The Blob Code is Competitive with Edge-Sets in Genetic Algorithms for the Minimum Routing Cost Spanning Tree Problem

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

ABSTRACT

Among the many codings of spanning trees for evolutionary search are those based on bijections between Prüfer strings—strings of $n-2$ vertex labels—and spanning trees on the labeled vertices. One of these bijections, called the Blob Code, showed promise as an evolutionary coding, but EAs that use it to represent spanning trees have not performed well. Here, a genetic algorithm that represents spanning trees via the Blob Code is faster than, and returns results competitive with those of, a GA that encodes spanning trees as edge-sets on Euclidean instances of the minimum routing cost spanning tree problem. On instances whose edge weights have been chosen at random, the Blob-coded GA maintains its time advantage, but its results are inferior to those of the edge-set-coded GA, and both GAs are hard pressed to keep up with a simple stochastic hill-climber on all the test instances.

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Numerical Analysis—*Optimization*; G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

General Terms

Algorithms

Keywords

Spanning trees, codings, Blob Code, edge-sets, routing cost

1. INTRODUCTION

Given a connected, weighted, undirected graph G, the familiar algorithms of Prim and Kruskal identify minimum spanning trees on G in times that are polynomial in the number n of G's vertices. Searching the space of G's spanning trees is often NP-hard, however, when the trees are constrained, as by bounding their degrees or numbers of leaves, or when the objective function is other than a tree's weight, as its diameter or number of leaves. In cases like these, we turn to heuristics, including evolutionary algorithms.

Researchers have described a variety of codings of spanning trees for evolutionary search [20, pp.119–198]. Among them, edge-sets [19] represent spanning trees directly as lists of their edges. Other codings are based on Cayley's formula, which tells us that the number of spanning trees in a complete graph on n vertices is n^{n-2} [4] [5, pp.103–4]. Prüfer [18] described inverse one-to-one mappings between the strings of length $n-2$ over an alphabet of n vertex labels and the spanning trees on n vertices. We call the strings Prüfer strings; when they represent spanning trees via Prüfer's mapping, they are Prüfer numbers. Positional genetic operators like k-point crossover and position-by-position mutation can be applied to Prüfer numbers, but this combination does not support effective evolutionary search [8] [16] [21].

Among the many other bijections between Prüfer strings and spanning trees, the Blob Code [17] appears to have good qualities for evolutionary search using positional operators. In particular, it exhibits much greater locality and heritability under positional crossover and mutation than do strings decoded with the Prüfer mapping [12]. However, EAs using the Blob Code have not performed as well as EAs using other representations, in particular edge-sets, on problems that search spaces of spanning trees [19].

The routing cost of a tree is the sum of the weights of all the paths in the tree that connect distinct pairs of vertices. Given a graph G, the minimum routing cost spanning tree problem seeks a spanning tree on G of minimum routing cost. Two genetic algorithms for this problem encode spanning trees as edge-sets and as Prüfer strings decoded via the Blob Code. On Euclidean problem instances, the Blob-coded GA is faster than the edge-set-coded GA, and its results are competitive with those of the latter algorithm. On instances whose edge weights have been chosen at random, the Blob-coded GA maintains its time advantage, but its results are not as good as those of the edge-set-coded GA. On all the test instances, both GAs are less effective than a simple stochastic hill-climber.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

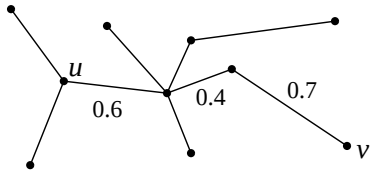


Figure 1: A spanning tree on ten vertices. The routing cost of the path connecting u and v is $0.6 + 0.4 + 0.7 = 1.7$.

The following sections describe the minimum routing cost spanning tree problem; edge-sets; the Blob Code; genetic algorithms that implement the two codings; the hill-climber; and comparisons of the three algorithms on a variety of problem instances, both Euclidean and with randomly-chosen weights.

