Steiner Trees in Industries
D.-Z. Du and X. Cheng (Eds.) pp. – - –

Genetic Algorithm Approaches to Solve Various Steiner Tree Problems

Goutam Chakraborty
Department of Software & Information Science
Iwate Prefectural University, Takizawamura
JAPAN 020-0193
E-mail: goutam@soft.iwate-pu.ac.jp

Contents

1 Introduction

Finding optimum Steiner tree is a well known **NP**- hard problem. Many recent works show that Genetic Algorithm (GA) is a viable alternative to heuristic approaches to solve constrained optimization problems. In the original problem proposed by Georg Steiner, the nodes are points in a plane and the costs of the links are Euclidean distance metric i.e. triangle inequality holds. In [1] it was shown that this problem is **NP**- complete.

Depending on the application, there are variations to the original Steiner tree problem. In the next section, we will give brief introduction to four different variations indicating corresponding application areas.

Different genetic algorithm based approaches to solve Steiner tree problems are discussed in this article. In section 3, a brief introduction to genetic algorithm [54], [55], [56], [57] and how it works to find global optimum solution in case of **NP**-complete problems, is described. Following that, in section 4, we will introduce genetic algorithm approaches for solving the problem of minimum Steiner tree in graph [24], [25], [61], [62], [63]. The solution to a delay constrained version of the minimum cost Steiner tree in graph problem was recently proposed in [36], and is explained in section 5. Genetic algorithm approach to solve minimum Euclidean Steiner tree problem, as proposed by Hesser [44], is introduced in section 6, followed by the proposal by Julstrom [53] to solve minimum Euclidean rectilinear Steiner tree problem in section 7.

While calculating chromosome fitnesses, most of the GA based approaches depend on some previously proposed heuristic algorithms and/or problem reduction techniques, which will be briefly discussed as and when necessary. We will also compare results obtained by other conventional methods whenever available. In conclusion, in section 8, we will mention the strong and the weak points of the genetic algorithm approaches, and discuss what are yet to be done to accept it as a reliable alternative for solving various Steiner tree problems.

2 Variations of Steiner Tree Problem and Applications

2.1 Minimum Steiner Tree in Graph (*MStTG*)

Given a distance graph $G = (V, E)$ and a subset of vertices $D \subseteq V$, a *Steiner tree* is a tree in G that spans all the nodes in $D = \{d_1, d_2, \dots, d_K\}$. The *minimum Steiner tree in graph* (*MStTG*) is the Steiner tree for which the total distance on its edges is minimum among all possible Steiner trees for given G and D. It is possible that the *MStTG* uses set of nodes $D' = \{d_1, d_2, \dots, d_K, \dots, d_{K'}\}$, where $D' \supseteq D$. The nodes in $(D' - D)$ is called Steiner nodes. In the example of Fig. 1, *MStTG* is shown with thick lines. Here $D = \{v_1, v_4, v_5, v_7, v_8\}$, but other node v_6 of the original graph G is used to construct the *MStTG*. Thus node v_6 is a Steiner node.

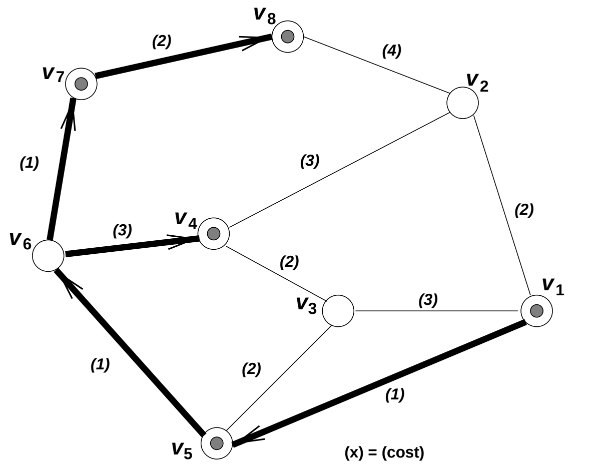

Figure 1: An example of Steiner tree in Graph

The problem of finding *MStTG* is proved to be **NP**-complete [1], [2]. Considerable research works had been done to solve it during the last three decades. Several algorithms, mostly during 70's were proposed [6], [7], [8], [9] to find the optimum solutions. But they are exponential in computation time and are applicable only for small problems. During early to mid 80's, many polynomial time heuristic algorithms [10], [11], [12], [13], [14], [15] were proposed, which are very efficient to find good near optimum solutions. Some of them [11] also ensure worst case bound on solution quality. During late 80's, there were proposals [16], [17] to reduce the complexity of the problem by some reduction methods for the number of vertices and edges. The most recent heuristic algorithms are due to Beasley in '89 [18], Voss in '92 [19], Winter et al. in '92 [20], Chopra in '92 [21], and Zelikovsky in '93 [22]. From early 90's there is a new trend of approach using soft computing methods and random algorithms to solve the $MStTG$. The earliest one is using simulated annealing due to Dowsland in '91 [23], followed by proposals based on genetic algorithm in mid 90's [24], [25]. Neural network algorithms are also proposed around the same time [26], which we will discuss in another article in this book.

$MStTG$ finds application in many problems where there is an existing network and we need to join optimally a sub-set of vertices. In recent years, the most useful application is to connect servers on a computer network for multicast communication. Other applications are like distribution of electricity from a source to different destinations over a power grid network, multiprocessor scheduling. A nice survey of this problem and applications could be found in [27], and [28]. A variation of the $MStTG$ problem arises in multicast communication when continuous media (like audio, video) is to be transmitted over the network. This problem is discussed in the next section 2.2.

2.2 Constrained Minimum Steiner Tree in Graph ($CMStTG$)

With the recent advancement in computer, high speed transmission and switching technology, computer networks are now capable of carrying continuous media traffic like voice and video. Most of these multimedia applications are not one-to-one but one-to-many or many-to-many and need multicast support. Multicast is the ability to logically connect a subset of hosts in a network. A packet switched network (PSN) is said to be able to provide multicast service, if it can deliver copies of a packet to a set of destinations simultaneously. Here, optimization goal is to minimize the overall cost of the multicast tree - which is the $MStTG$ problem. Yet, for meaningful multimedia communication, some quality of services (QoS) requirements has to be satisfied for all end-to-end transmissions. Most important requirements are end-to-end delay, delay-jitter and bandwidth. End-to-end delay is the total added delay on the links of the path from source to destination. This has to be considered for all paths from source to different destinations, while constructing the multicast Steiner tree. We name this problem as constrained minimum Steiner tree in graph $CMStTG$.

The problem of $CMStTG$ is illustrated with a simple example in Fig. 2. With every link two parameters are assigned, the first is the cost of the link and the second is the delay. Here, v_1 is the source, and the destination nodes are $\{v_4, v_5, v_7, v_8\}$. Apart from reaching all the destinations from the source, a constraint that the delay should be $\leq$ 15 units, has to be maintained. Path $\langle v_1, v_5, v_6, v_7, v_8 \rangle$ from source v_1 to destination v_8, as was selected in Fig. 1 with its total path delay of 18, could not satisfy this delay constraint. Therefore, though costlier, in this situation path $\langle v_1, v_2, v_8 \rangle$ has to be chosen to reach v_8 from v_1.

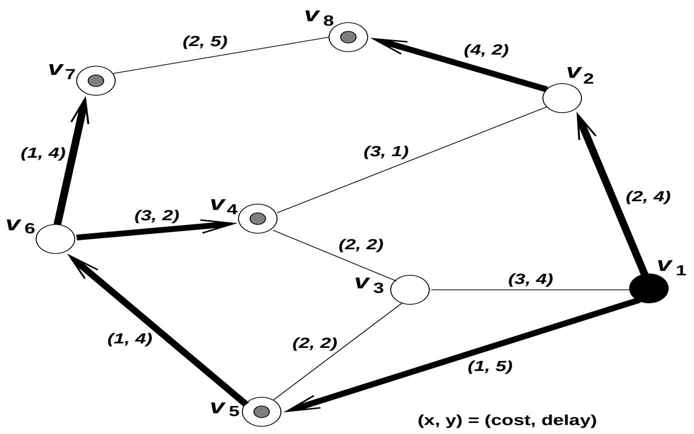

Figure 2: A Constrained Minimum Steiner tree

Heuristic solution to this constrained $MStTG$ problem was proposed in [29], [30], [31], [32]. A neural-network approach to solve this problem was proposed in [33], [34]. Recently there are two GA based papers [35], [36] on multiple destination routing problems, i.e. multicast routing. Leung et al. paper [35] did not consider any constraint, whereas Zhang's paper [36] considered the delay constraint. In Section 5, we will discuss Zhang's work in detail.

