J. Opl Res. Soc. Vol. 44, No. 4, pp. 397-406
Printed in Great Britain.

0160-5682/93 $8.00+0.00

Solving the Graphical Steiner Tree Problem Using Genetic Algorithms

A. KAPSALIS, V. J. RAYWARD-SMITH and G. D. SMITH
School of Information Systems, University of East Anglia

We develop a genetic algorithm (GA) to solve the Steiner Minimal Tree problem in graphs. To apply the GA paradigm, a simple bit string representation is used, where a 1 or 0 corresponds to whether or not a node is included in the solution tree. The standard genetic operators—selection, crossover and mutation—are applied to both random and seeded initial populations of representations. Various parameters within the algorithm have to be set and we discuss how and why we have selected the values used. A standard set of graph problems used extensively in the comparison of Steiner tree algorithms has been solved using our resulting algorithm. We report our results (which are encouragingly good) and draw conclusions.

INTRODUCTION

This paper describes the application of a genetic algorithm to the Steiner Tree Problem in graphs. The following section briefly reviews the problem and summarizes the various techniques previously used to solve it. We then describe a general genetic algorithm and set the parameters necessary in order to apply it to the Steiner Tree problem in graphs. Finally, we present the results of applying the algorithm to a standard set of test graphs.

THE GRAPHICAL STEINER TREE PROBLEM

Let $G = (V, E)$ be an undirected graph with a finite set of vertices, V, and an edge set, E. Let $c: E \rightarrow R^+$ be a cost function assigning a positive real to each edge in G. Assume we have a set, $S \subseteq V$, of special vertices. The *Steiner minimal tree problem in graphs* (GSTP) is that of finding a subgraph, $G' = (V', E')$, of G such that

(i) V' contains all the vertices in S;
(ii) G' is connected; and
(iii) $\Sigma\{c(e): e \in E'\}$ is minimal.

This connected subgraph must necessarily be a tree. It may contain vertices other than those in S; such vertices, i.e. those in $V' - S$, are called Steiner vertices.

As an example, consider the graph of Figure 1(a). The special vertices are those numbered 1, 2, 3, 6 and 7 and are shaded in the figure. The Steiner tree will include two Steiner vertices (numbered 4 and 5) and has a cost of six (see Figure 1(b)).

The GSTP has attracted considerable research interest especially over the last three decades. It has become one of the 'classic' combinatorial optimization problems alongside other famous problems such as the Travelling Salesman Problem (see, for example, Lawler *et al.*[1]) and the Knapsack Problem (see, for example, Martello and Toth[2]). In 1972, Karp[3] proved the decision problem associated with GSTP to be NP-complete and hence the problem is most unlikely to be solved by a polynomial time algorithm. Several exponential time algorithms have been proposed[4-8] but most of the research effort has been concentrated on polynomial time algorithms for 'good', suboptimal solutions. Early ideas are to be found in Takahashi and Matsuyama[9], Kou *et al.*[10], Plesnik[11] and Rayward-Smith and Clare[12]. There are also some recent developments[13-15] concerning simplification strategies which can be used to reduce the number of vertices and edges that need to be considered. More recently, promising algorithms have been described by Beasley[16],

Correspondence: A Kapsalis, School of Information Systems, University of East Anglia, Norwich, UK

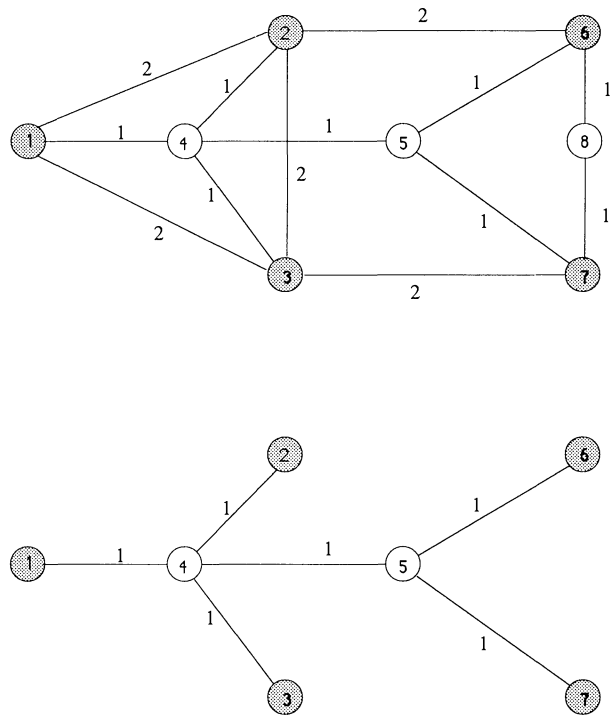

FIG. 1. *(a) Graph showing the special vertices (shaded) and weights. (b) The Steiner tree solution to the graph in (a).*

Voss[17] and Winter and MacGregor Smith[18]. Dowsland[19] described a simulated annealing approach to the solution of the GSTP.

