Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm

Henrik Esbensen
Computer Science Department
Aarhus University
DK-8000 Aarhus C, Denmark
email: hesbensen@daimi.aau.dk

February 1994

Abstract

A new Genetic Algorithm (GA) for the Steiner Problem in a Graph (SPG) is presented. The algorithm is based on a bit-string encoding. A bitstring specifies selected Steiner vertices and the corresponding Steiner tree is computed using the Distance Network Heuristic. This scheme ensures that every bitstring correspond to a valid Steiner tree and thus eliminate the need for penalty terms in the cost function.

The GA is tested on all SPG instances from the OR-Library of which the largest graphs have 2,500 vertices and 62,500 edges. When executed 10 times on each of 58 graph examples, the GA finds the global optimum at least once for 55 graphs and every time for 43 graphs. In total the GA finds the global optimum in 77 % of all program executions and is within 1 % from the global optimum in more than 92 % of all executions.

The performance is compared to that of two branch-and-cut algorithms and one of the very best deterministic heuristics, an iterated version of the Shortest Path Heuristic (SPH-I). For all test examples but one, even the worst result ever found by the GA is equal to or better than the result of SPH-I and in many cases the average error ratio of the GA is an order of magnitude better than that of SPH-I. The runtime of the GA is moderate for all test examples. This is in contrast to SPH-I as well as the branch-and-cut algorithms, for which the runtime in some cases are extremely high.

1 Introduction

The Steiner Problem in a Graph (SPG) is one of the classic problems of combinatorial optimization. Given a graph and a designated subset of the vertices, the task is to find a minimum cost subgraph spanning the designated vertices. The SPG arises in a large variety of diverse optimization problems such as network design, multiprocessor scheduling and integrated circuit design [10, 28].

Numerous algorithms of various kinds have been developed for the SPG. Exact algorithms can be found in e.g. [2, 3, 5, 8, 13, 23, 26]. However, since the SPG is NP-complete [19] these algorithms have exponential worst case time complexities. Therefore, a significant research effort has been directed towards polynomial time heuristics, cf. e.g. [2, 20, 24, 25, 27, 31]. Simulated annealing has also been applied to SPG [7].

The Rectilinear Steiner Problem (RSP) is an important special case of SPG [14], which is still NP-complete [11]. While at least two genetic algorithms for RSP have been published [15, 17], we are aware of only one previous genetic algorithm (GA) for the SPG, developed by Kapsalis, Rayward-Smith and Smith [18].

The contribution of this paper is a new GA for the SPG which differs significantly from the approach of Kapsalis et al. [18] in a number of ways. While invalid solutions are allowed but penalized in [18], our approach is to enforce constraint satisfaction at all times, thereby eliminating the need for penalty terms in the cost function. Another major difference is our use of an inversion operator.

The performance evaluation strategies also differs significantly. While the parameter settings used in [18] varies from problem to problem, a fixed set of parameter values has been used for all results reported in this paper. From a practitioners point of view a stochastic algorithm is of limited use if it requires its parameters to be tuned every time a new problem instance is presented. Therefore we consider a fixed parameter setting to be of major importance.

The presented algorithm is tested on all SPG instances from the OR-Library [4]. This test suite consists of randomly generated graphs with up to 2,500 vertices and 62,500 edges. The obtained performance is compared to that of the GA by Kapsalis et al. [18], an iterated version of the Shortest Path Heuristic called SPH-I, which is one of the very best deterministic heuristics [31], and two recent branch-and-cut algorithms by Lucena and

Beasley [23] and Chopra, Gorres and Rao [5]. The experimental results shows the following:

- The GA presented here clearly outperforms the GA in [18] with respect to solution quality as well as runtime.
- The solution quality obtained by our GA is always at least as good as that obtained by SPH-I, and often the error ratio is an order of magnitude better. Depending on the problem, the two algorithms either require similar amounts of runtime, or the GA is significantly faster.
- As opposed to the branch-and-cut algorithms, the GA is not guaranteed to find a global optimal solution. However, the experiments reveals that the GA do find the global optimum in more than 77 % of all runs and is within 1 % from optimum in more than 92 % of all runs. While the GA is capable of finding near-optimal solutions for *all* test examples in a moderate amount of time, the runtime of the branch-and-cut algorithms varies extremely and even prevent some of the largest problem instances from being solved.

The paper is organized as follows. A precise problem definition is given in Section 2. Section 3 presents a detailed description of the developed algorithm and discusses some of the main design decisions taken. The experimental method as well as detailed experimental results are given in Section 4, and in Section 5 possible directions for future work are suggested. Finally, Section 6 concludes the paper.

2 Problem Definition

The graph terminology used in this paper is as in [1]. For a given graph $G = (V, E)$ and a subset $V' \subseteq V$, the *subgraph of G induced by V'* is a graph $G = (V', E')$, such that 1) $E' \subseteq E$, 2) $(v_i, v_j) \in E' \Rightarrow v_i, v_j \in V'$, and 3) $[v_i, v_j \in V' \wedge (v_i, v_j) \in E] \Rightarrow (v_i, v_j) \in E'$. A graph is *complete* if it has an edge between every pair of vertices. The *distance graph* of G, denoted $D(G)$, is the complete graph having the same set V of vertices, in which the cost of each edge (v_i, v_j) equals the cost of the shortest path in G from v_i to v_j. For a given edge cost function $c : E \mapsto \Re$, the *cost of a graph G* is the sum of the cost of all edges of G, and is denoted by $c(G)$. The problem considered can

now be defined:

The Steiner Problem in a Graph (SPG): Given a connected, undirected graph $G = (V, E)$, a positive edge cost function $c : E \mapsto \Re_+$, and a subset $W \subseteq V$, compute a connected subgraph $G' = (V', E')$ of G, such that $W \subseteq V'$ and such that $c(G')$ is minimal.

Any acyclic subgraph G' of G such that $W \subseteq V'$ is called a *Steiner Tree for W in G*. A solution G' with minimal cost is called a *Minimal Steiner Tree (MStT) for W in G*. The set $S \subseteq V \backslash W$ such that $V' = W \cup S$ is called the *Steiner vertices* of G'. Note the generality of this problem formulation. We do not require G to be planar, and we do not require c to satisfy the triangle inequality.

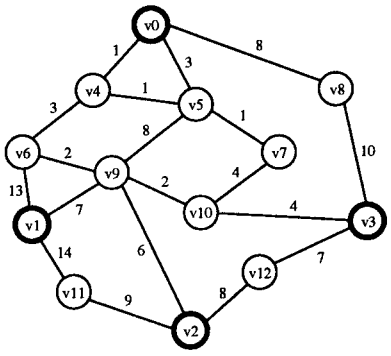

Figure 1: *An example instance of the SPG. The highlighted vertices constitutes W.*

Throughout this paper, let $n = |V|, m = |W|$ and $r = n - m$. If $m = 2$, SPG reduces to the shortest path problem, which can be solved by e.g. Dijkstra's algorithm [22] in time $O(|E| \log n)$. If $m = n$, SPG is the Minimum Spanning Tree problem (MSpT), which can be solved in $O(n^2)$ time by e.g. Prim's algorithm [1]. However, if $2 < m < n$, SPG is in general NP-complete[1] [19].

[1]Some special graph topologies do exist, for which SPG can still be solved in polynomial time [30].

3 Description of the Algorithm

In this section the developed algorithm is described in detail. First an overview of the algorithm is given in Section 3.1. Initially an attempt to reduce the size of a given problem is made by applying some graph reduction techniques described in Section 3.2. The main idea of the GA is the application of the Distance Network Heuristic for interpretation of the representation manipulated by the genetic operators. This is discussed in Sections 3.3 and 3.4. Other components of the algorithm is described in Sections 3.5, 3.6 and 3.7. Finally, the time complexity of the algorithm is discussed in Section 3.8.

3.1 Overview

The concept of genetic algorithms, introduced by John Holland [16], is based on natural evolution. In nature, the individuals constituting a population adapt to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, while the less fit individuals tend to die out. This *survival of-the-fittest* Darwinian principle is the basic idea behind the GA.

The algorithm maintains a *population of indiuiduals*, each of which corresponds to a specific solution to the optimization problem at hand. A measure of *fitness* defines the quality of an individual. Starting with a set of random individuals, a process of evolution is simulated. The main components of this process are *crossover*, which mimics propagation, and *mutation*, which mimics the random changes occurring in nature. After a number of *generations*, highly fit individuals will emerge corresponding to good solutions to the given optimization problem.

A *phenotype* is the physical appearance of an individual, while a *genotype* is the corresponding representation or genetic encoding of the individual. Crossover and mutation are performed in terms of genotypes, while fitness is defined in terms of phenotypes. For a given genotype, the corresponding phenotype is computed by a *decoder*. A good introduction to genetic algorithms is given in [12].

