Three Evolutionary Codings of Rectilinear Steiner Arborescences

Bryant A. Julstrom[1] and Athos Antoniades[2]

[1] St. Cloud State University, St. Cloud, MN 56301 USA
julstrom@eeyore.stcloudstate.edu
[2] University of Nicosia, Nicosia, Cyprus
athos@athosonline.com

Abstract. A rectilinear Steiner arborescence connects points in the Euclidean plane's first quadrant and the origin with directed rectilinear edges from the origin up and to the right. The search for arborescences of minimum total length is NP-hard and finds applications in VLSI design. A greedy heuristic for this problem often returns near-optimum arborescences. Three genetic algorithms encode candidate arborescences as permutations of the points, as perturbations of the points' locations, and as perturbations of the points' rectilinear distances from the origin. In comparisons on twenty collections of 50 to 250 points in the first quadrant, the permutation-coded GA returns arborescences that are longer than those of the greedy heuristic. The two perturbation-coded GAs return nearly identical results, their arborescences are consistently shorter than those of the heuristic, and they preserve their advantage as the number of points grows. These results support the usefulness of perturbation codings in evolutionary algorithms for geometric problems like the search for short rectilinear Steiner arborescences.

1 Introduction

An arborescence is a connected, directed graph that contains no cycles and in which each vertex has at most one entering edge. Because an arborescence is acyclic, one of its vertices has no entering edge. This vertex is the root, and there is a directed path in the arborescence from it to every other vertex.

In a rectilinear Steiner arborescence (RSA), the vertices are points in the first quadrant of the Euclidean plane and the origin. The origin is an RSA's root, and vertical and horizontal edges lead from it to all the points. On a path from the origin, every edge leads either up or to the right; no edge ever turns back toward either coordinate axis. Figure 1 shows a rectilinear Steiner arborescence on twenty points in the first quadrant and the origin. Note that in an RSA, the path connecting the origin to any point p has minimum length: the rectilinear distance from the origin to p.

A minimum rectilinear Steiner arborescence (MRSA) is a RSA of minimum total length. The minimum rectilinear Steiner arborescence problem, which seeks a MRSA, has applications in VLSI design, where signals from a source must

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 1282–1291, 2004.

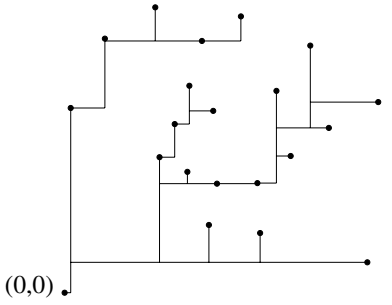

Fig. 1. A rectilinear Steiner arborescence on 20 points in the first quadrant and the origin

be delivered to terminals via rectilinear paths as quickly and economically as possible [1]. Shi and Su [2] established that the MRSA problem is NP-hard.

Rao *et al.* [3] presented an elegant greedy heuristic, described in Section 2 below, for the problem. Córdova and Lee [4] extended this heuristic to points in all four quadrants and observed that it often returns solutions that are very nearly optimal. Leung and Cong [5] based a branch-and-bound exact algorithm on the heuristic, and described another dynamic programming solution. Ramnath [6] described a new heuristic for the problem that he suggested extends more effectively to three dimensions.

Given a collection of points in the plane, a rectilinear Steiner tree (RStT) is a tree made up of vertical and horizontal line segments that connects them all; rectilinear Steiner arborescences are a special case of RStTs. Several evolutionary algorithms that seek short RStTs have been described [7] [8], but their codings of RStTs are based on representations of simple spanning trees and do not adapt well to RSAs. We describe and compare three codings designed to represent RSAs. In the first coding, permutations of points dictate the order in which they are greedily attached to growing arborescences.

The second and third codings are closely related and based on the heuristic of Rao *et al.* Both of them encode RSAs as strings of real values that perturb the points' locations or distances. Such perturbations have been used in EAs for other geometric problems, including the traveling salesman problem (Valenzuela and Williams, 1997; Cohoon et al., 1998).