Another variation of the multicasting problem in network arises, when the participants join and leave the group during the lifetime of the session.

This is called dynamic multicasting. Even when the joining and leaving time of the participants are available, it is a challenging problem to optimize the total cost of the Steiner tree over the whole session period. A heuristic solution is proposed in [37]. A genetic algorithm approach could be an interesting research topic.

2.3 Minimum Euclidean Steiner Tree (*MEStT*)

Apart from the above problem of Steiner tree in graphs, there is another class of problems which arises during the planning and construction stage of large networks. There we only have a set of points (nodes) in a defined space to be connected optimally, and there is no underlying graph already present. In general, we can add intermediate points arbitrarily in number as well as at any locations. This type of Steiner tree problem is called as Euclidean Steiner tree problem, mainly because the applications involve Euclidean distance metric. We also confine all the subsequent discussions in 2-dimensional plane.

Formally, we have a set $P = \{p_1, p_2, \ldots, p_N\}$ of N coplanar points, often called *site nodes*. The minimum Euclidean Steiner tree problem (*MEStT*) is to find the shortest tree connecting all N points, where the tree may contain nodes in the plane other than the *site nodes*. This set $S = \{s_1, s_2, \ldots, s_M\}$ of M extra vertices are called the *Steiner nodes*.

A simple example is given in Fig. 3. Here, the four nodes shown as hollow circles are to be connected by a minimum distance tree. If we just connect them by minimum spanning tree ($MSpT$), we get a tree as in Fig. 3(a) with cost 10. But if we add an extra (Steiner) node, shown as solid circle in Fig. 3(b), then the distance of the tree is reduced by more than 10%.

Similar comparison is shown in Fig. 4. Fig. 4(a) is the $MSpT$ and Fig. 4(b) is the $MEStT$, where three Steiner nodes are introduced.

Finding $MEStT$ is more flexible in the sense that one can add any number of intermediate nodes at any suitable locations to make the distance of the tree minimum. Yet, for the same reason, it makes the computation hard and it grows exponentially with increase in N. This problem had been proved to be **NP**-complete [38]. Exact methods [39], [40] to find the $MEStT$ are based on finding the optimum among all *full Steiner trees*, where the number of steiner nodes $K = (N - 2)$. They decomposed the problem into smaller subsets and solved. Even then it is computationally heavy and could not be used for $N > 30$. Cockayne et al. [41] further improved their earlier algorithm proposed in [40] to solve problems up to 100 site nodes in

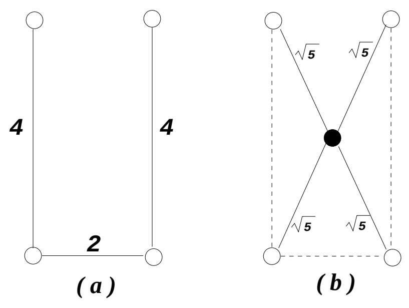

Figure 3: Minimum spanning tree vs. minimum Steiner tree: Example 1

reasonable time. The best known heuristic, based on spanning tree, is by Beasley [42]. A simulated annealing algorithm was proposed by Lundy [43]. Comprehensive surveys are done by Gilbert and Pollak [38], and Hwang and Richards [28]. In this article, we will discuss genetic algorithm approach by Hesser [44] in section 6.

MEStT problem finds its applications at the stage of planning and construction of large networks, say telecommunication network, power distribution network, or laying of oil pipelines. In such construction works, most of the cost is involved in realizing the links, and in general we can choose intermediate Steiner points more or less arbitrarily. In [45], a practical example for constructing communication network connecting the main cities of US is discussed with projected cost savings by proper design of Steiner nodes. We face similar problems for designing printed circuit boards, VLSI packages, mechanical systems in buildings etc. where very high order *MEStT* problems are to be solved.

2.4 Minimum Euclidean Rectilinear Steiner Tree (*MERStT*)

The *Euclidean Rectilinear Steiner Tree* (*ERStT*) problem is similar to Euclidean Steiner tree problem, with the restriction that the edges to connect the set of nodes are all horizontal and vertical line segments. Here too we consider 2-dimensional Euclidean space. As usual the problem is to find the Minimum distance Euclidean Rectilinear Steiner Tree (*MERStT*). In Fig. 5 we illustrate this problem. Here, 9 site nodes are connected by horizontal and vertical line segments to form the *MERStT*, as shown in Fig 5(b).

Garey and Johnson [46] had shown that the general case of *MERStT* is

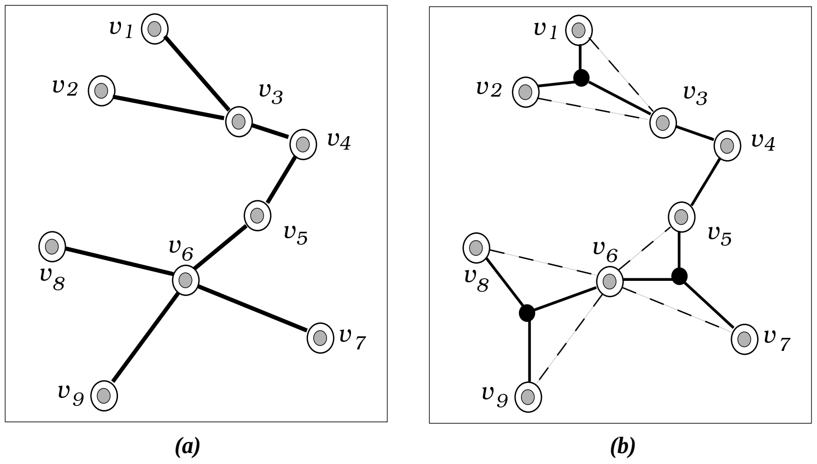

Figure 4: Minimum spanning tree vs. minimum Steiner tree: Example 2

NP-complete. Polynomial time algorithms for special cases were proposed by Aho, Garey and Hwang [47], and Agarwal and Shing [48]. Hwang in [49] had shown that the cost ratio of of *MSpT* and *MERStT* is not more than 3/2 and therefore many heuristic algorithms [50], [51] take *MSpT* as the starting point. The heuristic algorithm proposed by Ho, Vijayan and Wong [52] could produce optimum solutions under some conditions (e.g. when the layout of each edge is L-shape etc.). A genetic algorithm approach for *MERStT* by Julstrom [53] will be discussed in detail in section 7.

3 Introduction to Genetic Algorithm

Genetic Algorithm (GA) is a search algorithm based on the mechanics of natural selection [54]. Compared to other approaches, they are superior because, of wide applicability. They make few assumptions from the problem domain, and are not biased towards local minimums. At the same time, GAs are very efficient to direct the search towards relatively prospective regions of the search space.

The first step in GA is to encode the solution of the problem in binary bit string. The solution in its original form is referred to as *phenotype*, whereas its binary encoded version is called *genotype* or *chromosome*. It is best to have a one-to-one mapping between the solution of the problem and

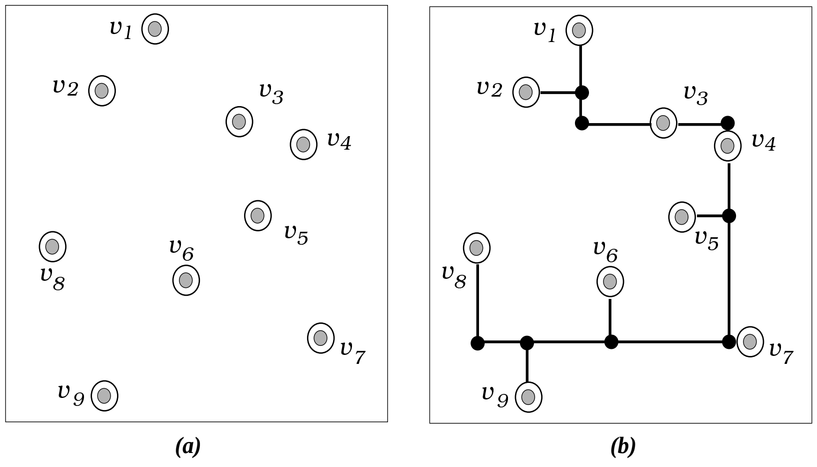

Figure 5: Minimum rectilinear Steiner tree

the chromosome representation. But it is possible to have a one-to-many mapping, where these redundant chromosomes could be the cause of inefficiency of GA. Many-to-one mapping from solution to chromosome is also possible, where the result obtained using GA would lack details and need some post-processing. The phenotype may be thought of as the semantics or the interpretation of the genotype. In general, there should be an easy and preferably injective mapping from genotype to phenotype. Each chromosome is composed of *genes*, the basic units of informations. A gene is usually a simple binary bit. It may be different, as we will see in orthogonal genetic algorithm in section 5. Though, for some problems, the shape of the search space and therefore the complexity of the search depends on this coding, in practice it is done in the easiest and most obvious way. In many problems, where the natural representation of the solution is in 1/0 string, phenotype and genotype are same.