2. THE PROBLEM

Let T be a spanning tree on a connected, weighted, undirected graph $G = (V, E)$. Any two vertices $u, v \in V$ are connected by a unique path in T; the sum of the weights of the path's edges is the path's routing cost $c_T(u, v)$. Figure 1 shows a spanning tree on ten vertices. The routing cost of the path connecting vertices u and v is $0.6 + 0.4 + 0.7 = 1.7$. The routing cost or total path length $C(T)$ of T is the sum of the routing costs of the paths in T between all the distinct pairs of vertices:

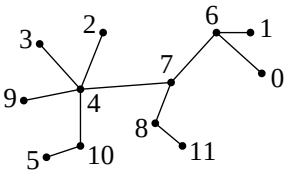
$$C(T) = \sum_{u,v \in V} c_T(u, v).$$

The minimum routing cost spanning tree (MRCST) problem seeks a spanning tree on G with minimum routing cost $C(T)$.

Hu [10] first described the MRCST problem as a case of the optimum communications spanning tree (OCST) problem, which specifies communications requirements $r(u, v)$ between each pair of vertices $u, v \in V$ in addition to the edge weights of the graph. Both problems are NP-hard [11]. Fischetti et al. [6] explored exact algorithms and Wu et al. [24] described a polynomial-time approximation scheme for the MRCST problem.

Several researchers have developed evolutionary algorithms for the OCST problem, including Palmer and Kershenbaum [16], Berry et al. [3], Li and Bouchebaba [15], Rothlauf et al. [22], Gaube and Rothlauf [7], and Li [14]. Julstrom [13] described a genetic algorithm, reprised in Sections 3 and 5 below, for the MRCST problem.

To determine the routing cost of a tree, it is not necessary to make explicit every path within it. The contribution of each edge to the routing cost, called the edge's routing load, is the edge's weight times the number of paths on which it lies, and that number of paths is the product of the numbers of vertices in the two components created by removing the edge from the tree. The routing cost is the sum of the routing loads over the tree's edges, and a traversal of the tree, starting at any vertex, can identify all the component sizes and accumulate the edges' routing loads. If adjacency lists [1, pp.232–3] represent the tree, traversal requires time that is linear in the number of edges, which is $n - 1$, thus the time to find the routing cost is $O(n)$.



Edge-set: {(1,6), (2,4), (3,4), (4,7), (5,10), (6,0), (7,6), (8,7), (9,4), (10,4), (11,8)}

Blob Code: (6 4 4 7 10 6 7 4 4 8)

Figure 2: A spanning tree on twelve vertices and two encodings of it: an edge-set and a Prüfer string that represents the tree via the Blob Code.

3. THE EDGE-SET CODING

Edge-sets represent spanning trees directly as lists of their edges [19]. Figure 2 shows a spanning tree on twelve vertices and an edge-set that represents it. To find the routing cost of the tree an edge-set represents, scan its edges and build corresponding unordered adjacency lists. Then traverse the tree, using the adjacency lists, to sum the edges' routing costs as Section 2 described. The time to build the lists is linear in n, so the time for evaluation remains $O(n)$.

A Kruskal-based crossover operator builds one offspring edge-set from two parents. It copies the edges common to the parents into the offspring, then chooses from the remaining parental edges, discarding those that create cycles, until the offspring represents a tree on all the vertices. An efficient implementation of this operator begins by sorting the parental edge-sets to facilitate finding their common edges and uses a union-find partition [1, pp.180–9] to track the connected components as it builds the offspring. The sorting step determines the operator's time complexity, which is $O(n \log n)$.

Mutation scans a parent edge-set and, with a small probability p_{mu}, replaces each edge. When an edge is replaced, a union-find partition identifies the two components. Tarjan [23] and, more recently Harfst and Reingold [9], have shown that the time of a sequence of operations, appropriately implemented, on a union-find partition grows as the product of the number of operations and an inverse of Ackermann's function. Identifying the two components created when an edge is removed requires $n - 2$ such operations, so the time mutation requires is just more than linear in n.

4. THE BLOB CODE

The Blob Code [12] [17] is one of many one-to-one mappings between Prüfer strings—strings of length $n - 2$ over an alphabet of n symbols—and spanning trees on n vertices. In the Blob decoding algorithm, which makes explicit the directed tree a string represents, the blob itself is a set of vertices. Assuming that the vertices are numbered from 0 to $n - 1$, the blob initially contains all the vertices except vertex 0. Each of the algorithm's iterations removes a vertex from the blob and records a directed edge in the spanning tree. Ignoring the edges' directions yields the undirected spanning tree that the string represents.