A comprehensive overview of the GSTP is to be found in Winter[20] and Hwang and Richards[21]. Applications of the problem to areas such as topological network design, printed circuit design, multiprocessor scheduling and phylogeny are described in Foulds and Rayward-Smith[8], and Winter[20].

For some special cases, the GSTP can be solved in polynomial time. In particular, if $|S| = 2$, GSTP reduces to a simple shortest path problem and can be solved using Dijkstra's algorithm[22]. If $S = V$ then GSTP reduces to finding the minimum spanning tree (MST) of G and we can use either the algorithm of Prim[23] or Kruskal[24]. The fundamental problem in the general case is that we do not know which vertices are Steiner vertices. Once we have determined the Steiner vertices, Z (say), the required Steiner tree is simply the MST of the subgraph of G induced by $S \cup Z$. Thus, in Figure 1, the Steiner tree is the MST of the subgraph induced by $\{1, 2, 3, 4, 5, 6, 7\}$. Most of the heuristics try to select the Steiner vertices using some measure of desirability. The average distance heuristic (ADH) of Rayward-Smith and Clare[12] uses this approach and is one of the most successful to date. It has been tested on a range of standard test data[25] as well as randomly generated data.

One of the strengths of the genetic algorithm approach is that it is usually easily modified to handle variants of the original problem. One variant of GSTP includes the incorporation of variable (and possibly non-linear) costs which are associated with the use of vertices in $V - S$. A straightforward modification of our genetic algorithm will handle this situation.

Other variants of GSTP include the NP-hard, rectilinear Steiner tree problem (see, for example, Aho *et al.*[26], Garey and Johnson[27]) and the directed Steiner tree problem (see, for example, Nastansky *et al.*[28], Wong[29], and Maculan *et al.*[30]). The application of GAs to these cases is also reasonably easy and the details are currently being researched.

Also, GSTP is the graph theoretical counterpart to an original problem in Euclidean geometry, namely the problem of connecting points in Euclidean space using as little 'ink' as possible. This problem too is NP-hard (see Garey *et al.*[31]). A genetic algorithm for its solution is described in Hesser *et al.*[32].

THE GENETIC ALGORITHM

The term Genetic Algorithm (or GA) refers to a class of adaptive search procedures based on principles derived from the dynamics of natural population genetics. The foundations of the technique were described by Holland[33], who also established much of the theory to explain the subsequent success of the application of GAs to a wide variety of problems, including pattern recognition, classifier systems, network configuration, prisoners' dilemma and other game problems, control of industrial systems and general combinatorial optimization. Liepins and Hilliard[34] provided a very good review of the major developments in the application of GAs, while Goldberg[35] is an ideal starter kit for anyone wishing to know more about how and why they work. Reeves[36] gives a good description of the GA, including the major problem areas in which it has been used.

A GA is a control system which is able to encode complex structures, representing solutions to the problem at hand, in simple representations such as bit strings, and to perform transformations on these representations in such a way as to evolve better solutions to the problem.

Typically, a GA comprises the following:

- a (chromosomal) representation of a solution to the specified problem.
- an initial population of solutions taken (sampled) from the totality of possible solutions to the problem;
- the competitive selection of solution representations for reproduction based on some evaluation function (equivalent to survival of the fittest);
- idealized genetic operators that recombine the selected representations to create new structures for possible inclusion in the population;
- a replacement strategy to maintain a steady population.

The above parameters must be determined prior to applying the GA to the solution of the problem in hand.

