Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search

Jens Gottlieb, **Bryant A. Julstrom**, **Günther R. Raidl** and **Franz Rothlauf**

Working Paper 12/2000
December 2000

Working Papers in Information Systems
Editor: Prof. Dr. Armin Heinzl

University of Bayreuth
Department of Information Systems
Universitaetsstrasse 30
D-95440 Bayreuth, Germany
Phone +49 921 552807, Fax +49 921 552216
E-Mail: wi@uni-bayreuth.de
Internet: *h*ttp://wi.oec.uni-bayreuth.de

Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search

Jens Gottlieb
SAP AG
Neurottstr. 16
69190 Walldorf, Germany
jens.gottlieb@sap.com

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
720 Fourth Avenue South
St. Cloud, MN 56301, USA
julstrom@eeyore.stcloudstate.edu

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

Franz Rothlauf*
Department of Information Systems
University of Bayreuth
Universitaetsstr. 30
D-95440 Bayreuth/Germany
rothlauf@uni-bayreuth.de

Abstract

The most important element in the design of a decoder-based evolutionary algorithm is its genotypic representation. The genotype-decoder pair must exhibit efficiency, locality, and heritability to enable effective evolutionary search. Prüfer numbers have been proposed to represent spanning trees in evolutionary algorithms. Several researchers have made extravagant claims for the usefulness of this coding, but others have pointed out that Prüfer numbers, though concise and easy to decode, lack the essential properties of locality and heritability. This conflict motivates our study. We examine the properties of Prüfer numbers and compare Prüfer numbers with other codings in evolutionary algorithms for four problems that involve spanning trees. Our conclusion is definite: Prüfer numbers cause poor performance in evolutionary algorithms and should be avoided.

1 Introduction

An evolutionary algorithm (EA) maintains a population of data structures that represent candidate solutions to a problem. In a *decoder-based* EA, each data structure can be thought of as providing instructions to a decoder that builds the solution the structure represents. The data structure is the solution's *genotype*; the decoded solution is its *phenotype*.

In EAs for problems of combinatorial optimization, the decoder can consider problem-specific information such as constraints; the EA's author then need not design complex evolutionary operators or penalty functions. The decoder should be fast, and the coding implemented in the genotype-decoder pair should exhibit *locality* and *heritability*: small changes in genotypes should correspond to small changes in the solutions they represent, and solutions generated by crossover should combine features of their parents.

Decoder-based EAs have been applied to problems that search spaces of spanning trees. Finding an unconstrained minimum spanning tree is easy, but many problems involving spanning trees are computationally hard and thus suitable targets for heuristics such as EAs. *Prüfer numbers* offer a deceptively elegant coding of spanning trees whose decoding algorithm is

*Also with Illinois Laboratory of Genetic Algorithms, University of Illinois at Urbana-Champaign, USA.

found in a constructive proof of Cayley's formula.

This formula identifies the number of unconstrained spanning trees in a complete undirected graph on n nodes as n^{n-2} (Cayley, 1889; Even, 1973, pp.98–106). Prüfer (1918) described a one-to-one mapping between spanning trees on n nodes and strings of $n-2$ node labels. Conventionally, the integers from 1 to n label the nodes, and the strings of labels are called Prüfer numbers.

Prüfer numbers thus encode spanning trees, and researchers have used Prüfer numbers in EAs for problems that search spaces of spanning trees. These have included the probabilistic minimum spanning tree problem (Abuali et al., 1994), the degree-constrained minimum spanning tree problem (Zhou and Gen, 1997), the time-dependent minimum spanning tree problem (Gargano et al., 1998), the fixed-charge transportation problem (Li et al., 1998) and a bicriteria version of it (Gen and Li, 1999), and a multi-objective network design problem (Kim and Gen, 1999). Some authors have made extravagant claims for the efficacy of this coding, including Kim and Gen, who wrote

> The Prüfer number is very suitable for encoding a spanning tree, especially in some research fields, such as transportation problems, minimum spanning problems, and so on.

Such claims are in error. In general, Prüfer numbers do not support effective evolutionary search. Alternate codings of spanning trees consistently provide better results, as we show in four examples.