Next, a pool of solutions of the problem, called initial population, is created. These solutions are generated simply randomly, without any consideration to how good they are. A fitness function has to be defined to measure the goodness of these encoded solutions. Genetic operators *selection*, *crossover* and *mutation* operate on the population to generate new population, i.e. new set of solutions, from the old ones. Good solutions are *selected* with greater probability to the next generation, in line with the idea of *survival of the fittest.* *Crossover* operation recombines arbitrarily selected solutions pairwise, by interchanging portions of them, producing divergent solutions to explore the search space. An occasional *mutation* operation is performed on a chromosome by flipping a bit at random position of the encoded chromosome, to facilitate jumping of solutions to new unexplored regions of the search space. As the algorithm continues and newer generations evolve, the quality of solutions improve. The success of genetic algorithm is explained by schema theorem and building block hypothesis in [55].

Many strategies for fitness calculation, selection, crossover and mutation are proposed. The basic steps for the Standard Genetic Algorithm (SGA) are shown below.

```
Algorithm SGA ( $g, MAX, \Phi(g), P$ )
01 begin
02     $g = 0$;  /* $g$: generation number */
03     Create $P$ members of the initial population $\Phi(0)$;
               /* $P$: population size,*/
               /* $\Phi(g)$: set of members at generation $g$ */
04     Calculate fitnesses of the members of $\Pi(0)$;
05     while ( $g \leq MAX$ )     /* $MAX$: maximum generation */
06        $g := g + 1$;
07        $\Phi''(g) \overset{selection}{\longleftarrow} \Phi(g-1)$;
08        $\Phi'(g) \overset{crossover}{\longleftarrow} \Phi''(g)$;
09        $\Phi(g) \overset{mutation}{\longleftarrow} \Phi'(g)$;
10     endwhile
11 end
```

Figure 6: Algorithm for Standard Genetic Algorithm

The success of genetic search depends on balancing the two aspects of (1) population diversity i.e. exploring the different regions of the search space, and (2) selection pressure i.e. to get to the optimum point fast. If the best few members of the initial population predominate the whole population in a few early generations, due to their much better fitnesses, it would result in poor exploration and premature convergence to suboptimal solution. On the other hand, at later stage of the search, when high performance regions are identified, disruption of good chromosomes after crossover with bad ones would slow down the process of reaching the global optimum. A number of strategies were proposed [56](chapter 4 & 6), [58], [59], [60] to overcome this problem by setting a balance between diversity of solutions in the beginning and selection pressure to concentrate on the best during later generations.

For constrained optimization problems, all encoded strings may not satisfy the different constraints. There are two possible approaches to overcome this problem:

1. Use the standard genetic algorithm and allow all encoded solutions, valid and invalid. Invalid solutions are penalized so that they may not go to the next generations. Valid solutions are assigned fitnesses according to how good they are, with respect to the optimization criteria.

2. The chromosome representation, the crossover and mutation operations are defined such that invalid solutions are always avoided. Higher fitnesses are assigned to chromosomes representing better solutions.

The above approach (1) is easier and softer (problem independent). But in practice they can produce optimum or near optimum results only for small problems. When the size of the problem and/or the accuracy of the solutions are increased, due to explosive increase in the size of the search space mostly crowded with invalid solutions, finding optimum or near optimum valid solution is almost impossible. The other problem is how to decide to what extent the invalid solutions are to be penalized. Between two invalid solutions, to what extent they are violating the constraint is difficult to judge? The fitness calculation then heavily affects the efficiency of the algorithm, and the quality of the solution.

Approach (2) is harder (problem specific approach) because we have to define the solution representation and/or genetic operators, so that the chromosomes always represent valid solutions. Therefore, the algorithm gets more strongly associated with the particular problem. As the valid and invalid regions are interlaced, it is difficult to define genetic operations that make the offsprings always valid. Correcting invalid chromosomes to their nearby valid ones is computationally costly. Yet, because the search space is now restricted to valid solutions only, the algorithm is more efficient, and the quality of solutions much better.

In Steiner tree problem, it is difficult to define problem specific crossover operators which would combine good parts of different solutions i.e. which in essence will support building block hypothesis. In this article we discuss how these problems are overcome in different research works.

4 Solving *MStTG* Problem by Genetic Algorithm

In [24] and [25], GA was used to solve *MStTG* problem. They are two different approaches. We discuss them separately, in sections 4.1 and 4.2.

4.1 The Work by Kapsalis et al. 1993

Kapsalis, Rayward-Smith and Smith [24] for the first time used GA to solve *MStTG* problem.

General considerations

Let the given graph be $G = (V, E)$. $D = \{d_1, d_2, \ldots, d_K\}$ is the set of nodes to be spanned and $D' = \{d_1,\ d_2, \ldots,\ d_K, \ldots,\ d_{K'}\}$ are the nodes actually spanned by the Steiner tree. The nodes in $(D' - D)$ are Steiner nodes. The chromosome is a bit string of length $|V| = N$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of nodes of the original graph. Bit position i in the string represent node v_i of G. If node v_i is in the Steiner tree the corresponding bit is 1, and otherwise 0. For the example in Fig. 7(a), as the number of nodes in G is 8, the length of the chromosome, shown in Fig. 7(b), is 8. The Steiner tree is formed with nodes v_1, v_4, v_5, v_6, v_7, v_8 and therefore we have 1s in the corresponding positions of the chromosome, as shown in Fig. 7(b).

When a chromosome represent a solution to the Steiner tree problem, all positions representing nodes in D have to be 1. There are two ways to enforce this condition,

1. Allow any bit string and penalize those chromosomes that does not contain nodes in D, by assigning very low fitnesses.

2. Logically OR every chromosome with the one that represents only the nodes in D and thus try to make them valid. This does not always make the chromosome valid, as those nodes may not form a tree.

The second strategy works better, especially for bigger problems. One possibility is to consider only $G - D$ nodes and form chromosomes of length $|G - D|$ to search for the Steiner nodes. Nodes in D are selected by default.

Another possible approach is to select proper edges of the graph. Then the chromosome length will be equal to the number of edges in the graph. GA would search the set of edges needed to form the optimum Steiner tree. Here fitness evaluation is easy, but the length of the chromosome would obviously be longer and therefore the search space is much wider with many invalid members.

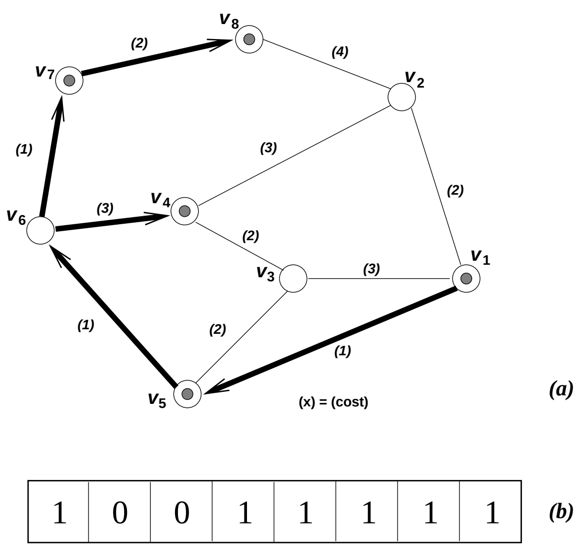

Figure 7: An example with 8 nodes graph

For creating the initial population many approaches are possible. The simplest is to create them at random, i.e., (1) put 1s at random positions in the chromosome. This will create many invalid solutions too. The other possibilities are, (2) this randomly created population is seeded by including a feasible solution equivalent to the minimum spanning tree of the entire graph, and (3) this minimum spanning tree is trimmed by a heuristic.

Regarding population size, the bigger is the better in general. A large population is a must when the search space is multimodal with many local optimums. But it means heavier computation and longer run time. Small population size may, on the other hand, lead to limited exploration of the search space and get trapped in local optimum. Kapsalis et al. In [24] found that a population size of 10 is a good choice.

Fitness evaluation: The fitness evaluation of the chromosomes were done as follows. The chromosome represents a set of nodes $U \subseteq V$. A subgraph of G induced by $U \cup D$ is constructed. Let the subgraph consists of $L \geq 1$ components, where all L components are separated. The minimum spanning tree for those components were found using Prim's algorithm [4]. Now the components are summed up to form the tree and the total cost is calculated. For $L > 1$, a large positive penalty, linearly proportional to L, is added to the previously calculated cost. This cost for every member of the population is calculated, and finally subtracted from the maximum of them to find the fitness. Thus the tree with least components and cost will have highest fitness, and the worst member will have fitness 0. As Prim's algorithm is fast, and additionally some evaluations could be stored and reused, the fitness calculation for all members could be done fast.