Firstly, each individual solution must be uniquely represented in a manner which is both appropriate to the application and amenable to the GA. In most applications, a bit string representation is sufficient and indeed preferable. The representation chosen is a bit string of size equal to $|V|$ in which each bit position i corresponds to a node v_i, and a '1' means that vertex $v_i \in V'$, a '0' means that $v_i \notin V'$. For example, referring to Figure 1, we see that the bit string 11100110 represents the subgraph containing only the special vertices, while 11111110 represents the Steiner tree for this problem. We must ensure that the solution includes all the special vertices, S. This can be achieved either by penalizing any chromosome not including all elements of S, or by explicitly ORing each chromosome with the chromosome representing S. We have experimented with each of these options and have found the second option marginally preferable, both in terms of speed and simplicity of algorithm. Under the circumstances, we could reduce the size of the representation to a bit string of size $|V-S|$, representing non-special vertices only, some of which may become Steiner vertices. Another possibility would be to represent edges rather than nodes. Although this would lead to improved individual chromosome evaluation times, it also introduces a much larger search space with a disproportionate increase in infeasible areas.

Secondly, an initial population of solutions is required in order to start the process. A number of experimental tests were carried out, with different techniques used to generate the initial population. In the first set, the population was created at random, while in the second set,this randomly created population was seeded by including a feasible solution equivalent to the MST of the entire graph, i.e. a string of 1s. In the third run, a heuristic solution derived from trimming this MST was included.

The question remains; how large a population should we consider? The population size is a major factor in the effectiveness of the algorithm. If the size is too small, then a relatively small part of the solution space is searched leading to fast convergence but with a higher probability of convergence to a local optimum. If, on the other hand, the size is too large, then a disproportionate amount of computation time is required per generation, and consequently per run. Recent results[37,38] point to the successful use of relatively small populations, and our experiments show that increasing the population size beyond ten, although increasing the computational effort,

is not rewarded by a corresponding increase in performance. We therefore use a population size of ten.

Thirdly, in order for solutions to vie with one another for survival of the fittest, it is necessary that each solution has an associated value. Given a chromosome representing a set, U, of vertices, the evaluation proceeds as follows. Firstly, we construct the subgraph of G induced by $U \cup S$. Say this subgraph comprises k ($\geqslant 1$) components. We compute the MST of each component using Prim's algorithm[23] and sum these values. If $k > 1$, we add a large penalty linearly dependent on k. The problem is converted to a maximization problem by subtracting the resultant fitness from an offset, the largest fitness value observed in the population. Each MST evaluation is $O(m^2)$, where m ($\leqslant n$) is the number of 1s in the chromosome and n is the length of the chromosome. With a population size p, the computation per generation cycle is at most $O(pm^2)$. In our case, although we have set $p = 10$, the number of evaluations per generation is approximately six, due to storage of the previous evaluations.

Once the fitness function has been determined for the problem in hand, individual solutions in the initial population are stochastically selected (with replacement) to join a gene pool to produce the next generation of solutions. Thus, individual solutions with above average fitness may contribute multiple copies to the gene pool, while those with well below average fitness may not contribute at all. Two mechanisms for selecting solutions to join the gene, or parent pool were tested. These are:

Roulette: in which a solution is selected with a probability equal to its relative fitness with respect to the whole population, see for example Goldberg[35];
Fibonacci: in which a solution is selected according to its position in the ordered set of solutions, but using a scaled Fibonacci sequence to favour those solutions which are higher in the order, i.e. the better solutions.

Kapsalis[39] has looked at a number of other selection mechanisms for this problem.

The breeding cycle consists of creating a new pool of solutions, the offspring pool, from the gene pool by applying genetic operators to individual chromosomes, or pairs of chromosomes in the gene pool. The genetic operators used are *crossover* and *mutation*, see for example Goldberg[35]. Tests were carried out with different probabilities of applying crossover and mutation to the selected parents. With crossover, it was found that varying the probability from 50% to 100% had little effect on performance, with a value of 85–90% being marginally optimal for the tests carried out. A value of 90% is used in the main runs. For mutation, values between 1% and 4% were seen to be giving better results than typically smaller values. A value of 2% is used in the main runs. However, it is to be noted that the GA is robust in the sense that the solutions to the test problems were achieved on the whole with a wide range of parameter values, and with no fine tuning required to achieve optimality.

The breeding cycle continues until the offspring pool has been generated, and is equal in size to the gene pool. In most applications of GAs, this new generation simply replaces the original population pool. However, it is possible to incorporate a replacement strategy to replace systematically some or all of the old population by the new solutions found. The strategies used here to generate the new population pool are:

(a) All new solutions replace solutions in the old population (Replace-all).
(b) Best for worst replacement, i.e. systematically replace the worst solution in the old population by the newly generated solution, until no improvement is attained (Best for worst).