Fig. 2 shows a template for the GA considered here. Before the GA itself is executed, routine *graphReductions* tries to reduce the size of the given problem as described in Section 3.2. Then the initial current population P_C is constructed from randomly generated individuals by routine

```
graphReductions();
generate(P_C);
evaluate(P_C);
s = bestOf(P_C);
repeat until stopCriteria():
   P_N = ∅;
   repeat M/2 times:
      select p_1 ∈ P_C, p_2 ∈ P_C;
      {c_1, c_2} = crossover(p_1, p_2);
      P_N = P_N ∪ {c_1, c_2};
   end;
   evaluate(P_C ∪ P_N);
   P_C = reduce(P_C ∪ P_N);
   ∀ p ∈ P_C : possibly mutate(p);
   ∀ p ∈ P_C : possibly invert(p);
   evaluate(P_C);
   s = bestOf(P_C ∪ {s});
end;
optimize(s);
output s;
```

Figure 2: *Outline of the algorithm.*

generate. Routine *evaluate* described in Section 3.5 computes the fitness of each of the given individuals, while *bestOf* finds the individual with the highest fitness. One execution of the outer "repeat" loop corresponds to the simulation of one generation. Throughout the simulation the number of individuals $M = |PC|$ is kept constant. We keep track of the best individual s ever seen. Routine *stopCriteria* terminates the simulation when no improvement of the best or the average fitness has been observed for S consecutive generations, or when the algorithm has converged so that all individuals have the same fitness. Each generation is initiated by the formation of a set of offspring P_N of size M. The two mates p_1 and p_2 are selected from P_C independently of each other, and each mate is selected with a probability proportional to its fitness. The *crossover* routine described in Section 3.6 generates two offspring c_1 and c_2. Routine *reduce* returns the M fittest of the given individuals, thereby keeping the population size constant. With

a small probability p_{mut}, the *mutation* operator randomly changes each of the components, or *genes*, of its argument, as described in Section 3.7. The genetic operator *invert*(p) alters the genotype of an individual p without altering the corresponding phenotype. As described in [12], the purpose of this operator is to optimize the relative positions of the genes of p with respect to the crossover operator. The inversion operator will be described in Section 3.7. Routine *optimize(s)* performs simple local hill-climbing by executing a sequence of mutations on s, each of which improves the fitness of s. An exhaustive strategy is used so that when the routine has been executed, no single mutation exists, which can improve s further. The output of the algorithm is then the solution s.

3.2 Graph Reductions

Before the GA itself is executed an attempt to reduce the size of the given problem is performed using standard graph reduction techniques. Routine *graphReductions* of Fig. 2 performs four kinds of rather simple reductions all of which are described in [30, 31]. More elaborate reductions as well as proofs of the correctness of the reductions used here can be found in [9]. Let e_{vw} denote the edge between vertices v and w, and let $\mathrm{sp}(u, w) \subseteq E$ denote the shortest path between v and w. The four reductions used are:

a) Assume $\deg(v) = 1$ and $e_{vw} \in E$. If $v \in W$ any MStT must include e_{vw}. Hence, v and e_{vw} can be removed from G and w is added to W if it is not already there. If $v \in V \setminus W$, no MStT can include e_{vw} i.e. in this case v and e_{vw} can also be deleted.

b) If $v \in V \setminus W$, $\deg(v) = 2$ and $e_{uv}, e_{vw} \in E$, then v, e_{uv} and e_{vw} can be deleted from G and replaced by a new edge between u and w of equivalent cost. More specifically, if $e_{uw} \notin E$ then $E = E \cup \{e_{uw}\}$ and $c(e_{uw}) = c(e_{uv}) + c(e_{vw})$. If there is an edge from u to w already, i.e., $e_{uw} \in E$, then $c(e_{uw}) = min\{c(e_{uw}), c(e_{uv}) + c(e_{vw})\}$.

c) If $e_{vw} \in E$ and $\mathrm{c}(e_{vw}) > c(\mathrm{sp}(v, w))$ then no MStT can include e_{vw}, which therefore can be deleted.

d) Assume that $v \in W$ and denote the closest neighbour to v by $u \in V$, and the second-closest neighbour by $w \in V$. Since G is connected, u always exists. If w does not exist, assume $c(e_{vw}) = \infty$. Let z be a vertex

in $W \setminus \{v\}$ which is closest to u. If $c(e_{vu}) + c(\mathrm{sp}(u,z)) \leq c(e_{vw})$ then any MStT must include e_{vu}. Therefore, G can be contracted along this edge. Note that $u \in W \Rightarrow z = u \Rightarrow c(\mathrm{sp}(u,z)) = 0$ i.e., contraction can always be performed in this case.

To obtain the largest possible overall reduction of G, the above reductions are performed repeatedly as described below. Knowledge of the cost of a shortest path is required whenever a reduction of type c or d is performed. Shortest paths are also repeatedly needed by the GA as will become apparent in Section 3.4. Therefore, the distance graph $D(G)$ is computed initially using Floyd's algorithm [1] which requires time $O(n^3)$. Whenever one of the above reductions are performed, $D(G)$ has to be dynamically updated. When representing $D(G)$ as an adjacency matrix the update is trivial for reductions of type a or b: It simply consists of deleting the row and column corresponding to the deleted vertex. Reductions of type c leaves $D(G)$ unchanged. However, for reductions of type d the update is slightly more involved. Whenever a contraction is performed, $D(G)$ is updated using an $O(n^2)$ algorithm by Dionne and Florian [6].

In [30, 31] the following reduction is also suggested along with the reductions described above: If $\max\{c(\mathrm{sp}(v,u)), c(\mathrm{sp}(v,to))\} < c(e_{uw})$, $e_{uw} \in E$ and $v \in W$, then no MStT can include e_{uw}, which therefore can be deleted. However, in this case the required update of $D(G)$ has a worst case complexity of $O(n^3)$ using Dionne and Florian's algorithm [6]. I.e., the update could be as expensive as recomputing the entire distance graph, and for this reason this reduction is omitted.

When performing a sequence of reductions of the same type, the overall result depends on the chosen traversal of the graph, that is, the order in which reductions are tried out. Furthermore, reductions of distinct types are mutually dependent in the sense that performing all possible reductions of some type may allow new subsequent reductions of another type. It is not clear in which order reductions should be performed to obtain the overall best reduction of a given graph [31]. The arbitrarily chosen scheme for performing reductions in routine *graphReductions* is shown in Fig. 3. Routine *reductions(x)* performs a single traversal of all vertices (or edges in the case of type c reductions) of G in an unspecified order and carries out a reduction of type x whenever possible. Routine *graphReductions* terminates when no reduction of any type succeeded for a complete iteration, i.e., when no reduction can reduce G further.

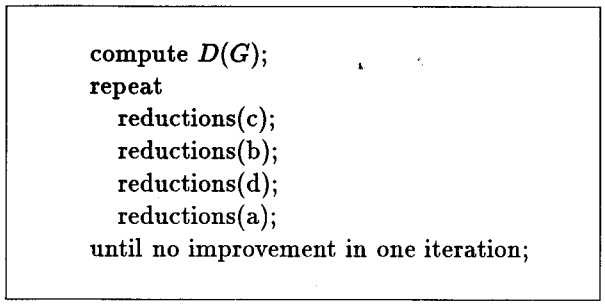

Figure 3: *Outline of routine graphReductions.*

To deduce the worst case time complexity of *graphReductions*, start by considering the maximum total time spend on reductions of type d. Due to the required update of $D(G)$ a single reduction requires time $O(n^2)$. Since vertices can be added to W when performing reductions of type a, $O(n)$ type d reductions are possible. Hence, the total time spend on type d reductions is $O(n^3)$. One execution of *reductions(x)* require at most time $O(n^2)$ when either $x \neq d$ or $x = d$ but no contraction is performed. Since each of the reductions a, b and d decreases the number of vertices by one, and since type c reductions are performed exhaustively in the sense that after executing *reductions(c)* no edge exist which can be removed by a type c reduction, at least one vertex must be removed in every second iteration of the "repeat" loop in *graphReductions*. Hence, there can be no more than $O(n)$ iterations. In total this gives routine *graphReductions* the time complexity $O(n^3)$.

Although it is not difficult to construct a graph for which none of the reductions performed by *graphReductions* applies, the routine has been observed to be very effective on many graphs, as will be seen in Section 4.4. When applied to the graph of Fig. 1, the result is the degenerate graph consisting of one vertex only, implying that a MStT has been found. In general, especially reductions of type d has been observed to be very powerful when m is relatively large, which coincides with the results reported in [31].

3.3 Distance Network Heuristic (DNH)

The key point in designing any GA is the design of a suitable genotype of an individual together with its interpretation, the decoder. The genetic encoding developed here is based on use of the Distance Network Heuristic

(DNH), a deterministic heuristic for the SPG, developed by Kou et al [20]. Therefore, before proceeding by presenting the genotype and the decoder, the DNH is described.

Given a graph $G = (V, E)$, a cost function c and a subset of vertices W in accordance with the definition of SPG in Section 2, the DNH computes an approximation T_{DNH} to the MStT for W in G in five steps:

1. Construct the subgraph G_1 of $D(G)$ induced by W.
2. Compute a MSpT T_1 of G_1.
3. Construct from T_1 the subgraph G_2 of G by substituting each edge in T_1 by the corresponding shortest path in G.
4. Compute a MSpT T_2 of G_2.
5. Compute T_{DNH} from T_2 by repeatedly deleting all vertices $v \in V \setminus W$ having $\deg(v) = 1$.

Any ties in Steps 2, 3 or 4 are broken arbitrarily. An example of how the DNH works is shown in Fig. 4, given as input the graph G of Fig. 1 and the subset $W = \{v_0, v_1, v_2, v_3\}$.

If $D(G)$ is not known, Step 1 of DNH requires time $O(mn^2)$ to compute shortest paths from each of the m vertices. Since G_1 is complete the MSpT in Step 2 is computed using Prim's algorithm requiring time $O(m^2)$. Each of the $m-1$ edges of T_1 may correspond to a path in G of up to $n-1$ edges. Hence, Step 3 requires time $O(mn)$ and Step 4 requires time $O(mn \log(nm))$ using Kruskal's algorithm [1]. The final step is done in time $O(n)$. Hence, if $D(G)$ is not known, Step 1 is the most expensive and gives the DNH a time complexity of $O(mn^2)$.