Genetic operations

Selection: Kapsalis et al. [24] used two types of selection mechanisms.

1. *Roulette selection* [55]: Here the probability of a solution to be selected to the next generation is proportional to its fitness. Naturally a solution with high fitness could be multiply copied to the next generation.

2. *Ranking* [56]: The solutions are ranked according to their fitnesses, better fitness higher rank. A solution is selected according to its position using a scaled Fibonacci sequence to favor solutions with higher rank.

Crossover and Mutation: After chromosomes are selected to the next generation, *crossover* and *mutation* operations are performed on them. For crossover, a portion of the population, determined by crossover probability p_c, is chosen randomly. It is then rounded to even number and paired, again randomly. For every pair, a location is randomly selected and the portion from that location to the end is swapped. It is reported that the value of p_c has little effect on performance, and $p_c = 0.9$ is marginally better. For mutation, bits on all chromosomes are flipped randomly with mutation probability (p_m), and a value of $p_m = 0.02$ is used. It was also mentioned that the performance was unaffected over wide choice of p_c and p_m.

New generation: After selection, crossover and mutation, a new pool of members is generated. The new population could:

1. Replace the whole of the old population, as in Fig. 6 (Replace all).

2. Replace the worst solution of the old population by the best solution of the new generation. Replace the second worst solution of the old population by the second best of the new population and so on, until no improvement is achieved (Best for worst).

Thus, one cycle of generation is completed. The cycles are repeated until some stopping criterion is reached, like having satisfactory solution or reaching execution time limit.

4.1.1 Simulation, Results and Discussion

Kapsalis et al. tried the above algorithm (with variations as mentioned) on the B-problem set from [64]. In [64], there are five sets of graphs, set A to set E, with increasing number of nodes and different node degrees. In B-set, there is a total of 18 examples. Six examples are of 50-nodes graph, three with average node degree 2.5 and three with average node degree 4.0. The three examples of the same node degree differ in number of destination nodes, where D consists of $\approx$ 17%, $\approx$ 25% and $\approx$ 50% of the total number of nodes in the graph. Similarly, six examples of 75-nodes and 100-nodes graphs each, make the set of 18 examples. The graphs are rather sparsely connected. The stopping criterion was set as either, (1) the optimum result was obtained (optimum result is known for all the examples), or (2) an execution time of 4000 seconds in Apple Mac IIfx was elapsed. The algorithm outperform two well known heuristics, the shortest path heuristic (SPH) [10], and the average distance heuristic (ADH) [15]. They found that random initial population, roulette selection and replace-all strategies work better in general. For all the B-problems in [64], the algorithm could find the optimum solutions. Values of p_m and p_c over a wide range gave similar performances. Experiments with different initialization procedures and selection mechanisms were also reported in [24], and found not to affect the result strongly. Only in case of trimmed initial population, the result sometimes gets trapped in the local minimum.

Discussion: The experiments were tried on simple problems of sparsely connected graphs, with maximum of 100 nodes only. It is not clear how it would work for bigger problems. The main weak point of the proposed algorithm is the fitness evaluation. Simply merging the penalty term, which is linearly proportional to the number of components L, and the cost of the different component trees, is not the best approach. This proportionality constant will be problem dependent, on the relative values of the edge costs. With a graph, where edge costs varies over a wide range, this algorithm will not be robust. Instead, a rank based selection, where first the solutions are categorized according to the number of components (L), and within a category ordering them according to the cost, would be more robust and could perform better on wider varieties of problems.

4.2 The Work by Esbensen 1995

General considerations

We will discuss in details the work by Esbensen [63], which though computationally heavy, performs better on bigger problems. The main idea of the algorithm is to use the popular Kou, Markowsky, Berman (KMB) heuristic [11] for evaluating the chromosomes, and though costly, fitness evaluation is more appropriate. This algorithm also exploits many other domain based knowledge (from different heristics) to improve the efficiency as well as the quality of results.

At the onset, a preprocessing on the original graph is done to reduce its size. Standard genetic algorithm as described in Fig. 6 is used, with the additional feature of using elitist model and domain specific knowledge to filter chromosomes during genetic operations. In elitist model, the best chromosome from earlier generation is preserved, if it is better than the best of the current generation. At the end, one could preserve the best of all generations. We will discuss the structure of the chromosome, and crossover and mutation operations specifically designed in [63].

Preprocessing for Graph Reduction

This step has nothing to do with genetic algorithm, but reduces the size of the graph and thus the computation time of further steps in genetic algorithm. These reduction steps are adopted from [20], [27] and illustrated in Fig. 8. The steps are as follows:

Step a: As shown in Fig. 8 (Step a), suppose there is a node v of degree 1 connected to a node w. If $v \notin D$, then node v and edge e_{vw} can be deleted. Even if $v \in D$, as e_{vw} must be included in $MStT$, still node v and edge e_{vw} can be deleted and w included in D.

Step b: As shown in Fig. 8 (Step b), suppose there is a node $v \notin D$ of degree 2 connected to nodes u and w. In that case, node v and edges e_{uv} and e_{vw} can be deleted and replaced by one edge e_{uw} whose cost is $c_{uv} + c_{vw}$. If an edge e_{uw} already exists and $c_{uw} \leq (c_{uv} + c_{vw})$, then only e_{uw} should remain. Here c_{uw} represents the cost of path uw etc..

Step c: An edge e_{uv} is to be deleted when the cost of the shortest path (sp) from u to v i.e. $c_{sp(uv)} < c_{uv}$.

Step d: Suppose $v \in D$ and its closest neighbor is u and the second closest neighbor is w as in Fig. 8 (Step d). If there is no such w, assume

$c_{vw} = \infty$. Let there is another node $z \neq v$ and $z \in D$, which is closest to u. If $(c_{uv} + c_{sp(uz)}) \leq c_{vw}$, then the $sp(zu)$ and e_{uv} can be replaced by a single edge of cost $(c_{uv} + c_{sp(uz)})$ as shown in Fig. 8 (Step d).

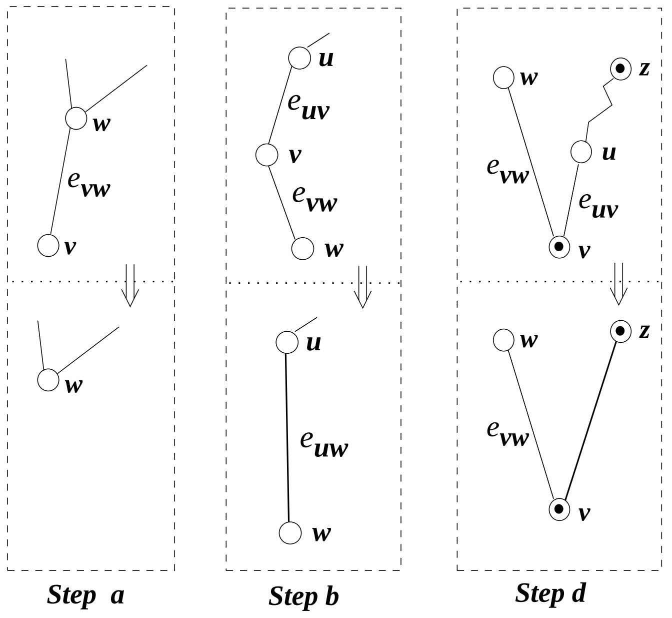

Figure 8: Steps of graph reduction

To perform the above steps, and during fitness evaluation using KMB, shortest path between different nodes are frequently needed. Therefore, the distance graph $D(G)$ is computed initially using Floyd-Warshall's algorithm [3] and dynamically modified as the reduction proceeds. The above steps are performed repeatedly until there is no further reduction in graph size. There is no indication in which order the above steps should be performed to have best results. In [63], they repeatedly executed the cycle of sequence of step c, step b, step d, and step a.

KMB Heuristic

At the heart of Esbensen's work [63] is the heuristic proposed by KMB [11], which is used to evaluate different genotypes (chromosomes). KMB algorithm can be stated in the following five steps:

1. Construct the subgraph G_1 of $D(G)$ induced by D.

2. Compute *MSpT* T_1 of G_1.

3. Replace the shortest paths in T_1 by the edges of the original graph G to obtain G_2.

4. Compute *MSpT* T_2 of G_2.

5. Delete, until possible, all nodes in T_2 that are $\notin D$ and with degree 1, to obtain *MStTG*.

The most expensive is step 1 for the construction of $D(G)$. But, during graph reduction $D(G)$ is constructed. If $D(G)$ is not available, the worst case time complexity of KMB is $O(K \times N^2)$, where $|D| = K$, and $|V| = N$.