Once the new population has been created, this becomes the population pool for the next generation and the whole generation cycle is repeated many times until there is no marked improvement in the fitness value of the best individual, or some other stopping criterion has been met. The cycle is illustrated in Figure 2.

Of course, there are many variations of this basic GA, including alternative selection and replacement mechanisms such as selection without replacement[40] and according to rank[41]. In one of Holland's original versions, each offspring replaces a randomly chosen member of the existing population as soon as it is created. The software GENITOR[42] is a genetic search algorithm

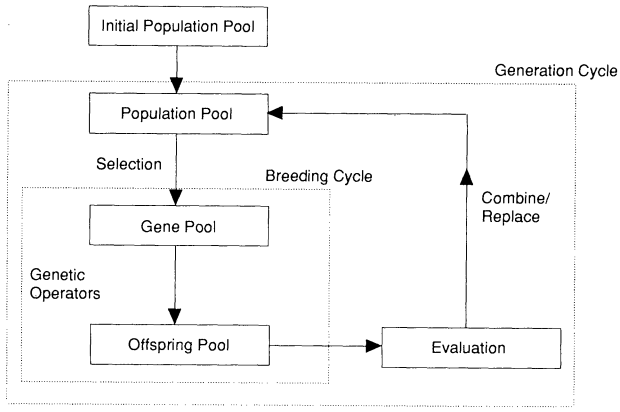

FIG. 2. *A schematic diagram of the Genetic Algorithm.*

which uses selection according to rank, and each new structure created replaces a randomly chosen structure in the old population. Thus, parents and offspring can coexist in the same generation.

RESULTS

The GA has been tested on the B-problem set from Beasley[25]. The advantage of using such a set is that the results can be compared, to some extent, with other methods used, and that they are readily available through electronic mail. The disadvantage is that they are all sparse.

The GA was run on this set of data five times for each problem and for a wide variety of settings of the various parameters. In all cases, the GA stopped either when the known optimal solution was found or when the time limit (of 4000 seconds) was reached. This time was recorded, along with the solution obtained and the number of chromosome evaluations. In the cases where an optimal solution was not discovered, we also recorded the time at which the best solution found was first discovered. We shall refer to this as the *last improvement time.*

From a wide range of experiments, we have come to the conclusion that the initial pool is best chosen to be random or MST seeded, the safest selection strategy is the Roulette strategy but that the Fibonacci strategy can work exceptionally well on occasions. Of the two replacement strategies, the Replace-all strategy appears to give better quality solutions. We will illustrate these claims with some typical results.

In Table 1, we give an outline description of the 18 problems, including the number of nodes $|V|$, the number of edges $|E|$, the number of special nodes $|S|$ and the optimal cost. In addition we give the resulting cost achieved by previously tried techniques[12], as well as the best results achieved by the GA with some or other settings of the parameters. These are expressed as percentage deviations from the optimal cost. Here, the data in the column headed 'Trim' is that derived by Kapsalis[39] from pruning the MST of the graph G, i.e. removing all vertices of degree 1. SDH is the Shortest Distance Heuristic, SPH the Shortest Path Heuristic and ADH the Average Distance Heuristic; see Rayward-Smith and Clare[12] for details of the algorithms.

It is to be noted from Table 1 that, using one or other of the parameter setting the GA finds the optimal solution *in all of the test problems.*

Figure 3 shows the best solution found for each of the choices of initial population, namely randomly generated, random with an MST seed and random with a trimmed MST seed. For this experiment, the selection method and the replacement method were fixed (Roulette and Replace-all respectively). The results show that, *for each setting*, the GA finds the optimal solution in all but 2 or 3 problems. In the cases when it does not find the optimal solution, it is never more than 7.3% off the optimal and typically within 2%. Over the entire set of experiments used to produce Figure 3, the GA found the optimal solution approximately 70% of the time.

