazirani, "On two geometric problems related to the traveling salesman problem," *Journal of Algorithms*, vol. 5, pp. 231–246, 1984.

[57] S. Fekete, S. Khuller, M. Klemmstein, B. Raghavachari, and N. Young, "A network-flow technique for finding low-weight bounded-degree spanning trees," *Journal of Algorithms*, vol. 24, pp. 310–324, 1997.

[58] M. Savelsbergh and T. Volgenant, "Edge exchanges in the degree-constrained minimum spanning tree problem," *Computers and Operations Research*, vol. 12, no. 4, pp. 341–348, 1985.

[59] Bryant A. Julstrom and Günther R. Raidl, "Weight-biased edge-crossover in evolutionary algorithms for two graph problems," In Lamont et al. [63], pp. 321–326.

[60] Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, Ali Zalzala, and William Porto, Eds., *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*. IEEE Press, 1999.

[61] Egbert J.W. Boers, Stefano Cagnoni, Jens Gottlieb, Emma Hart, Pier Luca Lanzi, Günther R. Raidl, Robert E. Smith, and Harald Tijink, Eds., *Applications of Evolutionary Computation*, vol. 2037 of *LNCS*. Springer, 2001.

[62] Chris Armstrong, Ed., *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, Las Vegas, NV, 2000.

[63] Gary Lamont, Janice Carroll, Hisham Haddad, Don Morton, George Papadopoulos, Richard Sincovec, and Angelo Yfantis, Eds., *Proceedings of the 16th ACM Symposium on Applied Computing*. ACM Press, 2001.

Proof that $p_{\text{path}} < p_{\text{unif}}$ for PrimRST

We show by induction that

$$p_{\text{path}} = \frac{2^{n-1}}{(n-1)!\,n!} \; < \; p_{\text{unif}} = \frac{1}{n^{n-2}} \quad \text{for } n \geq 4\,. \tag{1}$$

This inequality is trivially true for $n = 4$.

We assume it holds for any $n \geq 4$ and derive its validity for $n+1$:

$$\frac{2^n}{n!\,(n+1)!} < \frac{1}{(n+1)^{n-1}} \iff \underbrace{\frac{2}{n(n+1)}}_{A} \cdot \underbrace{\frac{2^{n-1}}{(n-1)!\,n!} < \frac{1}{n^{n-2}}}_{\text{assumption}} \cdot \underbrace{\frac{n^{n-2}}{(n+1)^{n-1}}}_{B} \tag{2}$$

Since

$$\frac{A}{B} = \frac{2(n+1)^{n-2}}{n^{n-1}} = \frac{2n}{(n+1)^2}\left(1+\frac{1}{n}\right)^n$$

and the infinite sequence $\{(1+1/n)^n\}$ is monotonically increasing with n and converges to Euler's number e, it follows that

$$\frac{A}{B} < \frac{2e \cdot n}{(n+1)^2} < 1 \quad (\text{for } n \geq 4)$$

and inequalities (2) and (1) are true. $\square$