The decoding algorithm uses two functions: **successor**(v) returns the vertex w if $(v \to w)$ is among the edges currently in the spanning tree, and **path**(v) returns TRUE if the directed path from v towards vertex 0 intersects the blob, FALSE if it does not. The following sketch summarizes the Blob algorithm; in it, $a_1 a_2 \ldots a_{n-2}$ is a Prüfer string and T is the directed spanning tree.

```
blob ← {1, 2, ..., n − 1};
T ← {(blob → 0)};
for i from 1 to n − 2 do
    blob ← blob −{i};
    if path(a_i)
        T ← T ∪ {(i → a_i)};
    else
        T ← T ∪ {(i → successor(blob))};
        T ← T − {(blob → successor(blob))};
        T ← T ∪ {(blob → a_i)};
blob ← (n − 1) in all edges;
return T;
```

Figure 2 includes a Prüfer string—(6 4 4 7 10 6 7 4 4 8)—that represents the spanning tree shown there via the Blob Code. Consider the algorithm's actions as it identifies the tree's edges.

Initially, the blob contains vertices 1 through 11, and the tree consists of the single edge (blob $\rightarrow 0$). The algorithm's first iteration removes vertex 1 from the blob. $a_1 = 6$, and the blob contains vertex 6, so that **path(6)** is TRUE and the edge $(1 \rightarrow 6)$ is added to the tree.

The second iteration removes vertex 2 from the blob. $a_2 = 4$, **path(4)** is TRUE, and the edge $(2 \rightarrow 4)$ is added to the tree. This process continues through seven more iterations, each of which increases the number of the tree's edges by one, and then the blob itself is replaced by vertex 11.

The Blob decoding algorithm's worst-case time is $O(n^2)$, but on average it appears to be much faster. An efficient implementation of it represents the tree T by an array `t[·]` in which `t[u] = v` represents the directed edge `(u → v)` and the value $n - 1$ represents the blob. Build adjacency lists that represent the undirected tree by scanning this array, then identify the tree's routing cost by traversing the tree.

Every Prüfer string represents a unique tree via the Blob Code, so traditional positional operators are appropriate. In particular, the crossover operator is two-point crossover, and the mutation operator is position-by-position mutation, in which, with a small probability p_{mu}, each symbol is replaced with a random one from $\{0, 1, \ldots, n - 1\}$. Both operators' times are $O(n)$.

5. TWO GENETIC ALGORITHMS

Two genetic algorithms for the MRCST problem encode spanning trees as edge-sets (ES-GA) and as Prüfer strings decoded with the Blob Code (B-GA), respectively. The GAs are generational, and they initialize their populations with random chromosomes of the appropriate types: ES-GA decodes random Prüfer strings to spanning trees, and the initial population of B-GA contains random Prüfer strings.

The GAs select chromosomes to reproduce in k-tournaments, and they apply crossover and mutation independently; one operator or the other builds each offspring. They implement 1-elitism—the best chromosome in the current generation survives into the next—and they run through fixed numbers of generations.

On a MRCST problem instance with n vertices, the GAs' populations contain $40\sqrt{n}$ chromosomes and they run through $250\sqrt{n} - 1$ generations, so that they evaluate $10000n$ chromosomes, including those in their initial populations. Their selection tournaments involve $k = 2$ contestants, and a tournament's winner is always selected to reproduce. In the edge-set-coded GA, the probability that crossover generates an offspring is 80%, and the probability of mutation is therefore 20%. In the Blob-Coded GA, these probabilities are 60% and 40%, respectively. In both GAs, mutation replaces an edge or a symbol with probability $2/n$.