TABLE 1. *Outline of the 18 problems, including the optimal cost, and percentage deviation from this cost using a variety of heuristics. See Rayward-Smith and Clare[12] for details of SDG, SPH and ADH. The final column is the best result obtained using one or other of the settings of the genetic algorithm*

Problem	$\|V\|$	$\|E\|$	$\|S\|$	Optimal cost	Trim (% dev)	SDG (% dev)	SPH (% dev)	ADH (% dev)	GA (% dev)
1	50	63	9	82	8.54	0	0	0	0
2	50	63	13	83	15.66	8.43	0	0	0
3	50	63	25	138	4.35	1.45	0	0	0
4	50	100	9	59	6.78	8.47	5.08	5.08	0
5	50	100	13	61	11.48	4.92	0	0	0
6	50	100	25	122	6.56	4.92	3.28	1.64	0
7	75	94	13	111	0.90	0	0	0	0
8	75	94	19	104	6.73	0	0	0	0
9	75	94	38	220	3.18	2.27	0	0	0
10	75	150	13	86	10.47	13.95	4.65	4.65	0
11	75	150	19	88	17.05	2.27	2.27	2.27	0
12	75	150	38	174	6.90	0	0	0	0
13	100	125	17	165	14.55	6.06	7.88	4.24	0
14	100	125	25	235	6.38	1.28	2.55	0.43	0
15	100	125	50	318	4.72	2.20	0	0	0
16	100	200	17	127	34.65	7.87	3.15	0	0
17	100	200	25	131	9.92	5.34	3.82	3.05	0
18	100	200	50	218	4.13	4.59	1.83	0	0

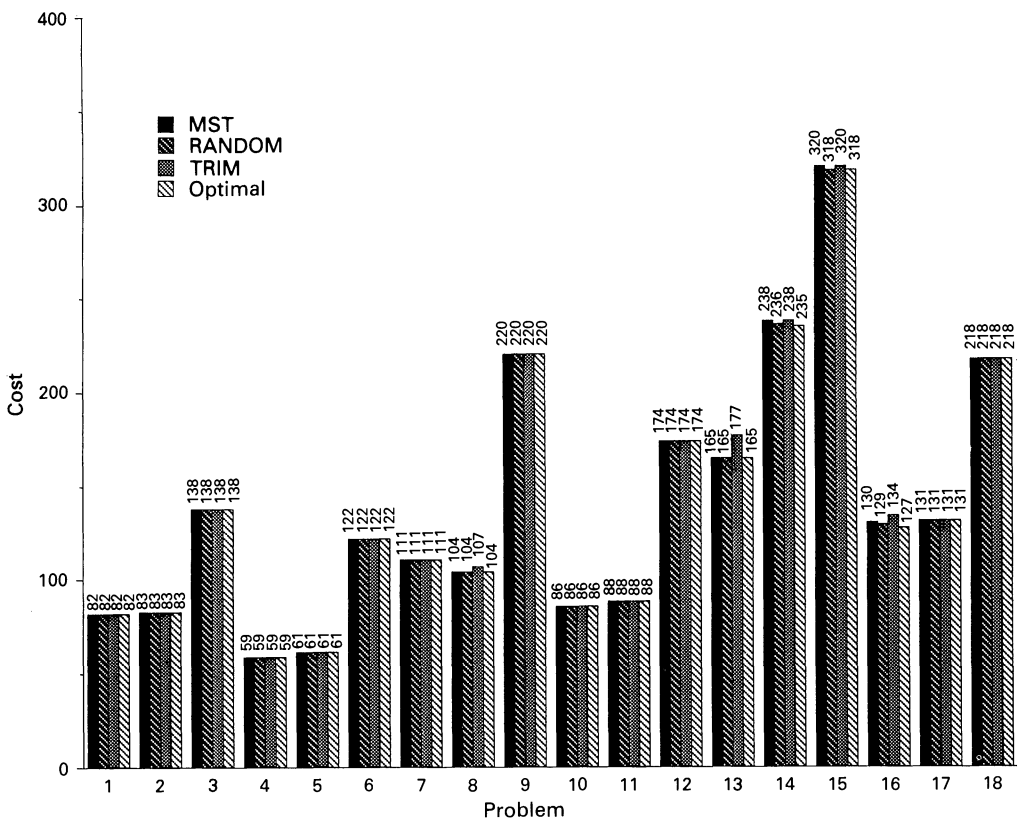

FIG. 3. *The best solution found over five runs for each of the 18 problems and for different types of initial population. (The selection method used is Roulette).*

Figure 4 shows a graph of the cost of the best solution found versus time for the three types of initial populations for a typical run of problem 1. It can be seen that the randomly generated