The Genetic Algorithm

Genotype, phenotype and decoding: The genotype, similar to that in KRS [24] work, is a string of 1s and 0s representing the nodes of the graph. A 1 means the node is selected for Steiner tree. Let the set of these nodes are U. The corresponding phenotype, i.e. the actual realization of the Steiner tree, is constructed by KMB using $U \cup D$ as the set of vertices to be spanned. Thus, every time a genotype is to be evaluated for fitness, KMB is to be run. The other difference from KRS is the construction of the genotype (chromosome). In KRS, the bit position defines the particular node it represents. Here, to realize other genetic operations defined in this work, the meaning of a genotype has to be independent of the order of bits in it. Every bit of the genotype is associated with a tag, fixed with it, representing the node label of G, as shown in Fig. 9. The two genotypes in Fig. 9(b) and Fig. 9(c) are interpreted to exactly same phenotype and same solution of the problem. Thus, phenotype is invariant to reordering of tuples in genotype.

Some additional filtering of genotype is done using problem domain knowledge. It is known that, in the optimum Steiner tree the number of Steiner vertices would at most be $(|D| - 2)$. If $(|D| - |U|) < 2$, then the excess $|U| - (|D| - 2)$ number of 1s are randomly selected and changed to 0s in the genotype. This filtering is performed after initialization and at every generation (after crossover and mutation), on all members of the population.

Next comes the decoding and fitness evaluation of a genotype. Though other faster heuristics e.g., shortest path heuristic [10], average distance

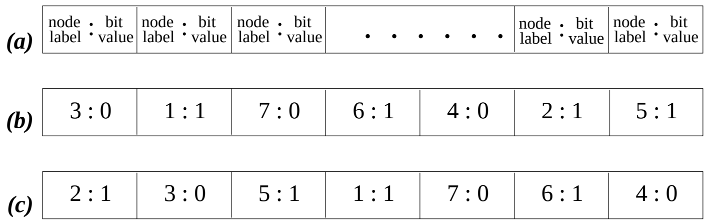

Figure 9: The structure of the genotype

heuristic [15], were available, KMB is chosen to decode the genotype for the following reasons:

1. The result quality of KMB outperforms other heuristics and provides a worst case lower bound on the solution quality.

2. It can decode any set of Steiner vertices (represented by the genotype) to a valid Steiner tree. Thus the confusing penalty term for invalid solutions could be avoided.

Fitness calculation: In the population $\Phi = \{\phi_0, \phi_1, \ldots, \phi_{P-1}\}$, for every member ϕ_i, the corresponding Steiner tree and its cost is determined by running KMB. The members are then ranked with highest cost first (i.e. the least fit first) and the lowest cost last. The fitness of the i^{th} ranked member is computed as $2 \times i/(P-1)$, where $P = |\Phi|$ is the population size. Fitness calculation from ranking rather than from absolute value of the solution quality is preferable [65], because it facilitates better balance of exploration of the search space and selection pressure.

Crossover, mutation and inversion operators: The crossover operation is same as classic one point crossover between two randomly selected members of the population, say ϕ_i and ϕ_j, at a randomly selected crossover point. As the different ordinal positions of two genotypes represent different nodes of the graph, they have to be rearranged first so that they both conform for crossover i.e., the ordinal positions describe the same nodes. This is explained in Fig. 10. Here tuples in ϕ_j are rearranged. $|D| = 5$ is assumed. After crossover, at crossover point 4, the number of 1s in ϕ_i become more than $(|D|-2)$, and was required to be filtered to reduce the number of 1s to $(|D|-2)$. Please note that, nodes in D are not included in the chromosome, and nodes in $(V-D)$ are labelled from 0 to 7, just for convenience.

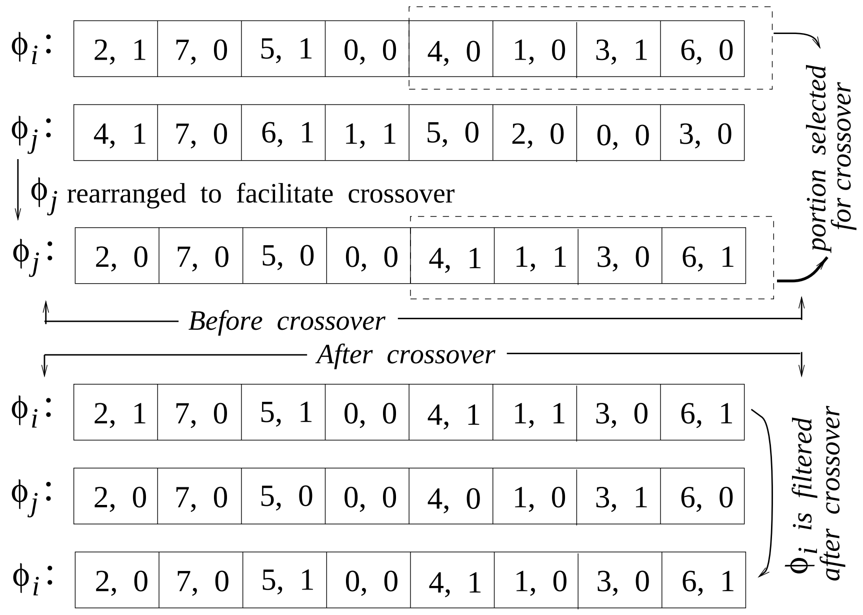
ϕi: 2, 1 7, 0 5, 1 0, 0 4, 0 1, 0 3, 1 6, 0
ϕj: 4, 1 7, 0 6, 1 1, 1 5, 0 2, 0 0, 0 3, 0
ϕj rearranged to facilitate crossover
ϕj: 2, 0 7, 0 5, 0 0, 0 4, 1 1, 1 3, 0 6, 1
portion selected for crossover
Before crossover
After crossover
ϕi: 2, 1 7, 0 5, 1 0, 0 4, 1 1, 1 3, 0 6, 1
ϕj: 2, 0 7, 0 5, 0 0, 0 4, 0 1, 0 3, 1 6, 0
ϕi: 2, 0 7, 0 5, 1 0, 0 4, 1 1, 0 3, 0 6, 1
ϕi is filtered after crossover

Figure 10: Crossover operation and subsequent filtering. Here, $|D| = 5$, crossover point $= 4$

For mutation operation, a bit is flipped with a low probability of p_{mut}. If, due to mutation, the number of 1s in genotype crosses $(|D| - 2)$, filtering is done. In addition to mutation, an inversion operation is performed with probability p_{inv}. If the genotype is considered as a ring and tuples are shifted and rotated by any random number of positions, the phenotype interpretation remains the same. But the genotype and the result of crossover changes. This shift and rotation of tuples in genotype is called inversion and it facilitates more exploration of the search space.

Overall algorithm: Now all the genetic operations defined, the overall execution of the algorithm is as follows.

```
01    **Graph reduction**;
02    Create $\phi_i$ ($i$ from 0 to $(P-1)$) members of the initial population $\Phi$;
03    Evaluate and rank members of $\Phi$ using KMB;
04    Save $\lambda$ = best of $\Phi$;
05    **Repeat** until stopping criterion is satisfied;
```

```
06          Take pairs randomly from $\Phi$, cross them over and save in $\Phi'$
               until all members of $\Phi$ are crossed over;
07          Take fittest $P$ members from $\Phi \cup \Phi'$ to make the new $\Phi$;
08          Execute mutation and inversion operations;
09          Save $\lambda$ = best of $\{\lambda\} \cup \Phi$;
10      **Optimize** $\lambda$ and report the result;
```

4.2.1 Simulation, Results and Discussion

This algorithm was tested on set-C, D and E graphs from [64], where set-C consists of 500 nodes graphs, set-D of 1000 nodes graphs, and set-E of 2,500 nodes graphs. Edge costs are random integers from 1 to 10. The population size was fixed at 40, $p_{mut} = 0.005$ and $p_{inv} = 0.1$. When there is no improvement of the best solution over consecutive 50 generations, the algorithm stops.

Discussion: In most of the experiments, the algorithm could find the optimum solution about 80% of the time it was executed and the result was within 1% of the optimum solution in more than 90% of the trials. These percentages are a little less for the big graphs in set-E. The quality of result was better or at least as good as the best of the heuristic algorithms. Though different algorithms were run on different machines, the execution time of GA was comparable to heuristics. This work concludes beyond doubt that GA is a high performance, robust and efficient algorithm for solving $MStTG$ problem, when the domain knowledge of the problem is effectively used.