This paper describes Prüfer numbers and their properties and compares Prüfer numbers with other codings of spanning trees in EAs for four problems: the degree-constrained spanning tree problem, the communication spanning tree problem, the rectilinear Steiner problem, and the fixed-charge transportation problem. In every case, evolutionary search is more effective with other codings than with Prüfer numbers.

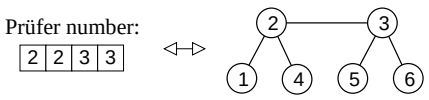

Figure 1: A spanning tree and its Prüfer number.

2 The Prüfer Number Representation

This section describes the algorithm that decodes a Prüfer number to a spanning tree, examines the locality and heritability of the Prüfer coding under conventional evolutionary operators, and finds that the region of the search space where a Prüfer-coded EA might perform well is vanishingly small.

2.1 Decoding Prüfer Numbers

Let $a_1 a_2 \cdots a_{n-2} \in \{1, \ldots, n\}^{n-2}$ be a Prüfer number. In the corresponding spanning tree, each node's degree is one more than the number of times the node's label appears. To identify the spanning tree's edges:

1. Scan the Prüfer number to identify each node's degree. Initialize a variable i to 1.
2. Find the node v of degree 1 with the smallest label. (v, a_i) is a spanning tree edge.
3. Decrement the degrees of v and a_i; increment i.
4. Repeat steps (2) and (3) until all nodes have degree 0, except two with degree 1. These form the spanning tree's last edge.

An efficient implementation of this algorithm uses a priority queue implemented in a heap to hold the nodes of degree 1. The algorithm's time complexity is then $O(n \log n)$. Figure 1 shows a Prüfer number of length four and the tree on six nodes to which it decodes.

Several features of Prüfer numbers suggest that they might support efficient evolutionary search of spaces of spanning trees. They can be decoded quickly; it is easy to generate random Prüfer numbers by choosing $n-2$ times from n node labels; and Prüfer numbers support conventional evolutionary operators like

k-point crossover and position-by-position mutation. However, Prüfer numbers' poor locality and heritability are not conducive to evolutionary search, as the remainder of this section demonstrates.

2.2 Locality

A coding has high locality if mutating a genotype changes the corresponding phenotype only slightly. Several researchers, including Palmer and Kershenbaum (1994) and Rothlauf and Goldberg (2000), have pointed out the poor locality of Prüfer numbers with respect to conventional position-by-position mutation.

In general, Prüfer numbers related by such mutations do not represent similar spanning trees. For example, changing the last digit in the Prüfer number of Figure 1 from 3 to 1 yields 2231, which decodes to the edges (2,4), (2,5), (3,2), (1,3), and (1,6). Only two of the original tree's five edges remain.

Some Prüfer numbers do have high locality. A star is a spanning tree in which every node but one is a leaf. A star on n nodes has $(n-1)(n-2)$ neighbors, obtained by replacing one of its edges with another feasible edge. In a star's Prüfer number, all the symbols are the same. A star's neighbors are represented by the neighbors of its Prüfer number, obtained by changing one of the number's symbols; these neighbors also number $(n-1)(n-2)$. For stars, the genotypic and phenotypic neighborhoods coincide, and locality is maximal.

This seems auspicious, but Prüfer numbers' localities vary with the shapes of the trees they represent. A list is a spanning tree with two leaves and $n-2$ nodes of degree 2. In a list's Prüfer number, all the symbols are distinct, and a list on n nodes has $\frac{1}{6}n(n-1)(n+1)-n+1$ neighbors. Stars and lists have the smallest and largest phenotypic neighborhoods, respectively. All other spanning trees fall between these extremes, which Figure 2 plots as a function of the number n of nodes, and random trees are in general more similar to lists than to stars.