3.4 Genotype and Decoder

The basic idea of the genotype and the associated decoder is the following: The genotype specifies a set of selected Steiner vertices. The decoder computes the corresponding phenotype by executing the DNH using the union of the selected Steiner vertices and W as the set of vertices to be spanned. The selected Steiner vertices are specified by a bitstring in which each bit corresponds to a specific vertex. If the bit is set, the vertex is selected. For

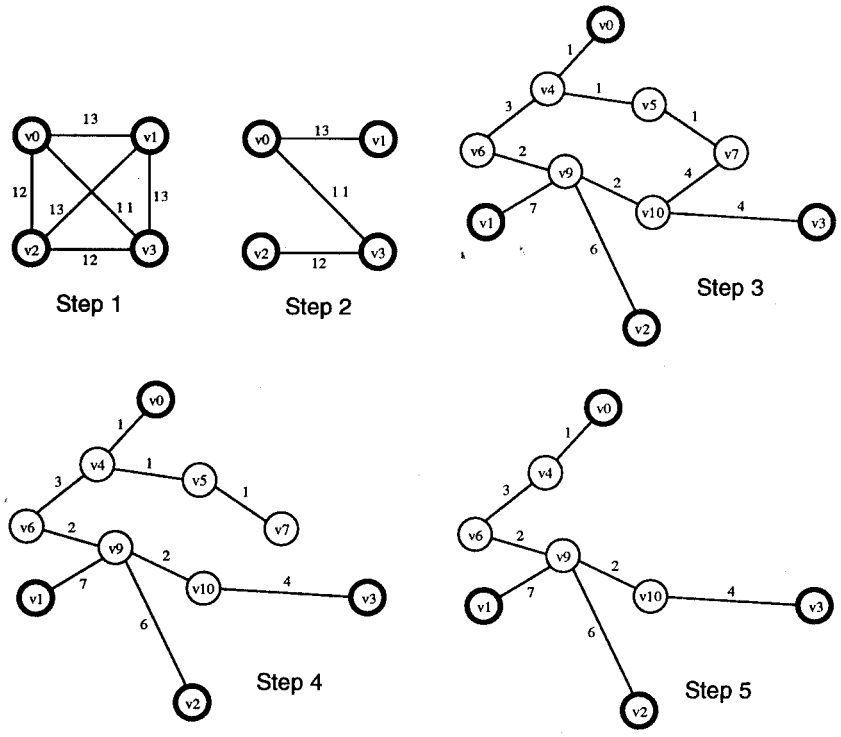

Figure 4: *The steps of DNH given the input graph from Fig. 1.*

reasons to be discussed in Section 3.7, we need the genotype to be independent of the ordering of the bits in the string. This is obtained by associating with each bit a tag which identifies the vertex specified by that bit.

Specifically, the genotype and the decoder can be described as follows. For a given instance of SPG, assume a fixed numbering $0, 1, \ldots, r-1$ of the vertices in $V \setminus W$. Let $\pi : \{0, 1, \ldots, r-1\} \mapsto \{0, 1, \ldots, r-1\}$ be a bijective mapping. A genotype is then a set of r tuples:

$$\{(\pi(0), i_{\pi(0)}), (\pi(1), i_{\pi(1)}), \ldots, (\pi(r-1), i_{\pi(r-1)})\}$$

where $i_k \in \{0, 1\}, k = 0, 1, \ldots, r-1$. The Steiner vertices $S \subseteq V \setminus W$ specified by the genotype is $S = \{v_k \in V \mid i_k = 1\}$. The Steiner tree in G corresponding to the genotype is the tree computed by DNH using the set $S \cup W$ as the vertices to be connected. In Step 5 of DNH every vertex $v \notin W$ of degree 1 is deleted. Note that the Steiner tree is independent of π.

In other words, the Steiner tree constituting the phenotype of an individual does not change if the tuples in its genotype are reordered.

Any set of values of the i_k's in a genotype correspond to a valid phenotype. However, Lawler [21] has shown that a MStT in $D(G)$ exists, which has at most $m-2$ Steiner vertices. This result relies on the fact that regardless of the edge cost function c, the edge costs in $D(G)$ always satisfy the triangle inequality. Hence, it is sufficient to consider only the subset of genotypes which satisfies $|S| \leq \min(m-2, r)$. To take advantage of this reduction of the search space, a routine filter has been defined, which given any genotype g enforces the satisfaction of $|S| \leq \min(m-2, r)$ by randomly selecting and clearing the necessary number of set bits.

When the initial random population has been generated, the filter is applied to each of the individuals. From then on, the search is limited to the restricted region by applying the filter to every new individual generated by one of the genetic operators.

It is important to note that the DNH is not chosen for use as decoder because it is a especially good heuristic in terms of result quality. In [31] the performance of DNH is compared to that of two other well-known polynomial time heuristics for the SPG: The Shortest Path Heuristic (SPH) by Takahashi and Matsuyama [27] and the Average Distance Heuristic (ADH) by Rayward-Smith and Clare [25]. With respect to result quality the DNH is clearly outperformed by both these heuristics. The reason to use DNH for decoding is first of all that it provides a way to interpret *any* set of selected vertices as a *valid* Steiner tree, and secondly, that it is relatively fast. The important advantage of considering valid Steiner trees only is that it eliminates the need for penalty terms in the cost measure, and thus avoids potential problems of assigning a suitable cost value to an invalid or incomplete solution.

3.5 Fitness Measure

Given a population $P = \{p_0, p_1, \ldots, p_{M-1}\}$ the routine evaluate of Fig. 2 computes the fitness of each individual as follows. Let $C(p)$ be the cost of individual p, i.e. the cost of the Steiner tree represented by p, and assume that P is sorted so that $C(p_0) \geq C(p_1) \geq \ldots \geq C(p_{M-1})$. The fitness F of p_i is then computed as

$$F(p_i) = \tfrac{2i}{M-1} \qquad i = 0, 1, \ldots, M-1.$$

This fitness computation scheme is called *ranking* and is discussed in [29]. Controlling the variance of the fitness values is one of the frequent problems of GA's [12]. Ranking assures that the variance is constant throughout the optimization process. The specific scheme chosen here constantly gives the best individual twice the probability of the median individual of being selected for crossover.

3.6 Crossover Operator

Given two parent genotypes α and β, the crossover operator generates two offspring, ϕ and ψ. The parent genotypes are not altered by the operator. An example of crossover is shown in Fig. 5. In this section, a superscript specifies which individual the marked property is a part of Crossover consists of three steps:

1. One of the parents, say β, is chosen at random, and a copy γ of β is made. γ is then reordered so that it becomes homologous to α, that is, $\pi^\gamma = \pi^\alpha$.

2. Both offspring are given the same ordering as their parents, i.e., $\pi^\phi = \pi^\psi = \pi^\alpha$. Standard 1-point crossover is then performed [12, 16]: A crossover-point x is selected at random in $\{0, 1, \ldots, r-2\}$. The selection of Steiner vertices in ϕ and ψ is then defined by

$$i^\phi_{\pi(k)} = \begin{cases} i^\alpha_{\pi(k)} & \text{if } k \leq x \\ i^\gamma_{\pi(k)} & \text{if } k > x \end{cases}$$

   and

$$i^\psi_{\pi(k)} = \begin{cases} i^\gamma_{\pi(k)} & \text{if } k \leq x \\ i^\alpha_{\pi(k)} & \text{if } k > x \end{cases}$$

   where $\pi = \pi^\alpha$.

3. Finally, both ϕ and ψ are subjected to the filter routine, if necessary.

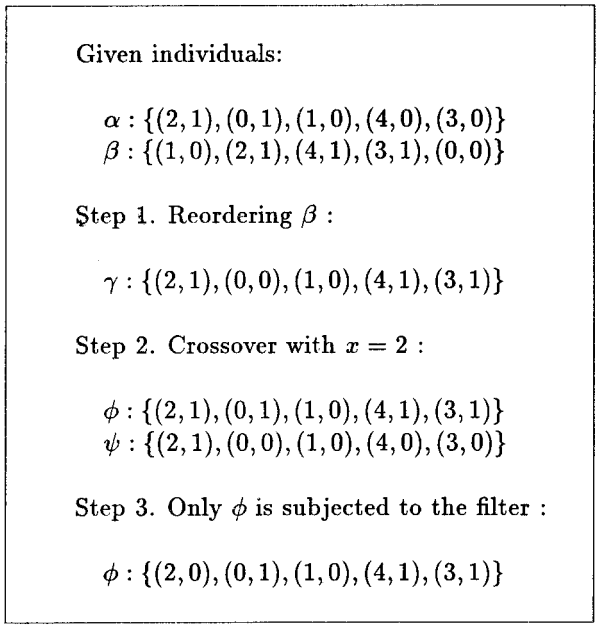

Figure 5: *Illustration of the crossover operator with* $m = r = 5$.

3.7 Mutation and Inversion Operators

The mutation operator is extremely simple. Given a genotype g, the operator inverts each of the r bits in g with a small given probability p_{mut}. This scheme is called pointwise mutation. If necessary, g is then passed through the filter routine.

For a given phenotype, several equivalent genotypes usually exist. Since crossover is performed in terms of genotypes, the fitness of produced offspring depends on which of the possible genotypes are used as codings of the given phenotypes. The purpose of inversion is to optimize the performance of the crossover operator by rearranging the components within a given genotype, as explained in detail in [12, 16].

With a given probability p_{inv}, the inversion operator reorders the tuples of a given genotype g by altering its ordering π. This does not change the phenotype corresponding to g. To obtain a uniform probability of movement of all tuples, we consider the genotype to form a ring. A part of the ring is then selected at random and reversed. More specifically, two points $x, y \in \{0, 1, \ldots, r-1\}, x \neq y$, are selected at random. The operator then defines

the new ordering π' of g as[2]

$$\pi'((x+i) \bmod r) = \begin{cases} \pi((y-i) \bmod r & \text{if } 0 \leq i \leq (y-x) \bmod r \\ \pi((x+i) \bmod r & \text{otherwise} \end{cases}$$

for all $i = 0, 1, \ldots, r-1$. The inversion operator is illustrated in Fig. 6.

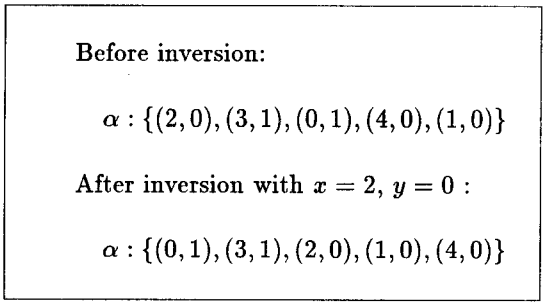

Figure 6: *Illustration of the inversion operator with* $r = 5$.