Table 1: Parameters of the two genetic algorithms that depend on the problem size: population size $= 40\sqrt{n}$, generations $= 250\sqrt{n} - 1$, and $p_{\mathbf{mu}} = 2/n$.

n	PopSize	Gens	p_{mu}
50	283	1767	0.040
100	400	2499	0.020
250	632	3952	0.008
300	693	4329	0.007

6. A STOCHASTIC HILL-CLIMBER

Hill-climbing is a search heuristic that begins with a random solution to the target problem instance and then incrementally improves it. A hill-climber repeatedly generates one or several neighboring solutions and moves to the best of the current solution and the neighbors. Hill-climbers are distinguished by how they generate and examine neighbors. A simple stochastic hill-climber (SHC) repeatedly generates one random neighbor and moves to it if it is better than the current solution:

```
S ← a random solution;
while ( not done )
    S_1 ← a random neighbor of S;
    if ( S_1 is a better solution than S )
        S ← S_1;
report S;
```

In a SHC for the MRCST problem, solutions are spanning trees represented by edge-sets, neighboring solutions differ in one edge, and a solution S_1 is better than another S if it has smaller routing cost. The hill-climber begins with a random spanning tree and runs through $10000n$ iterations, so that it performs as many evaluations as do the two genetic algorithms. If the hill-climber reaches a local minimum, it stays there, as no neighbor can improve on the current solution and the algorithm does not start over.

7. COMPARISONS

The edge-set-coded GA, the Blob-coded GA, and the stochastic hill-climber were compared on 35 instances of the MRCST problem. Twenty-one of these instances are Euclidean, seven each of $n = 50$, 100, and 250 vertices. They are listed in Beasley's OR-Library[1] [2] as instances of the Euclidean Steiner problem. Each consists of n points in the unit square; we treat the points as vertices of complete graphs whose edge weights are the floating-point distances between the points.

The library lists fifteen instances of each size (and others); we use the first seven of each group. The remaining fourteen instances, seven each of $n = 100$ and 300 vertices, are complete graphs with edge weights chosen at random on the interval $[0.01, 0.99]$. Table 1 shows the values of the GAs' parameters that depend on the problem size.

[1]http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Table 2: Results of the trials of the two genetic algorithms and the hill-climber on the twenty-one Euclidean (EUC) and fourteen random-weight (RAN) MRCST problem instances. For each set of 30 trials, the table lists the routing cost of the best tree found and the mean and standard deviation of the trials' 30 routing costs. The smaller best and mean values for ES-GA and B-GA are underlined. The smallest best and mean values for all three algorithms are bold.