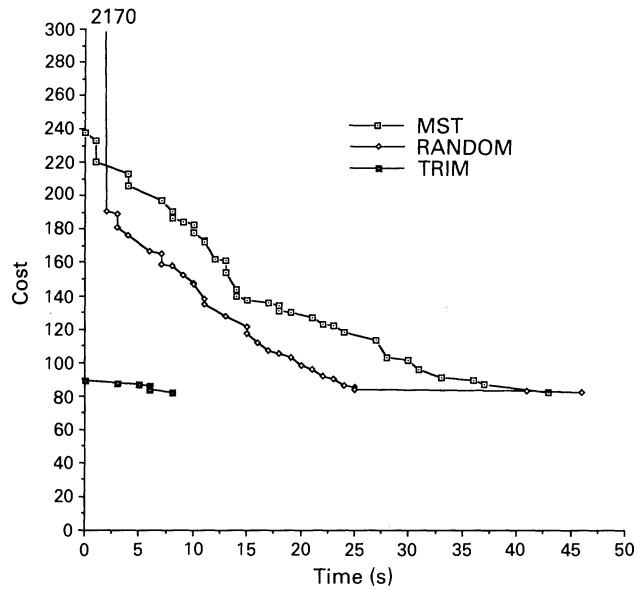

FIG. 4. *Cost against Time for Problem 1 and for different types of initial population, showing the improvement in solution with each generation. (Roulette selection).*

population converged quicker than the MST seeded population, with Trim converging fastest. This pattern repeated itself for most of the problems.

We now consider the last improvement time (LIT) for each problem with a variety of settings of the initial population, selection mechanism and replacement strategy. In Figure 5, we show the minimum LITs obtained during the five runs, for each of the initial population choices but with selection and replacement methods fixed (Roulette and Replace-all as before), while in Figure 6 we show the average LITs. Although the Trim seeded initial population gave slightly worse results in terms of the solution found (see Figure 3), it is generally faster in operation. This is because it tends to converge to a local suboptimal solution.

There appears to be little to choose between a random initial population and a MST seeded population, either in terms of quality of solution or LITs. We fixed the choice of initial population to be MST seeded and the replacement strategy to be Replace-all. We then considered the two different selection methods used, i.e. Roulette or Fibonacci. The results in Figure 7 show the average and minimum LITs for each problem. It can be seen that, although the best (or minimum)

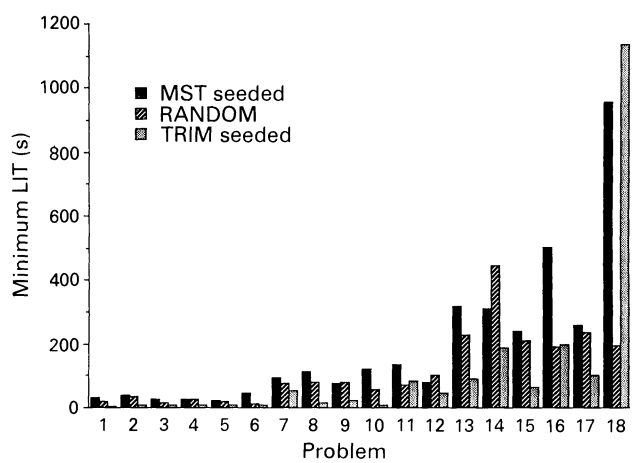

FIG. 5. *The minimum Last Improvement Times (LITs) over five runs for each problem using different types of initial population. (Roulette selection).*

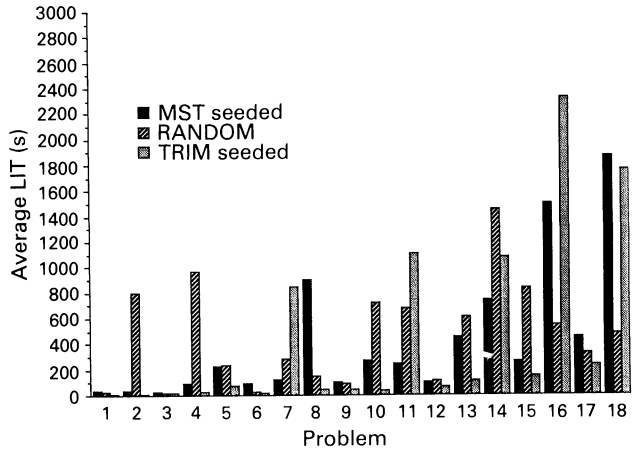

FIG. 6. *The average Last Improvement Times (LITs) over five runs for each problem using different types of initial population. (Roulette selection).*

Fibonacci method is generally better than the best Roulette method for each problem, the average Fibonacci performance is worse due to the more erratic behaviour of this technique.