3.8 Time Complexity

The filter routine described in Section 3.4, the generation of each of the initial individuals, and the genetic operators crossover, mutate and invert each requires time $O(r) = O(n-m)$. The repeated decodings using DNH is the most expensive operation of the GA. Since knowledge of shortest paths is also required when performing some of the initial graph reductions, $D(G)$ is precomputed once and for all as mentioned in Section 3.2. This reduces the time of Step 1 of DNH to $O(1)$ and as a consequence, one decoding can now be performed in time $O(mn \log(nm))$. Fitness computation requires $O(M \log M)$ to sort the individuals. In total, the GA's setup time is $O(n^3)$, and each generation requires time $O(M[nm \log(nm) + logM])$.

Measurements reveals that the vast majority of the total runtime is spend on decodings. It also turns out that in practice the graph formed in Step 3 of the decoding process is almost always a tree, and as a consequence, Step 4 is rarely executed. Therefore, the true bottleneck of the algorithm is the MSpT computation performed in Step 2 of the decoding, which requires time $O(m^2)$.

[2]The definition of π' relies on the matematical definition of modulo, in which the remainder is always non-negative.

4 Experiments

This section describes the experimental method applied and the results obtained. Characteristics of the test examples used are given in Section 4.1. The deterministic heuristic SPH-I used for comparison is described in Section 4.2 and Section 4.3 describes the chosen method for performing the comparative experiments. The results are reported and discussed in Section 4.4. As mentioned in Section 1 an earlier GA for SPG has been developed by Kapsalis et al. and a comparison to that algorithm is presented in Section 4.5. Finally, Section 4.6 describes the typical behaviour of the GA during an optimization process.

4.1 Test Examples

The algorithm is tested on all 78 SPG instances from the OR-Library [4]. According to their size, these graphs are divided into four classes denoted by B, C, D and E. All graphs are generated at random subject only to the connectivity constraint, that is, the topology is random and the vertices to be spanned are selected at random. Every edge cost is a random integer in the interval [1, 10]. In class B each graph has n equal to 50, 75 or 100. The value of m is either $n/6$, $n/4$ or $n/2$ and the average vertex degree is either 2.5 or 4. Since all combinations exists, class B consists of 18 graphs. Classes C, D and E consists of graphs with n equal to 500, 1,000 and 2,500 respectively. m equals 5, 10, $n/6$, $n/4$ or $n/2$ and the average vertex degree is 2.5, 4, 10 or 50. Thus, each of the classes C, D and E consists of 20 graphs.

One of the main advantages of using this test-suite is that it facilitates comparison with the global optimal solutions. The global optima were first computed by J. E. Beasley who developed a branch-and-cut algorithm which was executed on a Cray X-MP/48 supercomputer [3].

For a given graph, the size of the search space $S(n, m)$ to be explored by the GA is

$$S(n,m) = \sum_{i=0}^{k} \left(\begin{array}{c} n-m \\ i \end{array} \right)$$

where $k = \min(m-2, n-m)$, since this is the number of possible distinct choices of the Steiner vertices. Some of the problem instances considered represents extremely large search spaces, as will be seen in Section 4.4.4.

However, as mentioned in Section 3.7, the corresponding phenotype spaces are smaller.

4.2 Iterated Shortest Path Heuristic (SPH-I)

As mentioned in Section 3.4 a comparative study of the deterministic heuristics SPH, DNH and ADH has been made by Winter and Smith [31]. Several variants of these heuristics, especially a number of repetitive variants of SPH, are also considered in the study. The ADH is in general considered to be one of the best deterministic heuristics, which is also confirmed by the investigation in [31]. However, the results also reveals that some of the repetitive variants of SPH consistently outperform ADH with respect to result quality. Furthermore, by applying initial graph reductions the runtime of the repetitive SPH variants can be made comparable to that of the other heuristics. One of the specific conclusions in [31] is that on the largest random graphs considered, the repetitive SPH variant denoted SPH-ZZ outperforms all other heuristics. Therefore, this heuristic has been chosen for comparison with the GA.

Fig. 7 outlines our implementation of SPH-ZZ, denoted by SPH-I. It starts by computing $D(G)$ and performing graph reductions as described in Section 3.2. For given vertices x and y, $G_{xy} = (V_{xy}, E_{xy})$ denotes the subgraph of G corresponding to the shortest path between x and y. In each iteration of the outer loop a tree T is build which spans all vertices in W. T is initialized with a shortest path between two of the vertices to be spanned, and T is then extended by repeated addition of a shortest path to a closest, not yet connected vertex. This scheme is tried for all possible initializations of T, and the algorithm outputs the best such tree obtained.

As described in Section 3.2 routine *graphReductions* requires time $O(n^3)$. The construction of each candidate solution T takes time $O(m^2 n)$ since the "while" loop is iterated $O(m)$ times and it takes time $O(mn)$ to find each z vertex and extend T with a shortest path to it. This is due to the fact that all distances have been precomputed. Since $O(m^2)$ candidate solution trees T are computed, the total runtime of SPH-I becomes $O(n^3 + m^4 n)$.

4.3 Experimental Method

The GA is evaluated by four kinds of comparisons:

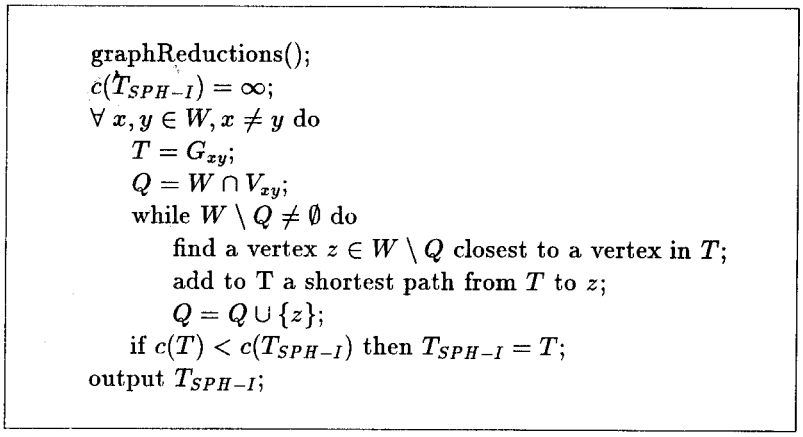

Figure 7: *Outline of SPH-I.*

- The solution quality obtained is compared to the global optimum.
- The absolute runtime is compared to that of two distinct branch- and-cut algorithms by Lucena and Beasley [23] and Chopra, Gorres and Rao [5].
- Solution quality and absolute runtime is compared to that of SPH-I.
- Comparison with the GA by Kapsalis et al [18].

The branch-and-cut algorithms are guaranteed to find the global optimum. However, runtime may be unacceptable for some problem instances or may even prevent some problems from being solved. It is therefore of interest to investigate if a near-optimal solution can be found for all problems by using a moderate amount of time.

The GA has been executed 10 times for each example in the B, C and D classes. Solution quality is then evaluated in terms of best, average and worst results produced. However, due to runtime requirements the GA was only executed once for each of the examples in class E. The parameter settings are $M = 40$, $S = 50$, $p_{mut} = 0.005$ and $p_{inv} = 0.1$. These values are used for all executions, i.e., no problem specific tuning has been made. As mentioned in Section 1 fixed parameter values are of major importance from a practical point of view.

The GA as well as SPH-I are implemented in the C programming language. For both algorithms, examples from classes B, C and D are executed on a Sun Sparc IPX workstation having 32 Mb RAM. These examples require at most 10 Mb of memory. For the class E examples, the memory requirement is about 58 Mb. Therefore, for these examples the GA as well as SPH-I are executed on a DEC Mips 5000-240 workstation having 128 Mb RAM.

The branch-and-cut algorithm by Lucena and Beasley [23] is a further development of the algorithm presented in [3], but instead of using a Cray, it is now executed on a Sun Sparc 2 workstation. This machine is roughly as fast as the Sun Sparc IPX, but probably somewhat slower than the DEC Mips 5000-240. Chopra et al's algorithm [5] is executed on a VAX 8700 which is at most as fast as the other machines. When comparing absolute runtimes in Section 4.4 the reader should keep these differences regarding the used hardware in mind. However, the runtime variations caused by the different machines are insignificant compared to the variations caused by different problem instances when considering a specific algorithm.

4.4 Results

In the following sections the detailed experimental results for all four problem classes are commented. The tables referenced can be found in Appendix A. A summary and conclusion of the results are given in Section 4.4.5.

4.4.1 The B Graphs

Table 2 lists the characteristics of the problems in class B before and after the graph reductions of Section 3.2 are performed. The reductions significantly impacts all graphs. Especially, graphs B-1, B-3 and B-9 are reduced to the degenerate graph consisting of a single vertex only, which means that the optimal solution is found solely by performing graph reductions.

Table 3 compares the solution quality obtained by the GA to the globally optimal solutions as well as to the solutions found by SPH-I. C_{opt} is the global optimum and C_{sph} is the solution found by SPH-I. C_{best}, C_{avg} and C_{worst} is the best, average and worst result produced by the GA in the 10 runs, while C_σ denotes the standard deviation of the 10 cost values. $\Delta C_{sph} = 100(C_{sph} \backslash C_{opt} - 1)$ is the relative error in percent of the solution found by SPH-I compared to the optimum solution. Similarly,

$\Delta C_{avg} = 100(C_{avg} \backslash C_{opt} - 1)$ denotes the average error of the solutions found by the GA, and $\Delta C_{worst} = 100(C_{worst} \backslash C_{opt} - 1)$ is the worst error produced by the GA. Finally, N_{ga} denotes the number of the 10 runs which did not find the global optimum. This notation is also used in the following sections.

As can be seen, the GA finds the global optimum for all examples in every execution. SPH-I performs similarly for all graphs except B-13, for which it has a 1.82 % overhead.