Figure 3 illustrates the low locality of most Prüfer numbers. For it, we examined the neighborhoods of 1 000 spanning trees—stars, arbitrary trees, and lists—on $n = 16$ nodes. Figure 3(a) shows distributions of genotypic distances for neighboring spanning trees; that is, for spanning trees that differ in one edge. Figure 3(b) shows distributions of phenotypic distances for neighboring Prüfer numbers; that is, for Prüfer numbers that differ in one digit. Only for stars and trees similar to stars is the locality of the Prüfer coding high. In general, the phenotypes of genotypic neighbors are very different, and conversely.

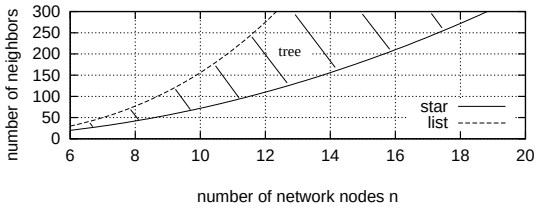

Figure 2: Phenotypic neighborhood sizes for lists and stars, as functions of the number of nodes. The values for all other trees lie between these curves.

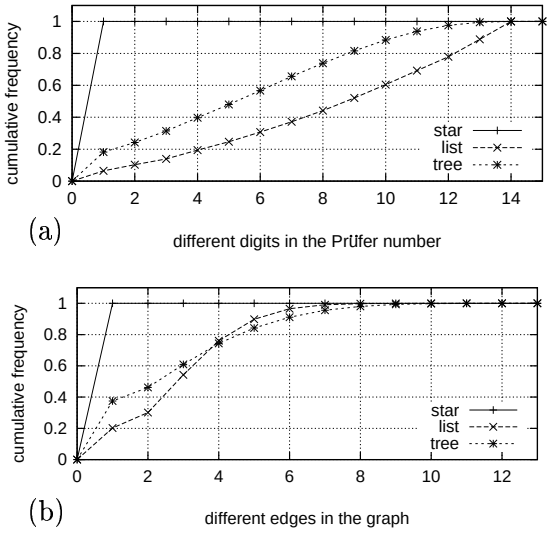

Figure 3: Distributions of (a) genotypic distances for neighboring spanning trees, and (b) phenotypic distances for neighboring Prüfer numbers, on 16 nodes.

2.3 Heritability

A coding has high heritability, with respect to a crossover operator, if offspring phenotypes consist mostly of substructures of their parents' phenotypes. When genotypes encode spanning trees, offspring trees should consist mostly or entirely of parental edges. Usually, with conventional operators, heritability will be low where locality is low. In Prüfer numbers, the meanings of genotypic symbols depend on their contexts, and k-point crossover will not preserve parental substructures in offspring phenotypes. Thus Prüfer-coded EAs using such a crossover will search effectively only near stars, where locality is high.

Experiments with a simple problem confirm that this is so. In the One-Max problem, a bit string's fitness is the number of 1s in it. In the One-Max-Tree problem (Rothlauf et al., 2000), an optimum spanning tree is specified, and the fitness of any tree is the number of edges that it shares with this target. An EA whose coding has sufficient heritability should solve this problem easily, by preserving edges of the target tree from parents to offspring.

A generational EA for One-Max-Tree encoded spanning trees as Prüfer numbers. It applied no mutation, only one-point crossover, and used $(\mu + \lambda)$ selection, with $\mu = \lambda = 1500$. Figure 4 illustrates the algorithm's performance on several One-Max-Tree instances with $n = 32$ nodes, whose target trees were variously stars, random trees, and lists. Only when the optimum tree's locality was high—*i.e.*, it was a star—did the EA find it. In other cases, the EA's search failed, and failed badly.

2.4 Where Locality is High

An EA will search effectively in regions of high locality, but when Prüfer numbers encode spanning trees, these regions are tiny. We extend the definition of neighbors to include trees whose Prüfer numbers differ in at most $i_{\max}$ digits ($i_{\max} \ll n$). The number of neighbors of stars is then $\sum_{i=0}^{i_{\max}} \binom{n-2}{i}(n-1)^i$; this value is $O(n^{2i_{\max}})$. The number of spanning trees on n vertices, and thus the size of the search space, is n^{n-2}.