	Type	Edge-Set-Coded GA			Blob-Coded GA			Stochastic Hill-Climber		
n	Inst	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD
EUC	1	**984.8**	**998.1**	17.1	987.6	1008.1	15.6	985.1	1003.5	23.9
50	2	**901.4**	**907.8**	6.7	902.3	912.2	12.5	902.0	912.6	13.3
	3	**888.3**	913.1	21.8	889.9	935.5	31.7	**888.3**	**908.7**	18.4
	4	778.2	793.8	18.8	777.1	797.3	17.4	**776.9**	**792.5**	18.5
	5	**847.9**	**858.3**	15.1	848.1	865.5	20.4	**847.9**	860.8	30.2
	6	818.4	**825.5**	6.1	819.3	843.3	34.4	**818.1**	829.0	22.7
	7	**865.6**	**881.5**	14.8	865.9	889.1	16.6	865.9	886.3	16.7
EUC	1	3538.4	3585.5	56.9	3525.5	3592.8	65.0	**3513.2**	**3553.5**	47.1
100	2	3315.2	3400.2	48.5	3342.7	3407.3	46.3	**3310.6**	**3359.7**	52.8
	3	3576.0	3641.8	56.7	3573.9	3665.6	76.1	**3566.9**	**3610.7**	38.4
	4	3464.5	3541.3	57.0	3468.1	3551.2	57.6	**3458.4**	**3504.0**	39.9
	5	3652.8	3764.3	83.7	3641.9	3737.8	63.0	**3639.9**	**3707.6**	62.9
	6	3455.0	3487.1	19.9	3443.3	3501.3	37.5	**3436.8**	**3461.3**	18.8
	7	3730.1	3783.5	61.2	3733.8	3819.5	76.2	**3711.6**	**3741.8**	29.0
EUC	1	22545.7	23225.3	556.1	22543.0	23013.6	306.9	**22177.2**	**22568.3**	359.6
250	2	23286.6	24310.7	590.7	23149.1	23834.0	563.1	**22961.9**	**23433.1**	464.9
	3	22394.7	23094.2	519.7	22237.0	22821.4	420.6	**22055.7**	**22473.7**	310.7
	4	23725.8	24824.2	743.9	23837.3	24647.0	607.1	**23598.3**	**23903.2**	293.8
	5	22604.0	23352.8	455.0	22739.4	23264.8	306.3	**22458.1**	**22880.5**	256.3
	6	22662.0	23326.6	448.8	22507.4	23102.6	407.4	**22334.3**	**22559.3**	171.1
	7	23390.0	23853.7	538.7	23238.2	23781.2	328.5	**23039.9**	**23226.9**	190.5
RAN	1	598.5	628.0	33.8	625.8	724.4	86.9	**597.9**	**609.8**	18.4
100	2	586.0	640.7	37.1	623.0	730.5	53.2	**586.0**	**601.6**	26.6
	3	607.7	668.9	49.5	678.2	756.5	48.1	**607.0**	**637.3**	55.8
	4	607.3	636.8	20.2	628.3	715.9	66.3	**598.4**	**610.9**	16.9
	5	628.4	668.4	34.9	641.7	757.1	74.8	**624.4**	**643.1**	19.1
	6	615.6	655.1	30.2	639.1	732.8	54.9	**615.5**	**615.5**	0.0
	7	514.9	539.0	24.7	531.3	646.8	66.4	**514.7**	**514.8**	0.1
RAN	1	4926.3	5444.7	316.9	5361.7	6048.2	618.6	**4131.1**	**4259.8**	119.4
300	2	4957.5	5488.0	314.2	5236.3	6017.8	413.5	**4040.7**	**4237.6**	180.7
	3	5013.1	5452.4	300.8	5093.5	5917.3	396.3	**4134.8**	**4259.5**	87.3
	4	5012.4	5773.5	378.4	5320.0	6135.4	472.9	**4229.3**	**4397.1**	163.8
	5	4622.4	5433.1	362.7	5118.2	6010.7	423.7	**3951.9**	**4132.3**	177.4
	6	5259.9	5641.2	222.3	5408.8	6083.7	360.4	**4314.4**	**4517.6**	80.2
	7	4868.3	5468.5	342.1	5342.6	599.02	442.9	**4093.9**	**4268.5**	144.9

All tests were performed on a Pentium 4 processor with 256 megabytes of memory, running at 2.53 GHz under Red Hat Linux 9.0.

The three algorithms were each run 30 independent times on each instance. Tables 2 and 3 summarize the results of these trials. For each algorithm on each instance, Table 2 lists the smallest routing cost found and the mean and standard deviation of the 30 routing costs. In the comparison of the edge-set-coded GA ES-GA and the Blob-coded GA B-GA, the better smallest and mean values are underlined. In the comparison of all three algorithms, the best smallest and mean values are bold. For each algorithm on each kind and size of instance, Table 3 lists typical wall-clock times in seconds that they required.

Consider first the edge-set-coded GA and the Blob-coded GA on the Euclidean instances. On the smallest instances, ES-GA almost always identifies trees of smaller routing cost, both best and mean. On the instances with 100 vertices, the two algorithms' performances are similar. B-GA finds the better best tree more often, the better mean routing cost less often, than does ES-GA. On the instances with n=250 vertices, B-GA is almost always better, finding the better best tree on five instances and the better mean routing cost on every instance.

On the random-weight instances, the Blob-coded GA's performance is consistently worse than that of the edge-set-coded GA. On all these instances, ES-GA returns better best and mean values than does B-GA, and the differences are large. The Blob-coded GA's performance on these instances is poor.

Table 3 shows that B-GA is consistently faster than ES-GA. On all the test instances, its time is about 40% of that

Table 3: Typical wall-clock times in seconds that each algorithm required on each size of each kind of MRCST problem instance.

Type	n	ES-GA	B-GA	SHC
EUC	50	21.2	9.2	9.6
EUC	100	91.3	36.6	40.0
EUC	250	618.2	231.1	258.4
RAN	100	92.6	37.4	40.1
RAN	300	905.5	351.7	377.8

of ES-GA, and this advantage increases slightly as the instances get larger.