5 Solving $CMStTG$ Problem by Genetic Algorithm

The only work reported to use GA to solve $CMStTG$ is from Zhang and Leung in 1999 [36]. They defined a new orthogonal genetic algorithm (see also [66]), which they found to be more efficient and able to deliver better results, compared to traditional genetic algorithm.

5.1 Orthogonal Crossover and Orthogonal Genetic Algorithm

What is orthogonal array?

The origin of the idea is from a concept for designing experiments in efficient fashion [67]. Let us consider that for a given experiment there are k different factors involved, and each factor can take n different values in a particular (ascending/descending) order. For an exhaustive search for the best combination of different factors and their respective levels, we need to try n^k experiments. In orthogonal design [67], only a few orthogonal combinations corresponding to orthogonal array are to be tried, and it is proved that these selective combinations are good representatives for all the n^k possibilities.

For example, for a three factors two levels experiment, the four orthogonal arrays $L_4(2^3)$ are shown in Fig. 11, and listed in Table. 1. The four combinations in Table. 1 is a good representation of 2^3 possibilities. Location of these four points in the 3-dimensional (3 factors) space is shown by the solid circles in Fig. 11.

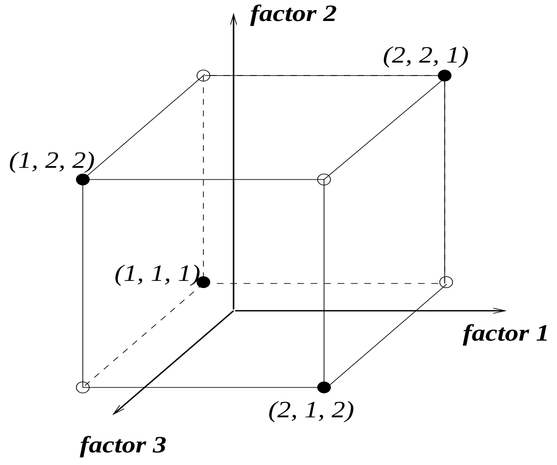

Figure 11: Orthogonality of the orthogonal array $L_4(2^3)$

Combination	Factor 1	Factor 2	Factor 3
1st.	level 1	level 1	level 1
2nd.	level 1	level 2	level 2
3rd.	level 2	level 1	level 2
4th.	level 2	level 2	level 1

Table 1: Orthogonal array $L_4(2^3)$ for three factors at two levels

For a four factor three level situation, the $L_9(3^4)$ orthogonal array is listed in Table.2. These array combinations, $L_4(2^3)$ and $L_9(3^4)$, is the basis in defining orthogonal crossover.

Combination	Factor 1	Factor 2	Factor 3	Factor 4
1st.	level 1	level 1	level 1	level 1
2nd.	level 1	level 2	level 2	level 2
3rd.	level 1	level 3	level 3	level 3
4th.	level 2	level 1	level 2	level 3
5th.	level 2	level 2	level 3	level 1
6th.	level 2	level 3	level 1	level 2
7th.	level 3	level 1	level 3	level 2
8th.	level 3	level 2	level 1	level 3
9th.	level 3	level 3	level 2	level 1

Table 2: Orthogonal array $L_9(3^4)$ for four factors and three levels

Chromosome encoding and decoding: The GA approaches discussed till now used binary bits in chromosomes to represent nodes of the graph. Here the chromosome consists of bit string of length l, where l is the number of edges in the graph. The i^{th} bit represents the i^{th} edge of the graph. If a particular bit is 1, the corresponding edge is selected for making the Steiner tree. Not all combinations of edges would form a tree connecting the source node to all destination nodes. To convert any chromosome to a valid Steiner tree, the subgraph induced by the selected edges is found. Then the spanning tree for the subgraph is created, and the unconnected nodes of D are connected by simple graph search method [68].

Orthogonal crossover based on orthogonal array: In genetic algorithm, the crossover operation is responsible for creating different combinations of the individual chromosomes to explore intermediate regions of the search space. But splitting chromosomes at any arbitrary point, and combining, may be wasteful and definitely computationally heavy. Thinking in the line of orthogonal array for designing experiments, Zhang and Leung [36] proposed this new orthogonal crossover, which they claimed and showed through experiments to be more effective way of exploring the search space, at least for this problem.

In conventional crossover, two chromosomes are combined. In orthogonal crossover it may be 2 or more. In general, if we take n levels and k factors and corresponding $L_m(n^k)$ orthogonal array, then we will be considering crossover among n chromosomes corresponding to n-levels, producing m number of offsprings. Every chromosome is divided into q number of genes $p(1)$ to $p(q)$. q is arbitrarily set, where $q \geq k$. As far as possible, the genes are set equal in length. An onto mapping from particular gene $p(i)$ to one of the k factors is decided arbitrarily. Now every part of the chromosome i.e., the genes, could be mapped to one of the k factors. Out of n parents and m offsprings, j number of members are selected to the next generation. In [36], this way of combining chromosomes is named as n - to - j *orthogonal crossover.*

The idea of orthogonal crossover is best explained by an example. Let us use $L_9(3^4)$ as shown in Table. 2 to define the different offsprings. As $n = 3$, the crossover here involves 3 parent chromosomes. Let us divide every chromosome in 5 genes, namely $p(1)$, $p(2)$, $p(3)$, $p(4)$ and $p(5)$. So, $q = 5$, $k = 4$ and $q \geq k$ is satisfied. For example, if the chromosome is of length 28 bits, then the first 6-bits make the $p(1)$ gene, the 7th to 12th bits form $p(2)$ etc., and the 5th gene is formed by the last 4-bits. This division is arbitrary and has no fixed rules. Let us denote the three parent chromosomes as follows:

$$\begin{aligned}
\mathbf{X_1} &= (\mathbf{x_{1,1}}, \mathbf{x_{1,2}}, \mathbf{x_{1,3}}, \mathbf{x_{1,4}}, \mathbf{x_{1,5}}) \\
\mathbf{X_2} &= (\mathbf{x_{2,1}}, \mathbf{x_{2,2}}, \mathbf{x_{2,3}}, \mathbf{x_{2,4}}, \mathbf{x_{2,5}}) \\
\mathbf{X_3} &= (\mathbf{x_{3,1}}, \mathbf{x_{3,2}}, \mathbf{x_{3,3}}, \mathbf{x_{3,4}}, \mathbf{x_{3,5}})
\end{aligned}$$

The three chromosomes $\mathbf{X_1}$, $\mathbf{X_2}$ and $\mathbf{X_3}$ are the three levels($n = 3$). Because the number of genes are 5, to be able to use compositions suggested in $L_9(3^4)$, where there are only 4-factors ($k = 4$), the 5 genes are mapped as follows:

$p(1) \rightarrow \mathit{Factor}\ 1,\ p(2) \rightarrow \mathit{Factor}\ 2,\ p(3) \rightarrow \mathit{Factor}\ 1,\ p(4) \rightarrow \mathit{Factor}\ 3,\ p(5) \rightarrow \mathit{Factor}\ 4$

In fact, this mapping could be done in many different ways. The offsprings will have the same number of genes as their parents by combining factors (genes) from different levels (chromosomes). According to the above gene to factor mapping, gene $p(1)$ and $p(3)$ both are to be chosen from the chromosome (level entry) appearing in the first column of Table. 2 i.e., *factor 1.* For example, because row 6 of $L_9(3^4)$ is (2 3 1 2), the sixth offspring will be from parents (2 3 2 1 2), and it will be constructed as follows:

$$\mathbf{O_6} = (\ \mathbf{p(1)\ from\ X_2},\ \mathbf{p(2)\ from\ X_3},\ \mathbf{p(3)\ from\ X_2},\ \mathbf{p(4)\ from\ X_1},\ \mathbf{p(5)\ from\ X_2})$$