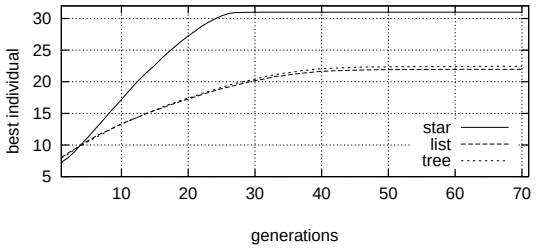

Figure 4: Performance of a Prüfer-coded EA on One-Max-Tree problems of 32 nodes, with optimum trees of various structures.

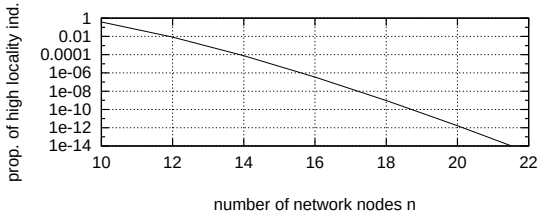

Figure 5: The proportion of spanning trees on n nodes whose Prüfer numbers have high locality, defined as differing from the Prüfer number of a star in no more than $i_{\max} = 5$ digits.

As Figure 5 illustrates, the proportion of these high-locality trees is small even for moderate n and diminishes exponentially as n grows. On problems of interesting size, Prüfer-coded EAs cannot succeed.

The following four sections compare codings of spanning trees in EAs for four substantive problems. They confirm the unsuitability of the Prüfer coding.

3 The Degree-Constrained Minimum Spanning Tree Problem

Given a weighted undirected graph G, the degree-constrained minimum spanning tree problem (d-MSTP) seeks a spanning tree on G of minimum weight whose degree does not exceed $d > 1$. This problem is NP-hard (Garey and Johnson, 1979, p. 206). Figure 6 shows an unconstrained spanning tree of degree 5 and a spanning tree with maximum degree $d = 3$ on $n = 11$ points in the plane.

The degree constraint suggests that Prüfer numbers might perform well in an EA for this

problem, since 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. Thus, it is easy to identify and repair Prüfer numbers whose trees violate the constraint.

Zhou and Gen (1997) presented an EA for the d-MSTP that encodes spanning trees as Prüfer numbers. However, it was tested only on very small, random problem instances. These were shown to be easily solved to optimality by greedy heuristics, branch-and-bound, or EAs using other encodings (Knowles and Corne, 2000; Krishnamoorthy et al., 1999; Raidl, 2000).

Other researchers have used other codings. Knowles and Corne (2000) described an EA in which genotypes are strings of integers that influence the order in which a variation of Prim's minimum spanning tree algorithm connects nodes to the spanning tree.

Palmer and Kershenbaum (1994) encoded spanning trees as strings of real-valued weights. The tree such a genotype represents is found by temporarily adding each node's weight to all the distances in which it participates, then applying Prim's algorithm to the modified distances. Raidl and Julstrom (2000) adapted this coding to the d-MSTP, using normally distributed, multiplicative weights.

Krishnamoorthy et al. (1999) compared two Prüfer-coded EAs, which differed in their crossover operators and breeding schemes, to a weight-coded EA and other optimization techniques. The Prüfer-coded EAs failed on all instances except the simplest; on average, the weight-coded EA performed best.

Recently, Raidl (2000) described an EA for the d-MSTP that stores the edges of each candidate spanning tree directly in lists. The recombination operator uses parental edges to build a new spanning tree; to avoid violating the degree constraint it must occasionally introduce edges that appear in neither parent. Mutation inserts a new edge and then removes an edge from the cycle so created. This approach was both effective and fast.

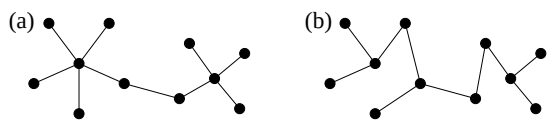

Figure 6: An unconstrained spanning tree (a) and a spanning tree with maximum degree $d = 3$ (b).

Table 1: Average weights of the best spanning trees found on 20 trials with four codings on six hard 5-MSTP instances.