These are the project's main results: On Euclidean instances of the MRCST problem, a genetic algorithm that encodes trees as Prüfer strings decoded with the Blob Code returns results competitive with those of a GA that encodes candidate trees as edge-sets. The Blob-coded GA's advantage over the edge-set-coded GA becomes more consistent as the instances grow larger. On all the instances, the Blob-coded GA requires only about 40% as much time as does the edge-set-coded GA.

When all three algorithms are compared, the hill-climber consistently wins. On the smallest Euclidean instances, ES-GA does best, then SHC, then B-GA, but the differences are small. On the remaining instances, SHC is always best, often by a large margin, though B-GA is always slightly faster than SHC.

The overall effectiveness of the hill-climber is probably due, as suggested in [13], to the fact that it conserves every improvement it finds in its one tree. In the GAs, good partial solutions must propagate through the populations to encounter each other. Further, the GAs evaluate and record offspring that may not encode improved trees and that may not participate in reproduction, in which case the information they contain is lost.

The fact that the hill-climber climbs effectively suggests that the landscape it searches, though not without local minima, is relatively smooth. This in turn suggests that the genetic algorithms, particularly ES-GA, whose landscape is similar to SHC's, would eventually catch up, and their populations of solutions provide a chance to escape local minima.

Figure 3 supports this observation. It shows the three algorithms' average performances on the first Euclidean instance with 250 vertices. The hill-climber identifies good solutions quickly, but the two GAs are still identifying better solutions when they terminate. The number of evaluations allowed the algorithms appears ample for SHC but, particularly on the larger instances, too small for the GAs to do all they can.

8. CONCLUSION

The strings of length $n-2$ over an alphabet of n symbols can be placed in one-to-one correspondence with the spanning trees on n vertices via a large number of mappings. One of these, the Blob Code, has shown promise as a representation of spanning trees for evolutionary search, but evolutionary algorithms using it have not performed as well as EAs that encode spanning trees in other ways, such as edge-sets. Here, a genetic algorithm that represents spanning trees as Prüfer strings decoded with the Blob Code competes effectively with another that uses edge-sets on Euclidean instances of the minimum routing cost spanning tree problem, though it performs poorly on instances with randomly-chosen edge weights, and both algorithms cannot keep up with a simple stochastic hill-climber. On all the test instances, the Blob-coded GA requires less than half the time of the edge-set-coded GA.

This inquiry raises additional questions and possibilities. The hill-climber's success relative to both genetic algorithms suggests that an EA augmented with local search—or parallel hill-climbers augmented with crossover—might be effective on the minimum routing cost spanning tree problem. Operators that incorporate domain knowledge, say by favoring edges of lower weight, might improve the performance of the hill-climber and the edge-set-coded GA.

It is not clear why the Blob-coded GA is competitive on the Euclidean instances of the MRCST problem but not on the random instances. Apparently the operators the GA applies to Prüfer strings, the Blob Code mapping, and the structure of the problem instances interact beneficially when the instances are Euclidean and not when their edge weights are chosen randomly, but the nature of that interaction remains a mystery. More generally, there are likely other spanning tree problems on which Blob-coded EAs might be effective, as well as other mappings from Prüfer strings to spanning trees that better support evolutionary search of spaces of spanning trees.

9. REFERENCES

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

[2] J. E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[3] L. T. M. Berry, B. A. Murtagh, S. J. Sugden, and G. B. McMahon. Application of a genetic-based algorithm for optimal design of tree-structured communication networks. In *Proceedings of the Regional International Teletraffic Conference*, pages 361–369, Pretoria, South Africa, 1995.

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

[5] Shimon Even. *Algorithmic Combinatorics*. The Macmillan Company, New York, 1973.

[6] Matteo Fischetti, Giuseppe Lancia, and Paolo Serafini. Exact algorithms for minimum routing cost trees. *Networks*, 39(1):161–173, 2002.

[7] Thomas Gaube and Franz Rothlauf. The link and node biased encoding revisited: Bias and adjustment of parameters. Technical Report 2001011, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2001.

[8] Jens Gottlieb, Bryant A. Julstrom, Günther R. Raidl, and Franz Rothlauf. Prüfer numbers: A poor representation of spanning trees for evolutionary search. In L. Spector et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO–2001*, pages 343–350, San Francisco, CA, 2001. Morgan Kaufmann. July 7–11, San Francisco, CA.