Table 4 compares the runtime of the GA with that of SPH-I and the branch-and-cut algorithm by Lucena and Beasley [23]. T_{bc2} denotes the runtime of the latter algorithm and T_{sph} is the time of SPH-I. The average time spent by the GA is denoted T_{avg} while T_σ denotes the standard deviation of the time for the 10 runs. Chopra et al [5] gives no computational results for these graphs. It can be seen that all runtimes are very small and within the accuracy of these measurements it is difficult to draw any conclusions regarding differences in speed for the different algorithms.

The fact that all three algorithms finds optimal solutions (except for SPH-I on B-13) in a very short time suggests that these examples are simply too small to facilitate any distinction of performance of the algorithms. For several of the graphs the search spaces after graph reductions are indeed very small and the largest search space is that of B-17 with less than 10^9 points, which is not that much for a combinatorial optimization problem.

4.4.2 The C Graphs

From Table 5 it can be seen that the graph reductions are also very effective on most graphs in the C class. Note especially graph C-5 which after reductions has a search space size of only approximately 106 points. However, as the average vertex degree increases, the effect of reductions of types a and b (see Section 3.2) decreases significantly. When m is small, the effect of reductions of type d is also very limited, as can be seen by the results for C-11, C-12, C-16 and C-17. The obtained reduction in search space sizes for these problems are negligible. The effect of reductions of type c increases with the number of edges. For C-16 through C-20 about two thirds of all edges are eliminated by graph reductions, mainly of type c. However, since the GA operates in terms of shortest paths, minimum spanning trees, etc., the number of edges are not that important for the performance of the algorithm.

Table 6 shows that the GA finds the global optimum at least once for all examples and every time for 12 of the graphs, while SPH-I finds the optimum

for 10 of the graphs. When neither the average GA run nor the SPH-I finds the global optimum, ΔC_{avg} is often an order of magnitude better than ΔC_{sph}. This is the case for C-3, C-4, C-9, C-14, C-18 and C-l9. For C-18 and C-l9 the solutions produced by SPH-I are very poor with errors in the 6 - 7 % range. The results for C-16 are in direct contrast to all other results. While the SPH-I finds optimum, the GA encounters severe problems. In 7 of 10 runs it misses the global optimum value of 11 and outputs a tree of cost 12. This corresponds to a huge relative error ΔC_{worst} of 9.09 %.

In Table 7 and subsequent tables T_{bc1} denotes the runtime of the branch-and-cut algorithm by Chopra et al [5]. Depending on the problem, the runtime for both branch-and-cut algorithms varies extremely. Chopra's algorithm spans from 10 secs. for C-16 to more than $45,000$ secs. for C-18, while Lucena's algorithm varies from 5 secs. for C-5 to more than $20,000$ secs. for C-18. As a consequence, the branch-and-cut algorithms are significantly faster than both the GA and SPH-I for some graphs and significantly slower for others. The runtimes of the GA and the SPH-I are similar for most graphs, although the GA is significantly faster for graphs C-15, C-19 and C-20. The time variation T_σ of the GA is relatively small.

4.4.3 The D Graphs

The effect of graph reductions on the class D graphs shows a pattern similar to that observed for the C graphs although now the pattern is even clearer. Most graphs are reduced significantly, note especially D-5. The effect of reductions decreases as m decreases and as the average vertex degree increases.

On the class D graphs SPH-I finds optimum for 7 of the graphs, while the GA finds the optimum at least once for 17 graphs and every time for 13 graphs. SPH-I has relative errors exceeding 2 % for 5 graphs while that only happens for the GA on graph D-18. For all graphs we have $C_{worst} \leq C_{sph}$ and $C_{worst} < C_{sph}$ holds for 13 graphs.

On this class of problems the runtimes for both branch-and-cut algorithms varies by three orders of magnitude and are as high as in the $200 - 300,000$ secs. range corresponding to 2-3 days of computation. The runtime of SPH-I now also varies significantly. For practical reasons it became necessary to introduce a CPU-time limit of $50,000$ secs. for this algorithm on graphs from classes D and E. When SPH-I did not complete its computation within this limit, it was terminated and the best solution found so far was used. This happened for graphs D-l9 and D-20. For these graphs the total time needed

by SPH-I is estimated to be $95,000$ secs. and $679,000$ secs., respectively. These estimates can be considered to be quite accurate since they are based on measurements of the CPU-time spend for each pair of vertices $x, y \in V$, cf. Fig. 7, which is then scaled with the relative number of vertex pairs not yet considered at the time the CPU-limit is exceeded. The average runtime of the GA varies from 504 secs. for D-5 to $3,441$ secs. for D-19, i.e., by a factor of 7. This variation is small compared to the variation of the other algorithms considered. For graphs D-8, D-9, D-10, D-13, D-14, D-15, D-18, D-19 and D-20 the GA is on average an order of magnitude faster than SPH-I while for the remaining graphs the runtimes of these algorithms are comparable.

4.4.4 The E Graphs

For the graphs from class E the effect of graph reductions follows a pattern which coincides perfectly with the patterns observed for classes C and D. Even after reductions the search space sizes for the class E graphs are enormous. Using the bound

$$S(n,m) > \left(\begin{array}{c} n-m \\ k \end{array} \right) \geq \left(\frac{n-m-k+1}{k} \right)^k$$

where $k = \min(m-2, n-m)$ reveals that a number of graphs in this class has search spaces exceeding 10^{100} points. Especially, the search space for E-13 exceeds 10^{231} points and for E-18 it exceeds 10^{242} points. These bounds are computed after graph reductions have been performed.

Table 12 lists the solution qualities obtained by the GA and the SPH-I together with the runtimes of all algorithms considered. Due to the extensive runtimes required for the graphs in this class, the GA was executed only once for each example. Cga denotes the cost obtained by the GA, ΔC_{ga} is the relative error of the solution found by the GA and the time spend by algorithm is denoted by T_{ga}. Hence, C_{ga} and T_{ga} can be considered estimates of C_{avg} and T_{avg}, respectively.

It should be noted that the listed value of C_{opt} for E-18 may not be the global optimum, but according to the information in OR-Library it is the best known solution as found by Beasley's algorithm [3]. The optimum for this graph was not found within a CPU-limit of 21,600 secs. on the Cray X-MP/48. Chopra et al [5] also encountered problems with E-18. No runtime

is listed for this graph since the algorithm did not terminate within a CPU-limit of 10 days on the VAX 8700 [5]. Lucena and Beasley [23] does not report any results for graphs E-6 through E-20, and a reason is not given. However, considering the progression of runtime for the graphs in classes C and D, it is reasonable to assume that the algorithm is unable to solve some of these problems in a reasonable amount of time.

SPH-I exceeds the CPU-time limit of $50,000$ secs. for graphs E-3, E-8, E-9, E-10, E-13, E-14, E-15, E-18, E-l9 and E-20. The estimated total time required by SPH-I for these graphs varies from $81,000$ secs. for E-3 to 4.3×10^7 secs., or more than 16 months, for E-20. Compared to the branch-and-cut algorithms and SPH-I the runtimes of the GA are very moderate for all graphs with a maximum runtime of $29,105$ secs. for E-18. For most of the graphs for which SPH-I terminates within the CPU-time limit the runtimes of the GA and SPH-I are very similar. Regarding solution quality, SPH-I finds the global optimum for 4 of the graphs and has a worst relative error ratio exceeding 9 % for E-18. The GA finds optimum for 11 graphs and has a worst relative error ratio less than 2 %.

4.4.5 Summary of Results

This section summarizes the experimental results with respect to solution quality and runtime. When comparing the solution quality obtained by the GA to that obtained by SPH-I for all graphs in classes B, C and D the following can be observed: Of a total of 58 graphs, SPH-I finds the global optimal solution for 34 graphs, while the GA finds optimum 10 times out of 10 for 43 graphs and at least one time of 10 for 55 graphs. For the class E examples, SPH-I finds optimum for 4 of the 20 graphs, while the GA finds the optimum for 11 of these graphs.$\Delta C_{worst} \leq \Delta C_{sph}$ holds for all but one graph in classes B, C and D, and in class E we have $\Delta C_{ga} \leq \Delta C_{sph}$ for all graphs. In other words, with a single exception even the worst results generated by the GA are equal to or better than the result generated by SPH-I. Furthermore, for the graphs where both SPH-I and the average execution of the GA fails to find the global optimum, the expected relative error ratio ΔC_{avg} of the GA is often an order of magnitude better than the error ratio ΔC_{sph} of SPH-I.

Algorithm	Error Ratio		
	$= 0\%$	< 0.5 %	< 1.0 %
SPH-I	48.7	66.7	70.5
GA	77.1	86.7	92.6

Table 1: *Summary of solution qualities obtained by the GA and SPH-I.*

Table 1 summarizes the solution qualities obtained by the GA and the SPH-I. These figures are based on the results of all 600 executions of the GA and all 78 executions of SPH-I performed in total. For each algorithm Table 1 gives the accumulated percentage of runs which gave a result within the stated relative error from optimum. E.g., 66.7 % of all executions of the SPH-I gave a result which was less than 0.5 % from the optimum solution. When computing the values listed for the GA the results for the class E examples have been weighted by a factor of 10 to compensate for the imbalance in the number of executions for each graph.

The results regarding runtimes can be summarized in three main points:

- The GA is capable of finding a high-quality solution for all graphs considered in a moderate amount of time. This is not the case for any of the two branch-and-cut algorithms or for SPH-I.
- In most cases the runtime of the GA is very similar to that of SPH-I. In a few cases the GA is significantly faster than SPH-I, while the opposite is never the case.
- The variation of the runtime of the GA i8 very small compared to the variation observed for the branch-and-cut algorithms as well as SPH-I. As a consequence, the branch-and-cut algorithms are significantly faster than both the GA and SPH-I for some examples, while they are significantly slower on other examples.