In the second coding, floating-point perturbations jiggle point locations, and the heuristic of Rao *et al.* builds an arborescence based on the perturbed locations. In the third coding, perturbations are applied directly to the rectilinear distances between points and the origin, and the heuristic of Rao *et al.* builds an arborescence by examining the perturbed distances.

Genetic algorithms using the codings are compared to each other and to the heuristic of Rao *et al.* on twenty instances of the minimum rectilinear Steiner arborescence problem of 50 to 250 points. The GA that uses the permutation coding cannot compete with the greedy heuristic, but the perturbation-coded GAs, which take advantage of the heuristic, almost always improve on its al-

ready short arborescences. The results of the two perturbation-coded GAs are essentially identical, though the second perturbation coding uses less space.

The following sections of this paper describe the heuristic of Rao *et al.*; the permutation coding of RSAs; the two perturbation codings of RSAs; the genetic algorithms; and comparisons of the heuristic and the GAs.

2 A Greedy Heuristic

The heuristic of Rao *et al.* [3] greedily develops a short RSA on a collection of points in the first quadrant and the origin. This presentation follows theirs, and begins with some definitions.

For a point $p = (p_x, p_y)$ in the first quadrant, let $\|p\| = p_x + p_y$; that is, $\|p\|$ is the rectilinear distance from p to the origin. For two points $p = (p_x, p_y)$ and $q = (q_x, q_y)$, let $\min(p, q)$ be the point $(\min\{p_x, q_x\}, \min\{p_y, q_y\})$. Figure 2 shows $\min(p, q)$ for two arrangements of p and q.

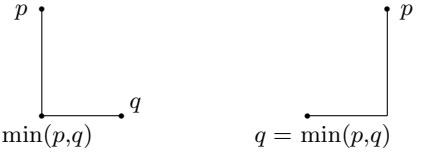

Fig. 2. $\min(p, q)$ for two arrangements of p and q

One iteration of the heuristic identifies the points p and q for which $\|\min(p, q)\|$ is largest, replaces p and q by $\min(p, q)$, and appends the horizontal and vertical segments connecting p and q to $\min(p, q)$ to the developing arborescence. (One of these segments is degenerate when p and q share a horizontal or vertical line.) The heuristic repeats this step until only the origin, which is one of the points, remains.

An efficient implementation of the heuristic maintains a priority queue of the points eligible to be merged and requires time that is $O(n \log n)$. An arborescence that the heuristic finds is never more than twice as long as a shortest RSA [3].

3 A Permutation Coding

Prim's algorithm [11] builds a minimum spanning tree in a weighted, connected, undirected graph from an arbitrary vertex by repeatedly extending the tree with the lowest-weight edge between a vertex in the tree and one not yet in it. In the permutation coding, permutations of point labels represent RSAs via a decoding algorithm based on Prim's algorithm.

The decoder identifies the arborescence a permutation represents by beginning with an arborescence consisting only of the origin. It attaches the points to

the arborescence in their listed order, always increasing the arborescence's length as little as possible. The length of the resulting structure is the permutation's fitness, which we seek to minimize.

As in Prim's algorithm, the decoder keeps track of the shortest legal connection from the growing arborescence to each point not yet in it. There are three cases: an unconnected point's nearest connection may be a point below it and to its left, a horizontal segment below it, or a vertical segment to its left. When a point joins the arborescence, the decoder uses this shortest connection.

In the first case, the decoder uses the horizontal and vertical segments whose intersection lies nearer the line $y = x$, to facilitate shorter subsequent connections. The decoder records the segment or segments, accumulates their lengths, and updates the shortest-connection information for the remaining unconnected points. Because updating occurs after each point joins the arborescence, the decoder's time, like that of Prim's algorithm, is $O(n^2)$.