All of the above experiments were carried out on an Apple Mac IIfx.

The results shown here for Problems 13 through 18 can be compared, to a limited extent, with the results of Dowsland[19], who used a combination of reduction and simulated annealing to solve randomly generated graphs, each with 100 vertices. Although the average times are comparable, the minimum times achieved using GA can be dramatically better than the convergence time achieved by simulated annealing.

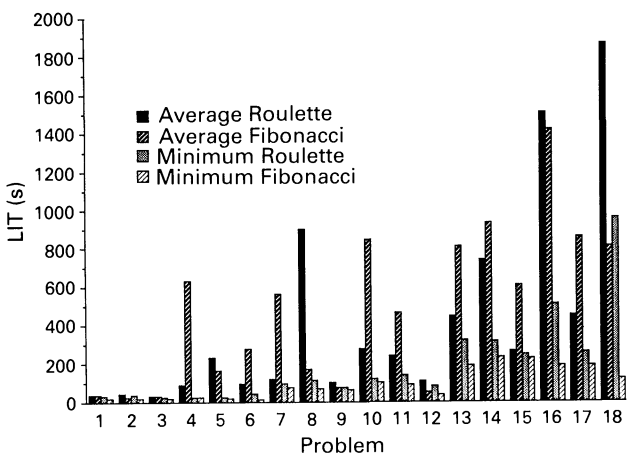

FIG. 7. *The minimum and average Last Improvement Times (LITs) over five runs for each problem using different selection methods. (Initial population is MST, while replacement strategy is Replace-all.).*

CONCLUSIONS AND IDEAS FOR FURTHER RESEARCH

Our main conclusion is that Genetic Algorithms provide a remarkably successful method for finding solutions to the Steiner tree problem in sparse graphs. We also expect GAs to be successful in non-sparse graphs and a range of other NP-hard combinatorial optimization problems. We are currently researching this.

In applying the GA paradigm to the solution of the Steiner Tree problem in graphs, it is found that the technique is extremely robust in that the solutions to a set of test problems

were invariably found even with relatively large variations in the values of the parameters used. In practice, however, lengthy experimentation to fine-tune parameters is not a cost-effective exercise.

We are currently developing kernel software to support the Genetic Algorithm paradigm on a range of architectures. Since GA is intrinsically parallel, one of these architectures will be a Meiko transputer rack. We envisage our software being available in the late 1992 and enabling a wide number of applications to be easily implemented.