5-MSTP	n	PR	K&C	Wts	LoE
`m050n1`	50	13.0	8.5	6.7	6.6
`m050n2`	50	14.1	7.8	6.0	5.8
`m100n1`	100	35.8	13.7	11.4	11.1
`m100n2`	100	39.2	15.5	11.9	11.4
`m200n1`	200	80.5	20.9	18.8	18.4
`m200n2`	200	87.3	26.4	20.3	19.5

Table 1 shows some characteristic results from Raidl (2000) on six hard, misleading 5-MSTP instances due to Knowles and Corne (2000). The table presents results for an EA that encoded spanning trees as Prüfer numbers (PR) and for three other algorithms: the EA of Knowles and Corne (K&C), the weight-coded EA of Raidl and Julstrom (2000) (Wts), and the edge-list EA of Raidl (2000) (LoE). Aside from its coding and operators, the Prüfer-coded algorithm was identical to the edge-list EA. Each algorithm was run 20 independent times on each instance. As in Krishnamoorthy et al. (1999), the Prüfer-coded EA performed worst, and its performance deteriorated most quickly as the problem size increased.

4 The Optimal Communication Spanning Tree Problem

Consider a collection of nodes for which the communication demand between each pair of nodes is given. In the Optimal Communication Spanning Tree Problem (OCSTP), we seek a tree-structured network of minimum total cost that connects all the nodes. A link's flow is the sum of the communication demands between all pairs of nodes communicating directly or indi-

rectly over the link. The cost for each link is not fixed *a priori* but depends on the length and capacity of the link. A link's capacity must satisfy the link's flow and this flow depends on the entire tree structure. Like other constrained spanning tree problems, the OCSTP is NP-hard (Garey and Johnson, 1979, p. 207). Figure 7 shows a communication spanning tree on 15 nodes and emphasizes the path connecting nodes 3 and 14.

Several researchers have encoded candidate communication trees as Prüfer numbers in EAs for this and related problems. For example, Kim and Gen (1999) considered problems in which the nodes were partitioned into users and service centers. Each user connected to exactly one service center, and Prüfer numbers represented spanning trees on the service centers.

Rothlauf et al. (2000) adapted NetKeys (Bean, 1992) to represent communication trees. In this representation, a genotype is a sequence of real-valued priorities associated with the edges that might appear in a communication tree. The decoding algorithm builds the represented tree by including edges in order of these priorities, skipping edges that form cycles. NetKeys support conventional evolutionary operators.

Prüfer numbers and NetKeys were compared in a generational EA on four instances of the OCSTP derived from a real-world problem whose nodes represent locations throughout Germany. Depending on the flow, each link is assigned one of four line types; these have distinct costs per unit length. In two of the instances (Type 1), each link's cost includes a fixed installation cost that does not depend on the link's length. For these instances, optimal solutions resemble stars. In the other two instances (Type 2), each link's cost depends only on its length and capacity. Here, optimal solutions are similar to simple minimum spanning trees.

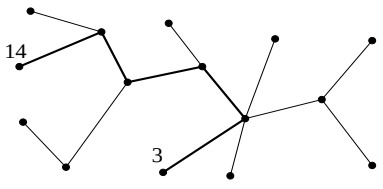

Figure 7: A communication spanning tree on 15 nodes, with the path connecting nodes 3 and 14 emphasized.

Table 2: The average percentage by which the best trees' costs with each coding exceeded the cost of the single best tree found, on each OCSTP instance.

OCSTP	size	Prüfer	NetKey
Type 1	16	3.38	0.02
	32	9.58	0.92
Type 2	16	5.71	0.78
	32	14.31	2.07

The EA in which the two codings were compared was that of Rothlauf et al. (2000). Its population size was 2 000 on all the problem instances. It selected parents in tournaments of size three and generated all offspring with uniform crossover; mutation was not used. The algorithm was run 50 independent times through 100 generations with each coding on each instance.