Figure 3 illustrates the operation of the decoder as it builds the arborescence represented by the permutation (3 8 1 2 0 7 6 5 9 4). The origin is at the figure's lower left. The solid lines are the segments the decoder identified as it attached the points 3, 8, 1, 2, and 0. The dotted lines indicate the segments added to the arborescence as the decoder attaches the next three points. These points illustrate the three cases listed above, in order.

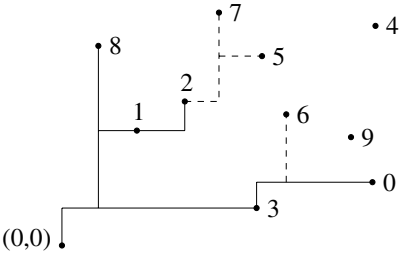

Fig. 3. A snapshot as the decoding algorithm identifies the arborescence that the permutation (3 8 1 2 0 7 6 5 9 4) represents. The solid segments attach the first five points to the arborescence. The dotted segments will connect the next three

In particular, point 7 is joined to point 2, which is below point 7 and to the left; one vertical and one horizontal segment make this connection. Next, point 6 is above a horizontal segment, to which a vertical segment joins it. Then point 5 is to the right of the recent vertical segment, to which the decoder connects it with a horizontal segment. The remaining two steps will join point 9 to the segment descending from point 6, and point 4 to point 5.

Random permutations can be generated in time that is $O(n)$. Though an earlier permutation-coded GA for this problem [12] used Goldberg and Lingle's Partially Mapped Crossover [13], the crossover operator here is Reeves' C1 [14], which has also been described, with different names, by Smith [15] and Prosser [16]. C1 chooses a crossover point at random and copies one parent into the

offspring up to that point. It then copies the remaining values in order from the second parent into the offspring's remaining positions. C1's time is $O(n)$.

Mutation chooses two positions at random in its one parent permutation and exchanges the points at those positions, thus exchanging the times at which the decoder appends the points to the arborescence. Copying the parent into the offspring requires linear time.

4 The Long Perturbation Coding

In this coding, a sequence of $2n$ floating-point perturbations, two for each point, represents a rectilinear Steiner arborescence. Identifying a genotype's RSA requires two steps. The first perturbs the points' locations: if $\mathtt{c}[\cdot]$ is a genotype and $p = (p_x, p_y)$ is the ith point, p is temporarily replaced by $(p_x + \mathtt{c}[2i], p_y + \mathtt{c}[2i+1])$. The second applies the heuristic of Rao *et al.* to the perturbed point locations, but the segments it records and their lengths (thus the length of the RSA) are based on the original coordinates. The length of the RSA is the genotype's fitness. Because the number of values in a genotype is twice the number of points, we call this the long perturbation coding.

Random genotypes consist of values from a normal distribution $N(0, \sigma_1)$. k-point or uniform crossover can be used with these genotypes. Mutation perturbs the values in the parent genotype with values from a normal distribution $N(0, \sigma_2)$. The standard deviations σ_1 and σ_2 of the two normal distributions depend on the magnitudes and ranges of the points' coordinates.

5 The Short Perturbation Coding

In the long perturbation coding, the only effect of the perturbations a genotype lists is to modify the rectilinear distances from the points to the origin. This modifies the pairs of points the decoder chooses to merge and the order in which it merges them. The same effect can be had more economically by coding the changes in the points' rectilinear magnitudes directly.

In the second perturbation coding, then, a genotype is a sequence of n floating-point values, one for each point. The rectilinear Steiner arborescence a genotype represents is identified by adding each point's value to its rectilinear distance from the origin. For a merged point $\min(p, q)$, the change in distance is the sum of the changes associated with p and q. The heuristic of Rao *et al.* uses the modified rectilinear distances to construct the RSA. Because the number of values in a genotype is n rather than $2n$, we call this the short perturbation coding.