REFERENCES

1. E. L. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN and D. S. SHMOYS (1985) *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley, New York.
2. S. MARTELLO and P. TOTH (1990) *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley, New York.
3. R. M. KARP (1972) Reducibility among combinatorial problems. In *Complexity of Computer Computations* (R. E. MILLER and J. W. THATCHER, Eds) pp 85–103. Plenum Press, New York.
4. A. LEVIN (1971) Algorithm for the shortest connection of a group of graph vertices. *Soviet Math. Dokladi* **12**, 1477–1481.
5. S. L. HAKIMI (1971) Steiner's problem in graphs and its implications. *Networks* **1**, 113–133.
6. S. E. DREYFUS and R. A. WAGNER (1972) The Steiner problem in graphs. *Networks* **1**, 195–207.
7. M. L. SHORE, L. R. FOULDS and P. B. GIBBONS (1982) An algorithm for the Steiner problem in graphs. *Networks* **12**, 323–333.
8. L. R. FOULDS and V. J. RAYWARD-SMITH (1983) Steiner problems in graphs: algorithms and applications. *Eng. Opt.* **7**, 7–16.
9. H. TAKAHASHI and A. MATSUYAMA (1980) An approximate solution for the Steiner problem in graphs. *Math. Japonica* **24**, 573–577.
10. L. KOU, G. MARKOWSKY and L. BERMAN (1981) A fast algorithm for Steiner trees. *Acta Informatica* **16**, 141–145.
11. J. PLESNIK (1981) A bound for the Steiner tree problem in graphs. *Math. Slovaca* **31**, 155–163.
12. V. J. RAYWARD-SMITH and A. CLARE (1986) On finding Steiner vertices. *Networks* **16**, 283–294.
13. J. E. BEASLEY (1984) An algorithm for the Steiner problem in graphs. *Networks* **14**, 147–159.
14. A. BALAKRISHNAN and N. R. PATEL (1987) Problem reduction methods and a tree generation algorithm for the Steiner network problem. *Networks* **17**, 65–85.
15. C. W. DUIN and A. VOLGENANT (1989) Reduction tests for the Steiner problem in graphs. *Networks* **19**, 549–567.
16. J. E. BEASLEY (1989) An SST-based algorithm for the Steiner problem in graphs. *Networks* **19**, 1–16.
17. S. VOSS (1992) Steiner's problem in graphs: heuristic methods. To appear in *Disc. Appl. Math.*
18. P. WINTER and J. MACGREGOR SMITH (1992) Path-distance heuristics for the Steiner problem in undirected networks. *Algorithmica* **7**, 309–327.
19. K. A. DOWSLAND (1991) Hill-climbing simulated annealing and the Steiner problem in graphs. *Eng. Opt.* **17**, 91–107.
20. P. WINTER (1987) The Steiner problem in networks: a survey. *Networks* **17**, 129–167.
21. F. K. HWANG and D. S. RICHARDS (1992) Steiner tree problems. *Networks* **22**, 55–89.
22. E. W. DIJKSTRA (1959) A note on two problems in connection with graphs. *Numer. Math.* **1**, 269–271.
23. R. C. PRIM (1957) Shortest connection networks and some generalizations. *Bell System Tech. J.* **36**, 1389–1401.
24. J. B. KRUSKAL (1956) On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. Am. Math. Soc.* **7**, 48–50.
25. J. E. BEASLEY (1990) OR-Library: distributing test problems by electronic mail. *J. Opl Res. Soc.* **41**, 1069–1072.
26. A. V. AHO, M. R. GAREY and F. K. HWANG (1977) Rectilinear Steiner trees: efficient special case algorithms. *Networks* **7**, 37–58.
27. M. R. GAREY and D. S. JOHNSON (1977) The rectilinear Steiner tree problem is NP-Complete. *SIAM J. Appl. Math.* **32**, 826–834.
28. I. NASTANSKY, S. M. SELKOW and N. F. STEWART (1974) Cost-minimal treest in directed acyclic graphs. *Z. für OR* **18**, 39–67.
29. R. T. WONG (1984) A dual ascent approach for Steiner tree problems on a directed graph. *Math. Program.* **28**, 271–287.
30. N. MACULAN, P. SOUZA and CANDIA-VEJAR (1991) An approach to the Steiner problem in directed graphs. *Annal. Opns Res.* **33**, 471–480.
31. M. R. GAREY, R. L. GRAHAM and D. S. JOHNSON (1977) The complexity of computing Steiner minimal trees. *SIAM J. Appl. Math.* **34**, 477–495.
32. J. HESSER, R. MANNER and O. STUCKY (1989) Optimization of Steiner trees using genetic algorithms. *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. SCHAFFER, Ed.) 231–236.
33. J. H. HOLLAND (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
34. G. E. LIEPINS and M. R. HILLIARD (1989) Genetic algorithms: foundations and applications. *Annal Opns Res.* **21**, 31–58.
35. D. E. GOLDBERG (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.
36. C. R. REEVES (1991) *An introduction to genetic algorithms. In OR Tutorial Papers* (A. G. MUNFORD and T. C. BAILEY, Eds) pp 69–83. Operational Research Society, Birmingham.
37. D. E. GOLDBERG (1989) Sizing populations for serial and parallel genetic algorithms. In *Proc. Third Int. Conf. Genetic Algorithms*, (J. D. SCHAFFER, Ed.) pp 70–79.

38. J. J. GREFENSTETTE (1986) Optimisation of control parameters for genetic algorithms. *IEEE Trans. Syst. Man, Cybernet.* **16**, 122–128.
39. A. KAPSALIS (1991) Steiner Trees. Masters thesis, University of East Anglia, Norwich.
40. K. A. DE JONG (1975) An analysis of the behaviour of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan. *Dissertation Abstracts International*, **36**(10), 5140B. University Microfilms No. 76-9381.
41. J. D. SCHAFFER (1985) Multiple objective optimisation with vector evaluated genetic algorithms. In *Proc. Int. Conf. Genetic Algorithms and Their Appl.* (J. J. GREFENSTETTE Ed.) pp 93–100.
42. D. WHITLEY (1989) The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proc. Third Int. Conf. on Genetic Algorithms* (J. D. SCHAFFER Ed.) pp 116–121.