Table 2 summarizes the results of these trials. It presents the average percentages by which the costs of the best trees in each trial exceeded the costs of the single best tree identified on each instance. The algorithm's performance with Prüfer numbers was inferior to its performance with NetKeys and deteriorated more quickly with increasing problem size.

5 The Rectilinear Steiner Problem

Given a collection of points in the plane, a rectilinear Steiner tree (RStT) is a tree of horizontal and vertical line segments that connects them all. A RStT's length is the sum of its segments' lengths, with overlapping segments included only once. The search for a RStT of minimum length on a set of points is the rectilinear Steiner problem (RStP), and it is NP-hard (Garey and Johnson, 1979, p. 209).

In a rectilinear Steiner tree, the additional points where the segments meet are called Steiner points. In searching for a minimal RStT, we need consider as Steiner points only the unoccupied corners of the rectangles, with sides parallel to the axes, that each pair of

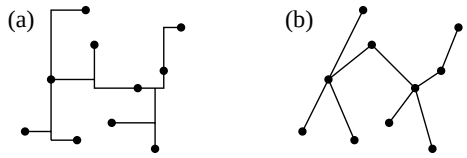

Figure 8: A rectilinear Steiner tree (a) and its underlying spanning tree (b).

points defines (Hanan, 1966). Each pair of points also defines an edge that may appear in a simple spanning tree. Assigning a Steiner point to each of the $n-1$ edges in a spanning tree specifies a RStT. Figure 8 shows a RStT and its underlying spanning tree.

Augmenting a Prüfer number with a string of $n-1$ binary symbols that indicate Steiner points encodes a rectilinear Steiner tree. Prüfer numbers and binary strings support conventional evolutionary operators.

An alternate coding augments Palmer and Kershenbaum's (1994) strings of weights, mentioned in Section 3, with binary strings. The binary symbols specify Steiner points for the edges of the spanning tree the weights represent. This coding also supports conventional operators; mutation changes parental weights and flips Steiner points.

A third coding of rectilinear Steiner trees is as lists of spanning tree edges augmented by Steiner point choices: the entry (17,25,1) represents the pair of rectilinear edges joining points 17 and 25 via, say, their right Steiner point. A list of $n-1$ such entries encodes a RStT. Crossover and mutation operators are based on union-find partitions of the given points.

Julstrom (2001) compared these codings in a generational EA with population size $2n$ and selection from tournaments of size four. With each coding, the algorithm was run 50 times on six RStP instances of from 50 to 150 points. Table 5 summarizes these tests; it lists the average length of the best trees found in each set of trials. The EA's performance is poor with the Prüfer coding, far better with augmented strings of weights, and best with lists of edges.

6 The Fixed-Charge Transportation Problem

Consider distributing a commodity from m sources (factories or warehouses) to each of n destinations (consumers). The amounts of the commodity available at each source and required at each destination are known, and any source can ship to any destination. We seek a pattern of shipments, called a transportation plan, that minimizes the total cost of the deliveries. Figure 9(a) shows an instance of this problem, with $m = 3$ sources and $n = 2$ destinations. Figure 9(b) shows a transportation plan for it.

In the linear transportation problem, the cost of each link between a source and a destination depends linearly on the amount shipped; this problem is solvable in polynomial time (Edmonds and Karp, 1972). In the fixed-charge transportation problem (FCTP), each link also has a fixed cost that is invoked if that link is used; this problem is NP-hard (Guisewite and Pardalos, 1990).

A transportation *tree* consists of the links in a transportation plan and possibly additional links with capacity zero, as Fig. 9(c) illustrates. In such a tree, the amounts assigned to its edges are calculated by traversing the edges, iteratively selecting an edge incident to a leaf, and then assigning the maximum feasible amount to each. Transportation trees represent a subset of the feasible transportation plans, and that sub-

Table 3: Average lengths of the shortest RStTs the EA found with Prüfer numbers, augmented weights, and lists of edges.

RStP	n	Prüfer numbers	Augmented weights	Lists of edges
`eil50`	50	629	461	436
`st70`	70	1287	730	695
`eil75`	75	823	574	554
`rand80`	80	1566	835	780
`rand100`	100	1920	890	824
`rand150`	150	5472	2168	2031

set always contains at least one global optimum, so it is reasonable to restrict search to trees on the sources and destinations. Not all trees represent valid transportation plans, as Fig. 9(d) shows.