Again, random genotypes are built of values from a normal distribution $N(0, \sigma_1)$. Positional crossover operators like k-point crossover are appropriate, and mutation modifies each value in a parent genotype with values from a normal distribution $N(0, \sigma_2)$. The values of the standard deviations again depend on the magnitudes and ranges of the points' coordinates.

6 Three Genetic Algorithms

Three generational genetic algorithms seek short rectilinear Steiner arborescences using the permutation, long perturbation, and short perturbation codings and their respective operators. All the GAs initialize their populations with random genotypes. They choose parents in tournaments without replacement and generate each offspring by applying either crossover or mutation, never both. The GAs are 1-elitist: they preserve the one best genotype of the current generation unchanged into the next. They run through a preset number of generations, then report the shortest RSA represented in their populations.

When applied to a problem instance with n points, each GA's population contained n genotypes. Each selection tournament compared two random contestants, and a tournament's winner always became a parent. The probability that crossover generated each offspring was 70%, and the probability of mutation therefore 30%. The GAs ran through $3n$ generations.

7 Comparisons

The heuristic of Rao *et al.*, the permutation-coded GA, the long-perturbation-coded GA, and the short-perturbation-coded GA were compared on twenty instances of the rectilinear Steiner arborescence problem. The instances are found in Beasley's [17] OR-Library[1], where they appear as instances of the Euclidean Steiner problem. The instances consist of random points in the unit square. The library lists fifteen instances of each of many sizes; the present algorithms were exercised on five instances each of $n = 50$, 70, 100, and 250 points.

The perturbation-coded GAs applied two-point crossover. With the long perturbation coding, the standard deviations of the initializing and mutating normal distributions were $\sigma_1 = 0.020$ and $\sigma_2 = 0.010$ when $n = 50$ and 70, $\sigma_1 = 0.010$ and $\sigma_2 = 0.005$ when $n = 100$, and $\sigma_1 = 0.004$ and $\sigma_2 = 0.002$ when $n = 250$. With the short perturbation coding, the two standard deviations were always half of these values. The standard deviations are small because the ranges of the points' coordinates are small and, as the results below show, the points need not "move" far to obtain shorter RSAs.

The heuristic of Rao *et al.* was run once on each instance, and the GAs were each run 40 independent times. Table 1 summarizes the results of these trials. For each instance, the table lists the size and number of the instance and the length of the RSA identified on it by the heuristic of Rao *et al.* For each genetic algorithm applied to each instance, the table lists the length of the shortest RSA the GA discovered, the mean of the 40 RSA lengths from its trials, and the standard deviation of those lengths.

Based on these results we make five observations. First, the permutation-coded GA cannot compete effectively with the heuristic of Rao *et al.*, particularly on the larger instances. On the 50-point instances, the GA's shortest arborescences are sometimes slightly shorter than those returned by the heuristic, but on

[1] `mcsmga.ms.ic.ac.uk/info.html`

Table 1. Results of the trials of the heuristic of Rao *et al.* and the three genetic algorithms. For each instance, the table lists its number of points, number, and the length of the heuristic's RSA. For each GA on each instance, it lists the length of its best RSA and the mean $\overline{X}$ and standard deviation s of the GA's 40 RSA lengths