AB problem size increases through the classes B, C, D and E the above observations become increasingly pronounced. If only class B graphs are considered, it is difficult to make any distinctions regarding performance of the algorithms. These examples appears to be too simple.

4.5 Comparison with Kapsalis Algorithm

In this Section the GA by Kapsalis, Rayward-Smith and Smith [18] is denoted GA-KRSS. As mentioned in Section 1 GA-KRSS differs from the GA

presented here in a number of ways. Among other things, neither an inversion operator nor a hill-climber is applied in GA-KRSS. However, the most significant differences concerns the decoder and the cost computation. In GA-KRSS a genotype is a bitstring of length n in which the i'th bit indicates if the i'th vertex is part of the phenotype tree. To assure that every tree spans W each genotype is xor'ed with the fixed string specifying W. Hence, the encoding is very similar to our encoding. However, the interpretation of a genotype is very different. Assume a genotype specifies the vertex set Z, $W \subseteq Z \subseteq V$. The corresponding graph is then computed as the subgraph G_Z of G induced by Z. In general G_Z is not connected. Assume it consists of $k \geq 1$ components. The cost of a solution is defined as the sum of the cost of a minimum spanning tree for each component plus a penalty term which grows linearly with k.

Computational results are given only for the class B graphs from the OR-Library. The solution quality obtained for each graph is reported as the best result of five runs. For each graph some parameter setting of GA-KRSS has been found with which the global optimum is found in five runs. However, the parameter setting varies with the problems given. When fixing the parameter setting for all graphs, GA-KRSS finds the global optimum in approximately 70 % of all runs and the worst result generated is 7.3 % above the global optimum.

All experiments with GA-KRSS are run on a Apple Mac IIfx. No total runtimes are given. Instead the time spend until the best solution found appears first time, referred to as Last Improvement Time (LIT), is measured. It is not clear exactly which stop criteria is used, i.e., how long the algorithm takes to terminate beyond LIT. For many of the graphs, the average LIT is in the range from 200 to $2,000$ secs. There is a time limit of $4,000$ secs. for a complete execution.

GA-KRSS is clearly inferior to each of the other algorithms considered in this paper, both with respect to solution quality as well as runtime. We believe that the main reason for the performance gap between GA-KRSS and the GA presented here is the different decoding strategy and consequently, the different cost evaluation strategy.

4.6 Typical Behavior

The progress of the typical optimization process is illustrated by Figures 9, 10 and 11, which stems from a sample execution of the GA with graph D-15

as input. It should be emphasized that although the graphs stems from a specific single run, the picture they give is very typical.

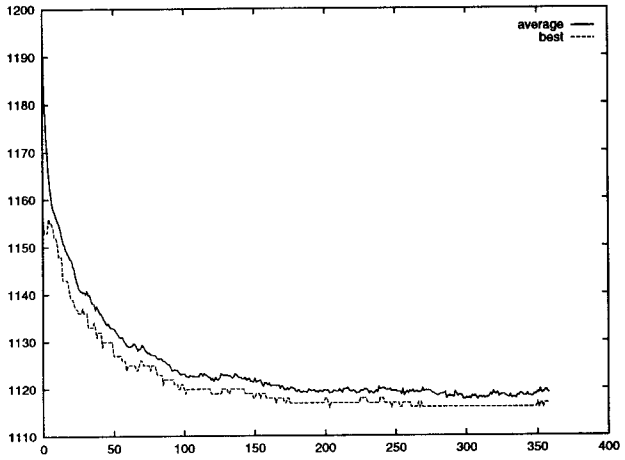

Figure 8: Cost of average and best individual as functions of generation number.

For each generation, the top graph of Fig. 9 indicates the average cost of the individuals in the population at that time, while the bottom graph indicates the cost of the current best individual. Initially, the average cost is $1,197$ and the best is $1,156$. The global optimum of $1,116$ is obtained first time in generation 203, and the algorithm terminates after 358 generations. Note that improvement is very rapid during the first part of the process. Then it levels out and further improvement is obtained only slowly. As mentioned in Section 3.1 the best as well as the average cost are parts of the stop criteria. If only the cost of the best solution were considered, the process would have terminated after generation 253, corresponding to a 29 % reduction of the runtime. However, the used stop criteria reflects a priority of solution quality as being more important than runtime.

Fig. 10 shows for each generation the standard deviation of cost in the population. From a value of 19.2 in generation 0, the standard deviation decreases within 10 generations to about 2.0 and then stays at that level throughout the optimization process.

As described in Section 3.1 each generation is initiated by the generation of M offspring individuals. From the total of $2M$ individuals the best M individuals are then kept as members of the new population while the rest

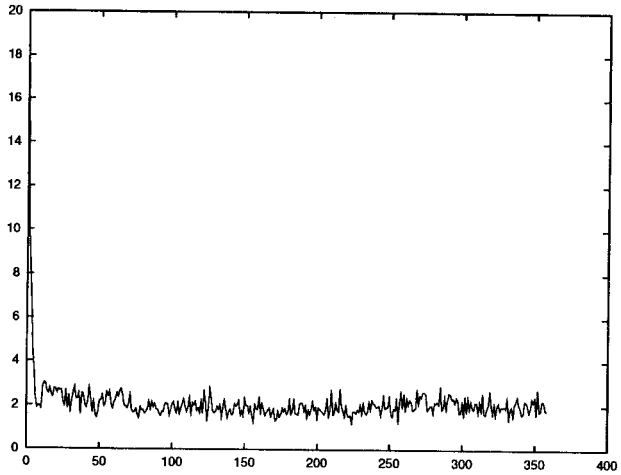

Figure 9: Standard deviation of cost as a function of generation number.

are discarded. Fig. 11 shows for each generation the percentage

of individuals in the newly created population which has just emerged as results of crossover. The percentage of newly generated individuals is very stable around 50. The important thing to note is that the fraction of new individuals do not decrease with time but is constant also into the late phase of the process. In other words, throughout the process half the individuals generated by the crossover operator are better than some other individual already in the population. This confirms the role of crossover as the most important of the genetic operators.

5 Future Work

The work presented here can be continued in at least three sections:

1. Performance improvement: As discussed in Section 3 the main idea of the GA presented is the application of the DNH for interpretation of bitstrings. In contrast, the genetic operators for crossover, mutation and inversion are all standard. They are characterized by being very simple and blind in the sense that they do not utilize knowledge of the application domain in any way. The same is true for the hill-climber. One frequently used way of improving the performance of a GA is to apply more advanced genetic operators and/or operators

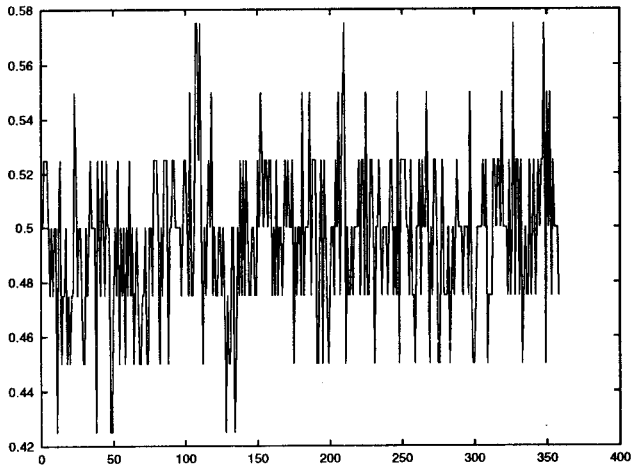

Figure 10: Percentage of new individuals in the population as a function of generation number.

exploiting application specific knowledge [12]. It is therefore likely that the performance of the GA presented here can be further improved by applying such techniques.

2. Other graph types: An obvious weakness of the test-suite used in this work is that all graphs are sparse and randomly generated. It remains to be seen how the GA performs on e.g. dense graphs, rectilinear graphs, non-random graphs arising in real-world applications, etc.

3. Contributions to performance: To obtain a detailed understanding of the reasons for the success of the algorithm it would be interesting to investigate how the various components of the algorithm contribute to the overall performance. What is the individual effect on solution quality and runtime caused by e.g. the decoding strategy, the inversion operator, the search space reduction or the initial graph reductions ?

6 Conclusion

In this paper a new Genetic Algorithm (GA) for the Steiner Problem in a Graph (SPG) has been presented. The main idea behind the algorithm is the application of the Distance Network Heuristic for interpretation of bitstrings

specifying selected Steiner vertices. This scheme ensures that every bitstring corresponds to a valid solution and eliminates the need for penalty terms in the cost measure, thereby avoiding potential problems of assigning a suitable cost value to an incomplete or invalid solution.

The performance of the algorithm has been tested on random graphs with up to $2,500$ vertices and $62,500$ edges. The experimental results shows that in more than 92 % of all executions the GA finds a solution which is within 1 % from the global optimum. This performance compares favorably with one of the very best deterministic heuristics for SPG as well as with an earlier GA by Kapsalis et al. Performance is also compared to that of branch-and-cut algorithms by Lucena and Beasley and by Chopra et al. While the runtimes of these algorithms varies extremely and prevents the solution of some of the problem instances considered, the GA is capable of generating a near-optimal solution for all problems within a moderate amount of time.

We therefore conclude the following: In cases where a globally optimal solution is absolutely required, the size of the given problem is not too big and runtime is not important, one of the branch-and-cut algorithms are preferable. On the other hand, if a near-optimal solution is sufficient, or the problem is very large or a moderate runtime limit is needed, the GA presented here is the best choice of the possibilities considered.

Acknowledgement

The author wishes to thank Pinaki Mazumder, University of Michigan, who supervised this work in its initial phase when the author was at University of Michigan. Also thanks to Jens Clausen and Pawel Winter, Copenhagen University, for pointing out possible improvements of an earlier version of the algorithm and for suggesting a suitable strategy for performance evaluation. Finally thanks to Peter Møller-Nielsen, Ole Caprani and Holger Orup, Aarhus University, for several useful discussions and suggestions concerning this work.