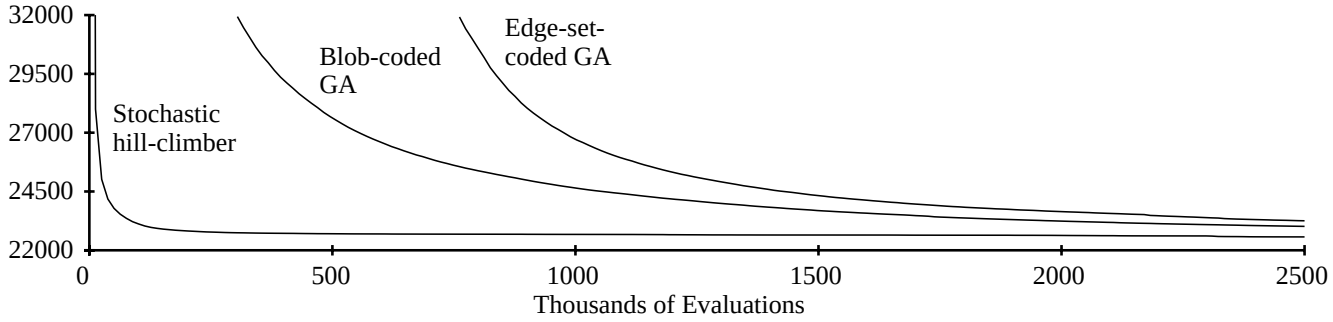


Figure 3: Average performances of the three algorithms on the first Euclidean MRCST instance with $n = 250$ points.

[9] Gregory C. Harfst and Edward M. Reingold. A potential-based amortized analysis of the union-find data structure. *SIGACT News*, 31(3):86–95, 2000.

[10] T. C. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974.

[11] D. S. Johnson, J. K. Lenstra, and A. H. G. Ronooy Kan. The complexity of the network design problem. *Networks*, 8:279–285, 1978.

[12] Bryant A. Julstrom. The Blob Code: A better string coding of spanning trees for evolutionary search. In Annie S. Wu, editor, *2001 Genetic and Evolutionary Computation Conference Workshop Program*, pages 256–261, San Francisco, CA, 2001. July 7.

[13] Bryant A. Julstrom. A genetic algorithm and two hill-climbers for the minimum routing cost spanning tree problem. In H. R. Arabnia and Youngsong Mun, editors, *Proceedings of the International Conference on Artificial Intelligence*, volume III, pages 934–940. CSREA Press, 2002.

[14] Yu Li. An effective implementation of a direct spanning tree representation in GAs. In Egbert J. W. Boers et al., editors, *Proceedings of EvoWorkshops 2001*, pages 11–19, Berlin, 2001. Springer-Verlag.

[15] Yu Li and Youcef Bouchebaba. A new genetic algorithm for the optimal communications spanning tree problem. In Cyril Fonlupt et al., editors, *Artificial Evolution: 4th European Conference*, number 1829 in Lecture Notes in Computer Science, pages 162–173, Berlin, 1999. Springer-Verlag.

[16] Charles C. Palmer and Aaron Kershenbaum. Representing trees in genetic algorithms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 379–384, June 1994.

[17] Sally Picciotto. *How to encode a tree*. PhD thesis, University of California, San Diego, 1999.

[18] H. Prüfer. Neuer beweis eines satzes über permutationen. *Archives of Mathematical Physics*, 27:142–144, 1918.

[19] Günther R. Raidl and Bryant A. Julstrom. Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3):225–239, 2003.

[20] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms*, volume 104 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, Heidelberg, 2002.

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

[22] Franz Rothlauf, David E. Goldberg, and Armin Heinzl. Network random keys – a tree network representation scheme for genetic and evolutionary algorithms. Technical Report No. 8/2000, University of Bayreuth, Germany, 2000.

[23] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[24] Bang Ye Wu, Guiseppe Lancia, Vineet Bafna, Kun-Mao Chao, R. Ravi, and Chuan Yi Tang. A polynomial time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing*, 29(3):761–768, 1999.