Li et al. (1998) described an EA for the FCTP that encodes transportation trees as Prüfer numbers. Gottlieb and Eckert (2000) extended this coding with repair mechanisms to ensure that each Prüfer number decodes to a feasible solution.

Vignaux and Michalewicz (1991) encoded transportation plans as permutations of the mn links between sources and destinations. A decoder scans the links in permutation order and assigns to each the largest amount of the commodity consistent with previous assignments and the problem's constraints. Gottlieb and Paulmann (1998) used this coding in an EA for the FCTP.

A list of its positive edges, each with the amount of the commodity assigned to it, can also represent a transportation tree. For this coding, there are two mutation operators. One introduces a new random edge and adjusts the tree accordingly; the other rearranges the edges incident to a randomly selected node. Crossover inserts half the edges of one parent into the other parent and removes existing edges as necessary.

These codings were compared in a steady-state EA for the FCTP. Its population contained 100 individuals (200 for `n3700` and `n370e`); it selected parents in tournaments of size two; it generated all offspring by crossover, then mutation; it always replaced the current worst genotype;

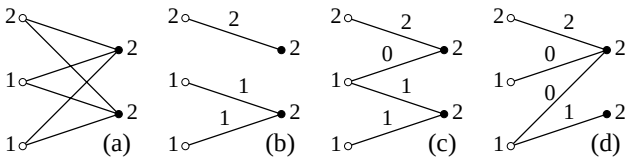

Figure 9: (a) A transportation problem with $m = 3$ sources (circles) and $n = 2$ destinations (dots). (b) A feasible transportation plan for the problem. (c) A transportation tree corresponding to the plan of (b). (d) A transportation tree whose plan violates the problem's constraints.

Table 4: The average percentage by which the twelve best plans' costs, with each coding, exceeded the costs of the best plans found, on six FCTP instances.

FCTP	Prüfer numbers	Permutations	Lists of edges
`bal8x12`	0.00	0.00	0.00
`ran4x64`	21.34	0.54	0.21
`ran14x18`	14.07	4.19	1.60
`ran16x16`	10.58	2.61	0.63
`n3700(50x100)`	65.96	13.79	0.83
`n370e(50x100)`	70.44	18.67	0.97

and it did not allow phenotypic duplicates. The EA was run twelve independent times with each coding on five FCTP instances ranging in size from $m = 8$ and $n = 12$ to $m = 50$ and $n = 100$. Table 4 summarizes these trials, some of which were reported in (Gottlieb and Eckert, 2000). For each instance and each coding, the table presents the percentage by which the average cost of the trials' best plans exceeded the cost of the single best plan found. Again, the EA performed worst with the Prüfer coding. Its performance was better with permutations and best when lists of edges represented transportation plans.

7 Conclusion

Prüfer numbers encode spanning trees on n nodes as strings of $n - 2$ node labels. Several researchers have claimed that Prüfer numbers allow effective evolutionary search of spaces of spanning trees. We have demonstrated that this is not so.

Though Prüfer numbers support conventional evolutionary operators like k-point crossover and position-by-position mutation, only a negligible fraction of the genotype space of Prüfer numbers provides high locality and heritability under those operators. In general, mutated Prüfer numbers do not represent trees similar to those of their parents, nor do the offspring of crossover encode trees that consist mainly of

substructures of the parental trees.

Empirical investigations on four NP-hard problems involving spanning trees confirm that the Prüfer coding is a poor choice in evolutionary algorithms. On every problem, an EA returned the worst results when Prüfer numbers encoded candidate trees, and much better results with other codings. Also, the performance of the Prüfer-coded versions deteriorated the most quickly as the problem instances got larger.

Why have some researchers reported good results with Prüfer numbers? In most cases, the problem instances were very small, so that any representation was adequate. In others, good solutions resembled stars, near which evolutionary search can be effective in spaces of Prüfer numbers.