A Computational Results

Graph	Problem size			Reduced size		
	n	m	$\mid E \mid$	n	m	$\mid E \mid$
B-1	50	9	63	1	1	0
B-2	50	13	63	7	4	12
B-3	50	25	63	1	1	0
B-4	50	9	100	34	7	72
B-5	50	13	100	35	10	76
B-6	50	25	100	25	10	60
B-7	75	13	94	16	6	26
B-8	75	19	94	16	7	25
B-9	75	38	94	1	1	0
B-10	75	13	150	50	10	115
B-11	75	19	150	47	8	108
B-12	75	38	150	31	11	74
B-13	100	17	125	28	9	47
B-14	100	25	125	22	8	42
B-15	100	50	125	16	9	28
B-16	100	17	200	63	9	148
B-17	100	25	200	51	12	113
B-18	100	50	200	35	12	84

Table 2: *Characteristics of the class B graphs before and after reductions.*

Graph	C_{opt}	C_{sph}	ΔC_{sph}	C_{best}	C_{avg}	C_{worst}	C_{σ}	ΔC_{avg}	ΔC_{worst}	N_{ga}
B-1	82	82	0	82	82	82	0	0	0	0
B-2	83	83	0	83	83	83	0	0	0	0
B-3	138	138	0	138	138	138	0	0	0	0
B-4	59	59	0	59	59	59	0	0	0	0
B-5	61	61	0	61	61	61	0	0	0	0
B-6	122	122	0	122	122	122	0	0	0	0
B-7	111	111	0	111	111	111	0	0	0	0
B-8	104	104	0	104	104	104	0	0	0	0
B-9	220	220	0	220	220	220	0	0	0	0
B-10	86	86	0	86	86	86	0	0	0	0
B-11	88	88	0	88	88	88	0	0	0	0
B-12	174	174	0	174	174	174	0	0	0	0
B-13	165	168	1.82	165	165	165	0	0	0	0
B-14	235	235	0	235	235	235	0	0	0	0
B-15	318	318	0	318	318	318	0	0	0	0
B-16	127	127	0	127	127	127	0	0	0	0
B-17	131	131	0	131	131	131	0	0	0	0
B-18	218	218	0	218	218	218	0	0	0	0

Table 3: *Comparison of solution quality for the graphs in class B.*

Graph	T_{bc2}	T_{sph}	T_{avg}	T_{σ}
B-1	0.1	0.1	0.1	0.0
B-2	0.1	0.1	0.2	0.0
B-3	0.1	0.1	0.1	0.0
B-4	0.6	0.1	1.2	0.6
B-5	1.9	0.1	0.7	0.2
B-6	0.6	0.1	0.2	0.1
B-7	0.2	0.2	0.5	0.1
B-8	0.1	0.2	0.5	0.1
B-9	0.1	0.2	0.2	0.0
B-10	3.1	0.3	1.7	0.5
B-11	1.4	0.3	1.4	0.6
B-12	0.6	0.3	0.6	0.1
B-13	0.7	0.4	1.4	0.4
B-14	1.2	0.5	0.9	0.3
B-15	0.3	0.5	0.8	0.1
B-16	18.4	0.6	4.4	1.9
B-17	3.3	0.6	2.3	0.6
B-18	1.0	0.6	1.5	0.30

Table 4: *Comparison of CPU-time in seconds for the graphs in class B.*

Graph	Problem size			Reduced size		
	n	m	$\mid E \mid$	n	m	$\mid E \mid$
C-1	500	5	625	145	5	263
C-2	500	10	625	130	8	239
C-3	500	83	625	120	35	232
C-4	500	125	625	109	38	221
C-5	500	250	625	37	17	91
C-6	500	5	1,000	369	5	847
C-7	500	10	1,000	382	9	869
C-8	500	83	1,000	336	54	818
C-9	500	125	1,000	349	78	832
C-10	500	250	1,000	213	76	624
C-11	500	5	2,500	499	5	2,184
C-12	500	10	2,500	498	9	2,236
C-13	500	83	2,500	463	65	2,108
C-14	500	125	2,500	427	81	1,961
C-15	500	250	2,500	299	92	1,471
C-16	500	5	12,500	500	5	4,740
C-17	500	10	12,500	499	9	4,698
C-18	500	83	12,500	486	70	4,668
C-19	500	125	12,500	473	98	4,490
C-20	500	250	12,500	386	143	3,850

Table 5: *Characteristics of the class C graphs before and after reductions.*

Graph	C_{opt}	C_{sph}	ΔC_{sph}	C_{best}	C_{avg}	C_{worst}	C_{σ}	ΔC_{avg}	ΔC_{worst}	N_{ga}
C-1	85	85	0	85	85	85	0	0	0	0
C-2	144	144	0	144	144	144	0	0	0	0
C-3	754	754	0.40	754	754.2	755	0.4	0.03	0.13	2
C-4	1,079	1,081	0.19	1,079	1,079.1	1,080	0.3	0.01	0.09	1
C-5	1,579	1,579	0	1,579	1,579	1,579	0	0	0	0
C-6	55	55	0	55	55	55	0	0	0	0
C-7	102	102	0	102	102	102	0	0	0	0
C-8	509	512	0.59	509	509	509	0	0	0	0
C-9	707	714	0.99	707	707.4	708	0.5	0.06	0.14	4
C-10	1,093	1,098	0.46	1,093	1,093	1,093	0	0	0	0
C-11	32	32	0	32	32	32	0	0	0	0
C-12	46	46	0	46	46	46	0	0	0	0
C-13	258	263	1.94	258	259.7	260	0.6	0.66	0.78	9
C-14	323	327	1.24	323	323.4	324	0.5	0.12	0.31	4
C-15	556	558	0.36	556	556	556	0	0	0	0
C-16	11	11	0	11	11.7	12	0.5	6.36	9.09	7
C-17	18	18	0	18	18	18	0	0	0	0
C-18	113	121	7.08	113	114.3	115	0.8	1.15	1.77	8
C-19	146	155	6.16	146	147	148	0.4	0.68	1.37	9
C-20	267	267	0	267	267	267	0	0	0	0

Table 6: *Comparison of solution quality for the graphs in class C.*

Graph	T_{bc1}	T_{bc2}	T_{sph}	T_{avg}	T_{σ}
C-1	27	25	61	79	6
C-2	812	45	61	79	3
C-3	543	25	72	104	19
C-4	510	23	75	83	10
C-5	474	5	61	63	0
C-6	49	561	83	130	11
C-7	83	522	86	153	24
C-8	674	1,106	260	263	39
C-9	1,866	5,813	966	425	93
C-10	246	32	544	181	49
C-11	333	2,769	119	187	20
C-12	120	1,175	119	224	19
C-13	9,170	9,895	646	544	91
C-14	212	1,150	1,316	547	130
C-15	211	913	1,544	262	56
C-16	10	877	119	180	22
C-17	98	14,557	119	203	26
C-18	45,848	20,276	873	563	102
C-19	117	1,689	3,050	601	136
C-20	15	225	11,374	334	57

Table 7: *Comparison of CPU-time in seconds for the graphs in class C.*

Graph	Problem size			Reduced size		
	n	m	$\mid E \mid$	n	m	$\mid E \mid$
D-1	1,000	5	1,250	274	5	510
D-2	1,000	10	1,250	285	10	523
D-3	1,000	167	1,250	224	67	441
D-4	1,000	250	1,250	159	66	339
D-5	1,000	500	1,250	97	48	246
D-6	1,000	5	2,000	761	5	1,741
D-7	1,000	10	2,000	754	10	1,735
D-8	1,000	167	2,000	731	124	1,708
D-9	1,000	250	2,000	654	155	1,613
D-10	1,000	500	2,000	418	146	1,317
D-11	1,000	5	5,000	993	5	4,674
D-12	1,000	10	5,000	1,000	10	4,671
D-13	1,000	167	5,000	922	122	4,433
D-14	1,000	250	5,000	853	160	4,173
D-15	1,000	500	5,000	550	157	2,925
D-16	1,000	5	25,000	1,000	5	10,595
D-17	1,000	10	25,000	999	9	10,531
D-18	1,000	167	25,000	978	145	10,140
D-19	1,000	250	25,000	938	193	9,676
D-20	1,000	500	25,000	814	324	8,907

Tabel 8: *Characteristics of the class D before and after reductions.*

Graph	C_{opt}	C_{sph}	ΔC_{sph}	C_{best}	C_{avg}	C_{worst}	C_{σ}	ΔC_{avg}	ΔC_{worst}	N_{ga}
D-1	106	106	0	106	106	106	0	0	0	0
D-2	220	220	0	220	220	220	0	0	0	0
D-3	1,565	1,570	0.32	1,565	1,565	1,565	0	0	0	0
D-4	1,935	1,940	0.26	1,935	1,935	1,080	0	0	0	0
D-5	3,250	3,254	0.12	3,250	3,250	3,250	0	0	0	0
D-6	67	71	5.97	67	67.1	68	0.3	0.15	1.49	1
D-7	103	103	0	103	103	103	0	0	0	0
D-8	1,072	1,095	2.15	1,072	1,072.7	1,074	0.6	0.07	0.19	6
D-9	1,448	1,471	1,59	1,448	1,448.4	1,450	0.7	0.03	0.14	3
D-10	2,110	2,120	0.47	2,110	2,110	2,110	0	0	0	0
D-11	29	29	0	29	29	29	0	0	0	0
D-12	42	42	0	42	42	42	0	0	0	0
D-13	500	514	2.80	500	500.6	502	0.7	0.12	0.40	5
D-14	667	675	1.20	668	669.7	671	0.9	0.40	0.60	10
D-15	1,116	1,121	0.45	1,116	1,116	1,116	0	0	0	0
D-16	13	13	0	13	13	13	0	0	0	0
D-17	23	23	0	23	23	23	0	0	0	0
D-18	223	239	7.17	226	227.7	230	1.2	2.11	3.14	10
D-19	310	335	8.06	312	313.3	315	0.9	1.06	1.61	10
D-20	537	539	0.37	537	537	537	0	0	0	0