Instance		Rao	Permutations			Long Perturbations			Short Perturbations		
n	num	*et al.*	best	$\overline{X}$	s	best	$\overline{X}$	s	best	$\overline{X}$	s
50	1	7.163	7.115	7.294	0.144	7.072	7.075	0.006	7.072	7.074	0.004
	2	6.576	6.569	6.855	0.171	6.524	6.534	0.009	6.524	6.525	0.002
	3	6.589	6.665	6.970	0.187	6.590	6.590	0.002	6.590	6.590	0.000
	4	6.509	6.384	6.657	0.159	6.272	6.281	0.010	6.271	6.273	0.002
	5	6.771	6.800	7.064	0.140	6.687	6.687	0.002	6.687	6.687	0.002
70	1	7.971	8.045	8.328	0.151	7.836	7.853	0.017	7.837	7.851	0.016
	2	7.594	7.789	8.015	0.174	7.501	7.505	0.007	7.501	7.502	0.003
	3	7.483	7.610	8.115	0.215	7.462	7.470	0.010	7.462	7.462	0.001
	4	7.835	7.894	8.165	0.194	7.643	7.661	0.011	7.643	7.655	0.009
	5	7.121	7.231	7.587	0.193	7.114	7.125	0.009	7.114	7.118	0.003
100	1	8.870	9.127	9.418	0.155	8.869	8.869	0.002	8.869	8.869	0.000
	2	9.161	9.400	9.753	0.191	9.003	9.006	0.003	9.003	9.004	0.001
	3	9.039	9.324	9.715	0.212	9.027	9.029	0.002	9.027	9.029	0.001
	4	9.408	9.272	9.658	0.156	9.046	9.050	0.006	9.049	9.055	0.009
	5	8.840	9.141	9.570	0.198	8.810	8.810	0.001	8.810	8.810	0.000
250	1	14.158	14.811	15.373	0.247	13.993	13.998	0.006	13.995	14.010	0.005
	2	14.358	14.880	15.346	0.189	13.977	13.987	0.005	13.984	13.998	0.008
	3	13.953	14.635	15.100	0.241	13.802	13.808	0.006	13.807	13.822	0.009
	4	14.277	14.881	15.296	0.187	14.017	14.024	0.006	14.023	14.050	0.016
	5	14.442	15.031	15.455	0.179	14.222	14.231	0.006	14.227	14.243	0.009

average its RSAs range from 1.8% to 5.8% longer, and the GA's relative performance deteriorates as the instances grow. On the 250-point instances, its shortest RSAs average 4.3% longer than the heuristic's, and its average RSA lengths range from 6.9% to 8.6% longer. Though the permutation coding's Prim-based decoding algorithm greedily joins points to arborescences with short connections, it is not as effective as the heuristic of Rao *et al.* The permutation-coded GA carries out a lot of computation for no discernible benefit.

Second, the results of the two perturbation-coded GAs are very similar, particularly on the smaller instances. Their mean results differ more than their best results, but those differences are still small relative to the standard deviations of the sets of RSA lengths. On the 250-point instances, the long-perturbation-coded GA enjoys a small but consistent advantage over the short-perturbation-coded GA, but in general it makes little difference to the perturbation decoding algorithm whether the choices of points to merge are modified by jiggling the points' positions or their distances from the origin.

Third, both perturbation-coded GAs almost always return shorter arborescences than those discovered by the heuristic of Rao *et al.* Their shortest arborescences are shorter than the heuristic's arborescences on every instance except

the third with 50 points, and the average lengths of their trials' arborescences are shorter than the heuristic's results on every instance except that one and, for the long-perturbation-coded GA only, one more. This illustrates the effectiveness of the heuristic of Rao *et al.* and suggests that perturbation codings should be effective in GAs for other geometric problems for which good heuristics exist.

Fourth, the advantage of the perturbation-coded GAs over the heuristic is small. On none of the instances does it exceed 3.9% (the long-perturbation-coded GA on the fourth 100-point instance), and it is usually less. This supports Córdova and Lee's observation [4] that the heuristic of Rao *et al.* often identifies short RSAs. It also explains the small standard deviations of the initializing and mutating distributions of both perturbation-coded GAs. The test instances' points lie in the unit square, and the heuristic is effective, so only small perturbations of points' locations or distances from the origin are necessary to find slightly shorter arborescences.

Last, the performance of the perturbation-coded GAs does not deteriorate as the instances get larger. The lengths of the GAs' arborescences are on average 1.0% to 1.6% shorter than the heuristic's RSAs, regardless of the size of the instances.

The genetic algorithms are, of course, much slower than the heuristic of Rao *et al.* The perturbation-coded GAs execute the heuristic on every evaluation; with population size n and running through $3n$ generations, they perform $3n^2$ such evaluations.