The nine offsprings are:

$$\begin{aligned}
\mathbf{O_1} &= (\mathbf{x_{1,1}}, \mathbf{x_{1,2}}, \mathbf{x_{1,3}}, \mathbf{x_{1,4}}, \mathbf{x_{1,5}}) \\
\mathbf{O_2} &= (\mathbf{x_{1,1}}, \mathbf{x_{2,2}}, \mathbf{x_{1,3}}, \mathbf{x_{2,4}}, \mathbf{x_{2,5}}) \\
\mathbf{O_3} &= (\mathbf{x_{1,1}}, \mathbf{x_{3,2}}, \mathbf{x_{1,3}}, \mathbf{x_{3,4}}, \mathbf{x_{3,5}}) \\
\mathbf{O_4} &= (\mathbf{x_{2,1}}, \mathbf{x_{1,2}}, \mathbf{x_{2,3}}, \mathbf{x_{2,4}}, \mathbf{x_{3,5}}) \\
\mathbf{O_5} &= (\mathbf{x_{2,1}}, \mathbf{x_{2,2}}, \mathbf{x_{2,3}}, \mathbf{x_{3,4}}, \mathbf{x_{1,5}}) \\
\mathbf{O_6} &= (\mathbf{x_{2,1}}, \mathbf{x_{3,2}}, \mathbf{x_{2,3}}, \mathbf{x_{1,4}}, \mathbf{x_{2,5}}) \\
\mathbf{O_7} &= (\mathbf{x_{3,1}}, \mathbf{x_{1,2}}, \mathbf{x_{3,3}}, \mathbf{x_{3,4}}, \mathbf{x_{2,5}}) \\
\mathbf{O_8} &= (\mathbf{x_{3,1}}, \mathbf{x_{2,2}}, \mathbf{x_{3,3}}, \mathbf{x_{1,4}}, \mathbf{x_{3,5}}) \\
\mathbf{O_9} &= (\mathbf{x_{3,1}}, \mathbf{x_{3,2}}, \mathbf{x_{3,3}}, \mathbf{x_{2,4}}, \mathbf{x_{1,5}})
\end{aligned}$$

Some of the offsprings could be same as the parent. Here $\mathbf{O_1}$ is same as $\mathbf{X_1}$. $\mathbf{O_1}$ is therefore a valid solution of the problem. For other newly generated offsprings, which may represent invalid solutions of the problem, have to be checked and repaired to a valid solution as described earlier in the paragraph **Chromosome encoding and decoding**. Now the fitnesses of the repaired offsprings are calculated from their respective Steiner trees.

Fitness calculation and selection: Once the Steiner tree from a chromosome is found, its fitness F is evaluated. Here F has two terms f_1 and f_2. Term f_1 represents the total cost of the Steiner tree. Term f_2 is the sum of the degrees of violation of the delay constraint (Δ) by different paths from source s to destinations $v \in D$. Thus,

$$f_2 = \sum_{v \in D} max\ \{0,\ \langle \text{Path delay from } s \text{ to } v \rangle - \Delta\}$$

If a path delay is $\leq \Delta$, its contribution to f_2 is 0. Because it is related to the constraint, f_2 is more important. Between two chromosomes and their corresponding Steiner trees T' and T'' with fitnesses f_1', f_2' for T', and f_1'', f_2'' for T'', T' is better when,

1. $f_2' < f_2''$ i.e. the delay violation in T' is less, or
2. $f_2' = f_2''$ and $f_1' < f_1''$ i.e., when both have same delay violation (or both satisfy the constraint), the tree with less cost is a better fit.

Of course, in this way absolute value of fitness to any individual chromosome can not be assigned. But we can compare any two and rank them in order of fitnesses.

Next is the selection. Out of n parents m offsprings are generated. From these $(n+m)$ chromosomes, one possibility is to select the best j. Another possibility is to select j chromosomes from $(n+M)$ not deterministically, but with greater probability for better chromosomes by ranking and tournament selection [56]. The selection can even be from offsprings only. After selection of j members, mutation is performed by flipping a randomly selected bit (with a small probability) from a randomly selected chromosome. Repair is done when necessary.

Orthogonal Genetic Algorithm

We now explain the overall operation of the orthogonal GA. Suppose the population size is P. Since from n parents j offsprings are selected to the next generation, this orthogonal crossover and selection operation is performed P/j times, so that the population size remains constant over generations. The genetic algorithm steps are:

I **Initialization:** An initial population $\{\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_P}\}$ is created with random bits of 0s and 1s. They are repaired to make them valid solutions of the problem.

II **While** $\langle$ stopping criterion is not met $\rangle$ repeat the Steps III to V.

III **Do** P/j times n-to-j Orthogonal crossover. Randomly select n parents from the whole population and perform orthogonal crossover P/j times to get members of the next generation.

IV Each chromosome in the new generation undergoes mutation with a small probability p_m. Flip every bit in the chromosome selected for mutation with a small probability p_f. Thus the probability that a bit will be mutated is $p_m \times p_f$.

V Repair invalid chromosomes to make them valid solutions.

5.2 Simulation, Results and Discussion

The algorithm was tried on two sets of problems as follows:

1. Network with 50 nodes and 100 edges. $|D|$ is varied from 5 to 12 in steps of 1, making 8 experimental set-ups.

2. Network with 80 nodes and 200 edges. $|D|$ is varied from 5 to 12 in steps of 1, making another 8 experimental setups.

These problems were generated using method proposed in [69], which creates the network topology, the link costs, a set of source and destination nodes, and cost of different edges. It also tells the optimal steiner tree and its cost. The point that is lacking with respect to the above $CMStTG$ problem is that the delay for edges are not assigned. A random delay is assigned to every link, and then the delay to different destinations for the minimum cost Steiner tree is calculated. The delay constraint Δ is set equal to the maximum delay from source to different destinations in the minimum cost Steiner tree. Now the problem including the constraint is defined. At the same time, the optimum solution is also known.

Traditional genetic algorithm, orthogonal genetic algorithm with $L_4(2^3)$, and with $L_9(3^4)$ were simulated and tried on two sets of 8 problems each. Population size is fixed at 30 for the first set of 50-nodes problems, and 80 for the 80-nodes problems. The stopping criterion was the maximum number of generations, which was set to 400, 300 and 100 for traditional GA, $L_4(2^3)$ orthogonal GA, and $L_9(3^4)$ orthogonal GA respectively. This ensures that the required execution time for different versions of GA are similar. j was set to 2, and the best two of all the offsprings were copied to the next generation. $p_m = p_f = 0.1$ was set, such that every bit would be flipped for mutation with a probability of 0.01. The algorithm quality is judged by the percentage of runs for which the Steiner tree cost reaches within 20%, or 10%, or 5%, or 2% or 0% of the known optimal solution. The results show that orthogonal $L_9(3^4)$ performs much better than orthogonal $L_4(2^3)$, which again performs much better than traditional GA.

Discussion: By orthogonal crossover the speed gain is appreciable i.e. better solutions were created in less number of generations. Another important difference is in the selection method. In conventional GA, selection involves all the members of the population, and is a difficult issue for distributed implementation. Here the selection is done only on the set of chromosomes involved in the orthogonal crossover and offsprings generated from them. Therefore, distributed implementation of crossover, mutation and selection is natural and easy.

Orthogonal crossover is a kind of multiple parents multiple point crossover. But, by restricting the crossover points at longer intervals (of genes), the diversity of the offspring chromosomes are restricted. In orthogonal design [67] the factors are well defined. But here the slicing of the chromosomes to different genes are arbitrary. The logic of the orthogonal design does not hold good in true sense. On the other hand to improve diversity, if the gene length is shortened and larger values for k are tried, the computational load will be too heavy. Thus it is obvious that $L_9(3^4)$ gives better result than $L_4(2^3)$, but at the cost of more computation load, when equal number of generations are executed.

6 Solving *MEStT* Problem by Genetic Algorithm

The only work [44] proposed to solve *MEStT* was due to Hesser, Manner and Stucky in 1989. The approach is traditional with some efficiency gained through use of problem domain knowledge. They compared their algorithm with simulated annealing and average distance heuristic (ADH) [15].

6.1 Chromosome Structure and the Genetic Algorithm

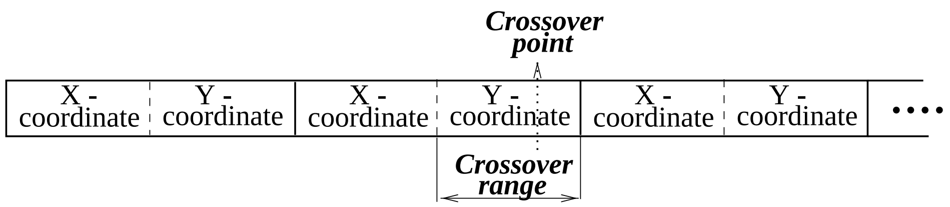

Figure 12: Chromosome structure

The chromosome structure is shown in Fig. 12. It consists of (x, y) coordinate values of a fixed number of Steiner points. Using the knowledge that the number of Steiner points could at most be $(|D| - 2)$, the length is fixed to $(|D| - 2)$. The range of x - and y - coordinates are determined from the span of site points. Number of bits to represent a coordinate value, along with the span of the original site points, determine the precision of the resulting Steiner node locations. A longer bit string would deliver results with higher precision, but at the cost of more computation time. In [44], 6 bits were used. Therefore the precision of the result is $range/2^6$.