Table 9: *Comparison of solution quality for the graphs in class D.*

Graph	T_{bc1}	T_{bc2}	T_{sph}	T_{avg}	T_{σ}
D-1	476	200	486	523	8
D-2	284	148	488	537	13
D-3	2,290	106	785	650	39
D-4	3,529	41	689	554	21
D-5	811	37	522	504	8
D-6	2,340	4,148	687	788	44
D-7	100	1,037	681	795	29
D-8	6,985	17,858	13,237	2,101	381
D-9	4,630	16,458	29,354	2,744	624
D-10	1,312	1,678	14,780	1,100	169
D-11	1,374	24,609	949	1,070	59
D-12	305	5,843	961	1,085	20
D-13	1,864	91,718	15,187	2,357	245
D-14	3,538	61,335	41,237	2,601	393
D-15	1,410	16,889	24,828	1,302	102
D-16	871	9,721	956	1,047	21
D-17	6,965	147,598	950	1,068	26
D-18	245,192	227,841	31,015	2,536	491
D-19	878	304,380	50,003	3,441	580
D-20	47	1,276	50,010	2,638	658

Table 10: *Comparison of CPU-time in seconds for the graphs in class D.*

Graph	Problem size			Reduced size		
	n	m	$\mid E \mid$	n	m	$\mid E \mid$
E-1	2,500	5	3,125	680	5	1,286
E-2	2,500	10	3,125	710	9	1,328
E-3	2,500	417	3,125	637	199	1,233
E-4	2,500	625	3,125	435	164	964
E-5	2,500	1,250	3,125	222	108	649
E-6	2,500	5	5,000	1,845	5	4,318
E-7	2,500	10	5,000	1,891	10	4,372
E-8	2,500	417	5,000	1,723	286	4,193
E-9	2,500	625	5,000	1,608	358	4,069
E-10	2,500	1,250	5,000	1,046	366	3,388
E-11	2,500	5	12,500	2,498	5	12,093
E-12	2,500	10	12,500	2,500	10	12,123
E-13	2,500	417	12,500	2,341	321	11,760
E-14	2,500	625	12,500	2,139	388	11,325
E-15	2,500	1,250	12,500	1,461	443	8,514
E-16	2,500	5	62,500	2,500	5	129,332
E-17	2,500	10	62,500	2,500	10	29,090
E-18	2,500	417	62,500	2,429	355	28,437
E-19	2,500	625	62,500	2,351	485	27,779
E-20	2,500	1,250	62,500	1,988	758	24,423

Table 11: *Characteristics of the class E graphs before and after reductions.*

Graph	Cost					CPU-time(secs)			
	C_{opt}	C_{sph}	ΔC_{sph}	C_{ga}	ΔC_{ga}	T_{bc1}	T_{bc2}	T_{sph}	T_{ga}
E-1	111	111	0	111	0	1,150	1,394	7,334	7,395
E-2	214	216	0.93	216	0.93	6,251	1,993	7,355	7,444
E-3	4,013	4,060	1.17	4,013	0	26,468	15,782	50,004	9,449
E-4	5,101	5,113	0.24	5,102	0.02	46,008	1,660	29,921	7,763
E-5	8,128	8,134	0.07	8,128	0	12,564	411	9,318	7,474
E-6	73	76	4.11	73	0	678	-	10,060	10,148
E-7	145	149	2.76	145	0	27,124	-	10,306	10,458
E-8	2,640	2,690	1.89	2,646	0.23	118,618	-	50,013	12,896
E-9	3,604	3,671	1.86	3,611	0.19	24,528	-	50,014	14,933
E-10	5,600	5,624	0.43	5,600	0	39,261	-	50,014	12,976
E-11	34	34	0	34	0	1,901	-	14,472	14,559
E-12	67	68	1.49	68	1.49	7,200	-	14,497	14,588
E-13	1,280	1,317	2.89	1,289	0.70	207,059	-	50,003	21,787
E-14	1,732	1,767	2.02	1,736	0.23	29,263	-	50,030	23,022
E-15	2,784	2,795	0.40	2,784	0	7,666	-	50,020	18,424
E-16	15	15	0	15	0	179	-	14,425	14,586
E-17	25	25	0	25	0	36,040	-	14,458	14,619
E-18	572	625	9.27	583	1.92	-	-	50,017	29,105
E-19	758	802	5.80	766	1.06	16,372	-	50,037	27,319
E-20	1,342	1,357	1.12	1,342	0	272	-	50,055	25,107

Table 12: *Comparison of solution quality and CPU-time for the graphs in class E.*

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman,
Data Structures and Algorithms,
Addison-Wesley, Reading, Mass, 1983.

[2] Y. P. Aneja,
"An Integer Linear Programming Approach to the Steiner Problem in Graphs,"
Networks, Vol. 10, pp. 167-178, 1980.

[3] J. E. Beasley,
"An SST-Based Algorithm for the Steiner Problem in Graphs,"
Networks, Vol. 19, pp. 1-16, l989.

[4] J. E. Beasley,
"OR-Library: distributing test problems by electronic mail,"
Journal of the Operational Research Society, Vol. 41, pp. 1069-1072, 1990.

[5] Sunil Chopra, Edgar R. Gorres, M. R. Rao,
"Solving the Steiner Tree Problem on a Graph Using Branch and Cut,"
Operations Research Society of America Journal of Computing, Vol. 4, No. 3, pp. 320-335, 1992.

[6] R. Dionne, M. Florian,
"Exact and Approximate Algorithms for Optimal Network Design,"
Networks, Vol. 9, pp. 37-59, 1979.

[7] K. A. Dowsland,
"Hill-climbing simulated annealing and the Steiner problem in graphs,"
Eng. Opt., Vol. 17, pp. 91-107, 1991.

[8] S. E. Dreyfuss, R. A. Wagner,
"The Steiner problem in graphs,"
Networks, Vol. 1, pp. 195-207, 1971.

[9] C. W. Duin, A. Volgenant,
"Reduction Tests for the Steiner Problem in Graphs,"
Networks, Vol. 19, pp. 549-567, 1989.

[10] L. R. Foulds, V. J. Rayward-Smith,
"Steiner problems in graphs: algorithms and applications,"
Eng. Opt. Vol. 7, pp. 7-16, 1983.

[11] M. R. Garey, D. S. Johnson,
"The Rectilinear Steiner Tree Problem is NP-complete,"
SIAM Journal of Applied Mathematics, Vol. 32, No. 4, pp. 826-834, 1977.

[12] D. E. Goldberg,
Genetic Algorithms in Search, Optimization, and Machine Learning,
Addison-Wesley, 1989.

[13] S. L. Hakami,
"Steiner's problem in graphs and its implications,"
Networks, Vol. 1, pp. 113-133, 1971.

[14] M. Hanan,
"On Steiner's Problem with Rectilinear Distance,"
SIAM Journal of Applied Mathematics, Vol. 14, No. 2, pp. 255-265, 1966.

[15] J. Hesser, R. Männer, O. Stucky,
"Optimization of Steiner Trees using Genetic Algorithms,"
Proceedings of the 3th International Conference on Genetic Algorithms,
pp. 231-236, 1989.

[16] John H. Holland,
Adaption in Natural and Artificial Systems,
University of Michigan Press, Ann Arbor, MI., 1975.

[17] Bryant A. Julstrom,
"A Genetic Algorithm for the Rectilinear Steiner Problem,"
Proceedings of the 5th international Conference on Genetic Algorithms,
pp. 474-480, 1993.

[18] A. Kapsalis, V. J. Rayward-Smith, G. D. Smith,
"Solving the Graphical Steiner Tree Problem Using Genetic Algorithms,"
Journal of the Operational Research Society, Vol. 44, No. 4, pp. 397-406,
1993.

[19] R. M. Karp,
"Reducibility among Combinatorial Problems"
In R. E. Miller, J. W. Thatcher (Eds.), *Complexity of Computer Computations*, Plenum Press, New York, pp. 85-103, 1972.

[20] L. Kou, G. Markowsky, L. Berman,
"A Fast Algorithm for Steiner Trees,"
Acta Informatica, Vol. 15, pp. 141-145, 1981.

[21] E. L. Lawler,
Combinatorial Optimization: Networks and Matroids,
Holt, Rinehart and Winston, New York, 1976.

[22] J. van Leeuwen,
"Graph Algorithms,"
in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. A: *Algorithms and Complexity*, Elsevier, Amsterdam, 1990.

[23] A. Lucena, J. E. Beasley,
"A branch and cut algorithm for the Steiner problem in graphs,"
working paper, The Management School, Imperial College, England, July 1992.

[24] J. Plesnik,
"A bound for the Steiner tree problem in graphs,"
Math. Slovaca, Vol. 31, pp. 155-163, 1981.

[25] V. J. Rayward-Smith, A. Clare,
"On finding Steiner vertices,"
Networks, Vol. 16, pp. 283-294, 1986.

[26] M. L. Shore, L. R. Foulds, P. B. Gibbons,
"An algorithm for the Steiner Problem in Graphs,"
Networks, Vol. 12, pp. 323-333, 1982.

[27] H. Takahashi, A. Matsuyama,
"An Approximate Solution for the Steiner Problem in Graphs",
Mathematica Japonica, Vol. 24, No. 6, pp. 573-577, 1980.

[28] R. Venkateswaran, P. Mazumder,
"Routing Algorithms in Semiconductor Circuit Design,"

In preparation, 1993.

[29] Darrell Whitley,
"The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best,"
Proceedings of the 3th International Conference on Genetic Algorithms, pp. 116-121, 1989.

[30] Pawel Winter,
"Steiner Problem in Networks: A Survey,"
Networks, Vol. 17, pp. 129-167, 1987.

[31] Pawel Winter, J. MacGregor Smith,
"Path-Distance Heuristics for the Steiner Problem in Undirected Networks,"
Algorithmica, Vol. 7, pp. 309-327, 1992.