Figure 4 shows, for the first 250-point instance, the rectilinear Steiner arborescence found by the heuristic of Rao *et al.* and the best RSAs found by the three GAs. As reported in Table 1, the heuristic's RSA has length 14.158. The permutation-coded GA's RSA has length 14.811, 4.6% longer than the heuristic's RSA. The long-perturbation-coded and short-perturbation-coded GAs trees have lengths 13.993 and 13.995, about 1.2% shorter than the heuristic's RSA.

8 Conclusion

A rectilinear Steiner arborescence connects the plane's origin to points in its first quadrant with horizontal and vertical directed edges that lead up and to the right. The search for RSAs of minimum total length, an NP-hard problem, has applications to VLSI design.

The greedy heuristic of Rao *et al.* identifies arborescences that are often near-optimum. Three genetic algorithms encode RSAs as permutations of their points, as perturbations of the points' locations, and as perturbations of the points' rectilinear distances from the origin. The permutation-coded GA uses a Prim-like decoder to identify the RSAs that permutations represent. The two perturbation-coded GAs decode sequences of perturbations via the greedy heuristic.

In tests on twenty sets of 50 to 250 points, the permutation-coded GA identified arborescences longer than those returned by the greedy heuristic, and its

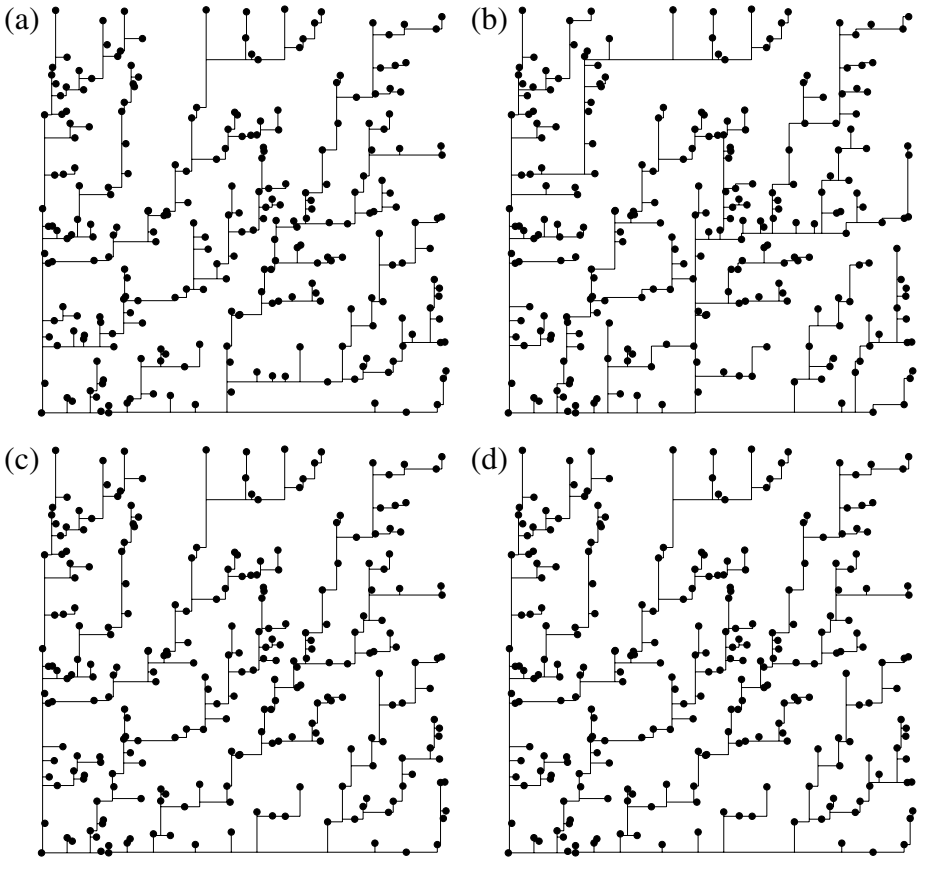

Fig. 4. On the first instance with $n = 250$ points: The RSA identified by the heuristic of Rao *et al.* (a); it has length 14.158, and the shortest RSAs found by the permutation-coded GA (b) (14.811), the long-perturbation-coded GA (c) (13.993), and the short-perturbation-coded GA (d) (13.995)