Another disadvantage of Prüfer numbers is that they correspond to unconstrained spanning trees in complete graphs. When the underlying graph is not complete or the problem is otherwise constrained, an offspring Prüfer number will not in general represent a valid solution. Offspring can be repaired, but such operations further degrade the representation's locality and heritability. More generally, operators that could enforce locality and heritability on Prüfer numbers must decode them and operate on the decoded representations.

For three of our four problems, one of the codings of candidate solutions was lists of spanning tree edges, sometimes augmented with information such as Steiner points or commodity amounts. The EAs returned their best results with these codings. With appropriate operators, lists of edges show high locality and heritability, and they can be manipulated to satisfy constraints. This suggests that lists of edges may be a generally effective representation of spanning trees for evolutionary search.

References

F. N. Abuali, D. A. Schoenefeld, and R. L. Wainwright. Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, editors, *Proceedings of the 1994 ACM Symposium on Applied Computing*, pages 242–246. ACM Press, 1994.

P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala, and W. Porto, editors. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, 1999. IEEE Press.

J. C. Bean. Genetics and random keys for sequencing and optimization. Technical Report 92-43, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1992.

A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889.

J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

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

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

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 J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, page 495. Morgan Kaufmann, 1998.

M. Gen and Y. Li. Spanning tree-based genetic algorithms for the bicriteria fixed charge transportation problem. In Angeline et al. (1999), pages 2265–2271.

J. Gottlieb and C. Eckert. A comparison of two representations for the fixed charge transportation problem. In Schoenauer et al. (2000), pages 345–354.

J. Gottlieb and L. Paulmann. Genetic algorithms for the fixed charge transportation problem. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 330–335. IEEE Press, 1998.

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

M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14(2): 255–265, 1966.

B. A. Julstrom. Encoding rectilinear Steiner trees as lists of edges. In *Proceedings of the 2001 ACM Symposium on Applied Computing*, 2001. Las Vegas, NV, March 11–14, 2001.

J. R. Kim and M. Gen. Genetic algorithm for solving bicriteria network topology design problem. In Angeline et al. (1999), pages 2272–2279.

J. Knowles and D. Corne. A new evolutionary approach to the degree constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2):125–134, 2000.

M. Krishnamoorthy, A. T. Ernst, and Y. M. Sharaiha. Comparison of algorithms for the degree constrained minimum spanning tree. Technical report, CSIRO Mathematical and Information Sciences, Clayton, Australia, 1999.

Y. Li, M. Gen, and K. Ida. Fixed charge transportation problem by spanning tree-based genetic algorithm. *Beijing Mathematics*, 4(2):239–249, 1998.

C. C. Palmer and A. Kershenbaum. Representing trees in genetic algorithms. In D. Schaffer, H.-P. Schwefel, and D. B. Fogel, editors, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 379–384. IEEE Press, 1994.

H. Prüfer. Neuer Beweis eines Satzes ueber Permutationen. *Archiv für Mathematik und Physik*, 27:742–744, 1918.

G. R. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In C. Fonseca, J.-H. Kim, and A. Smith, editors, *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pages 104–111. IEEE Press, 2000.

G. R. Raidl and B. A. Julstrom. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 440–445. ACM Press, 2000.

F. Rothlauf and D. E. Goldberg. Pruefernumbers and genetic algorithms: A lesson on how the low locality of an encoding can harm the performance of GAs. In Schoenauer et al. (2000), pages 395–404.

F. Rothlauf, D. E. Goldberg, and A. Heinzl. Network random keys – a tree network representation scheme for genetic and evolutionary algorithms. Technical Report No. 8/2000, University of Bayreuth, Germany, 2000.

M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature – PPSN VI*, number 1917 in Lecture Notes in Computer Science, 2000. Springer.

G. A. Vignaux and Z. Michalewicz. A genetic algorithm for the linear transportation problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(2):445–452, 1991.

G. Zhou and M. Gen. Approach to degree-constrained minimum spanning tree problem using genetic algorithm. *Engineering Design & Automation*, 3(2):157–165, 1997.