disadvantage grew with the number of points. Changing this GA's parameter values or crossover operator might improve its performance, but it is unlikely that it can ever compete effectively with the greedy heuristic. The two perturbation-coded GAs returned nearly identical results that almost always improved the heuristic's, and they maintained this advantage on the larger instances. These results illustrate both the effectiveness of the greedy heuristic and the usefulness of perturbation codings in evolutionary algorithms for geometric problems like the search for short rectilinear Steiner arborescences.

References

1. Cong, J., Khang, A.B., Leung, K.S.: Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **17** (1998) 24–39
2. Shi, W., Su, C.: The rectilinear Steiner arborescence problem is NP-complete. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms. (2000) 780–786
3. Rao, S.K., Sadayappan, P., Hwang, F.K., Shor, P.W.: The rectilinear Steiner arborescence problem. Algorithmica **7** (1992) 277–288
4. Córdova, J., Lee, Y.H.: A heuristic algorithm for the rectilinear Steiner arborescence problem. Technical Report TR-94-025, Department of Computer Science, University of Florida (1994)
5. Leung, K.S., Cong, J.: Fast optimal algorithms for the minimum rectilinear Steiner arborescence problem. In: Proceedings of the International Symposium on Circuits and Systems. (1997) 1568–1571
6. Ramnath, S.: New approximations for the rectilinear Steiner arborescence problem. IEEE Transactions on Computer-Aided Design **22** (2003) 859–869
7. Julstrom, B.A.: Encoding rectilinear Steiner trees as lists of edges. In Lamont, G.B., Yfantis, E.A., Haddad, H., Papadopoulos, G.A., Carroll, J., eds.: Proceedings of the 16th ACM Symposium on Applied Computing, New York, ACM Press (2001) 356–360
8. Julstrom, B.A.: A hybrid evolutionary algorithm for the rectilinear Steiner problem. In Barry, A., ed.: 2003 Genetic and Evolutionary Computation Workshop Program, Chicago, IL (2003) 49–55
9. Valenzuela, C.L., Williams, L.P.: Improving simple heuristic algorithms for the traveling salesman problem using a genetic algorithm. In Bäck, T., ed.: Proceedings of the Seventh International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufmann Publishers (1997) 458–464
10. Cohoon, J.P., Karro, J.E., Martin, W.N., Niebel, W.D.: Perturbation method for probabilistic search for the traveling salesperson problem. In: Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation. Volume 3455 of Proceedings of SPIE. SPIE Press (1998) 118–127
11. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal **36** (1957) 1389–1401
12. Julstrom, B.A., Antoniades, A.: Two hybrid evolutionary algorithms for the rectilinear Steiner arborescence problem. In: Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus (2004)
13. Goldberg, D.E., Robert Lingle, J.: Alleles, loci, and the traveling salesman problem. [18] 154–159
14. Reeves, C.R.: A genetic algorithm for flowshop sequencing. Computers and Operations Research **22** (1995) 5–13
15. Smith, D.: Bin packing with adaptive search. [18] 202–207
16. Prosser, P.: A hybrid genetic algorithm for pallet loading. In: Proceedings of the 8th European Conference on Artificial Intelligence, London, Pitman (1988)
17. Beasley, J.E.: OR-library: Distributing test problems by electronic mail. Journal of the Operational Research Society **41** (1990) 1069–1072
18. Greffenstette, J.J., ed.: Proceedings of the First International Conference on Genetic Algorithms. In Greffenstette, J.J., ed.: Proceedings of the First International Conference on Genetic Algorithms, Hillsdale, NJ, Lawrence Erlbaum (1985)