Fitness calculation: An individual chromosome represents the positions of a set of $(|D| - 2)$ Steiner points. For all members of the population of size P, the corresponding Steiner trees are constructed, and their costs calculated. The fitness F_i of an individual, say i^{th} member, is

$$F_i = \text{max. of } \{\mathrm{C_1, C_2, \ldots, C_P}\} - \mathrm{C_i} \tag{1}$$

C_i is the cost of the tree constructed by decoding the i_{th} chromosome. The worst chromosome will have 0 fitness. Lower tree cost will give higher fitness.

Crossover and mutation: Every chromosome is tried for crossover with a probability $p_c/2$. If the chromosome is selected for crossover, its partner is selected from the rest of the population with uniform probability. As shown in Fig. 12, crossover is one point, selected uniformly over the whole length of the chromosome, and its range of execution is over a single x- or y-coordinate value. The probability of crossover p_c is fixed at 0.5.

The mutation operation flips a bit with a fixed probability of 0.01, and is tried on every bit of every chromosome.

Selection: Suppose the current generation is designated by g and the next generation by $(g+1)$. If the i^{th} member of the population $\phi(i)$ appears with frequency $h_i(g)$ in the current generation, its fitness being F_i, and the average fitness of the whole population is $\langle F \rangle$, then in the subsequent generation the same member will appear with a frequency $h_i(g+1)$ where,

$$h_i(g+1) = \frac{F_i}{\langle F \rangle} \cdot h_i(g)$$

This selection is called roulette selection [55].

Population size: Hesser et al. proposed [44] a rule of thumb to determine optimum population size, in line with the idea proposed by Goldberg in [70]. Goldberg's report says that, if the number of binary bits in the chromosome be B, the population size should be about $1.65 \times 2^{0.21 \cdot B}$. For a 25 Steiner points problem it would be too large to implement. The authors calculated and plotted the number of different Steiner trees possible for various population sizes. They found that when the population size ranges between 30 to 110, there is the largest increase in the number of different structures. Following similar arguments as in [70], they found that a population size between 30 to 110 would work well. They fixed it at 50.

Heuristics: The following problem domain knowledge is used to improve the efficiency of the GA: (1) The number of Steiner points were limited to $(|D|-2)$. (2) The fact that the Steiner points in $MEStT$ are of degree 3 is used. (3) The Steiner nodes with degree 2 are removed, and the two nodes on both sides are directly joined. Steiner nodes of degree > 3 are left as it is. (4) Finally a local optimization on the solution is done by shifting every Steiner point locally, so that its edges are arranged symmetrically, where the shift leads to a shorter tree [38]. They have found that by applying these heuristics, the GA convergence is speeded up by 20 times. This, of course, means that otherwise also GA could find the same quality of results, but would take much longer time and many more generations.

6.2 Simulation, Results and Discussion

Experiments were done with a grid network of 5×5 nodes. They varied the probability of crossover p_c and mutation p_m. As is obvious, it was reported that very small values for these probabilities prolong the convergence, and on the other hand large values would destroy good chromosomes at the later stage. They also reported that roulette selection works better than ranking. The selection pressure could be controlled by changing the offset value from the maximum tree cost of all members (as in Eq. (1)), to some higher value. They also found experimentally that the population size between 30 to 100 members is the best choice, as predicted. It should be noted here that instead of simple binary coding, a *Gray code* interpretation of the bitstring would be more efficient. The reasons are explained in [71].

7 Solving *MERStT* Problem by Genetic Algorithm

The only GA based work [53] proposed to solve *MERStT* was due to Julstrom in 1993. The design of the chromosome for his work is interesting. The basic idea is from Cayley's counting of different spanning trees and Prüfer's encoding of them [72](pp.140-145). Though the algorithm does not ensure global optimal solution, even not as good as many heuristic approaches [51], [73], [52], the encoding of chromosome is interesting to be noted.

7.1 Chromosome Structure and the Genetic Algorithm

Basic Idea

The basic idea of [53] is illustrated in Fig. 13. In 1875, Arthur Cayley counted the number of spanning trees for a n-node complete graph as $n^{(n-2)}$. One of the most elegant encoding algorithm for the different spanning trees was due to H. Prüfer, and is called Prüfer sequence. The encoding algorithm is as follows:

Input: A spanning Tree with vertex-labeling.

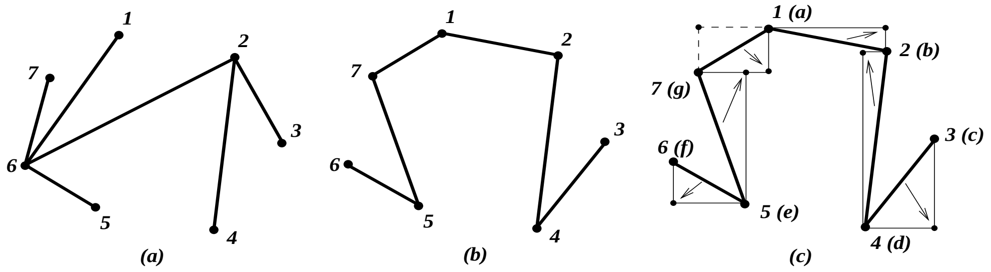

Figure 13: Prüfer's encoding of Spanning tree

```
Output: Prüfer sequence of length $(n-2)$.
Initialize: Prüfer sequence to NULL.
    for $i = 1$ to $(n-2)$
        Let $v$ be degree 1 vertex with smallest label.
        Let $s_i$ be the label of the only neighbor of $v$.
        Delete $v$ and its edge from Tree. Add $s_i$ to Prüfer sequence.
    Return Prüfer sequence $\langle s_1, s_2, \ldots, s_{(n-2)} \rangle$.
```

If we apply the above algorithm on the spanning tree of Fig. 13(a), we get a sequence $\langle 6,2,2,6,6 \rangle$ and for the spanning tree in Fig. 13(b) the sequence is $\langle 4,2,1,7,5 \rangle$. The number of times a node appears in the sequence is one less than its degree, as expected. The $(n-2)$ positions in the sequence could be occupied by any of the n-labels of n nodes, where every such sequence represent a different spanning tree. Thus the number of different spanning trees is $n^{(n-2)}$.

Following is the decoding algorithm from Prüfer sequence to tree.

```
Input: The Prüfer sequence $Q$ of node labels of length $(n-2)$.
Output: The $n$-vertex tree.
Initialize: A list $L$ of $n$ node labels in ascending order.
Initialize: Forest $F$ of $n$ isolated vertices, labeled 1 to $n$.
    for $i = 1$ to $(n-2)$
        Let $k$ be the smallest entry such that $k \in L$, but $k \notin Q$.
        Let $j$ be the first member in $Q$
        An edge joining $k$ to $j$ is added to $F$.
        Remove $k$ from $L$ and $j$ from $Q$.
```

Two ramaining nodes of L are joined to get the final tree.

When the Prüfer sequence is $Q = \langle 4\ 2\ 1\ 7\ 5 \rangle$, using the decoding algorithm we could construct the tree of Fig. 13(b) by adding edges in the following sequence:

$$3-4,\quad 4-2,\quad 2-1,\quad 1-7,\quad 6-5,\quad 5-7$$

Now, to construct a rectilinear Steiner tree, a direct link between any two nodes has to be converted to horizontal and vertical segments for which the direct link is a diagonal. For every edge this could be done in two ways, as drawn by thin solid and dashed lines for the edge between nodes 1 and 7 in Fig. 13(c). By convention, let us consider that when the Steiner point is on the left of the link it is represented by a 0 (the dashed line in Fig. 13(c)), and when on right it is a 1 (the thin solid line in Fig. 13(c)). For every spanning tree, as we have $(n-1)$ links, $2^{(n-1)}$ such rectilinear Steiner trees are possible. Thus in total $2^{(n-1)} \times n^{(n-2)}$ different rectilinear spanning trees are possible for a n-node graph. In Julstrom work, he tried to find the minimum of all such rectilinear Steiner trees. Though it may not be the optimum $MERStT$, it is a good approximation [49]. Such rectilinear trees are encoded in chromosomes as explained below.

Chromosome structure: The chromosome consists of two parts

1. $(n-2)$ symbols from n possible node labels representing a Prüfer sequence and thus a particular spanning tree.

2. $(n-1)$ number of 0s and 1s represting right or left choice of rectilinear representation for the $(n-1)$ links of the spanning tree.

To distinguish between node labels' symbols from 0/1, we label the nodes with alphabets, shown in brackets, in Fig. 13(c). Node 1 becomes a, node 2 b and so on. The chromosome corresponding to the rectilinear Steiner tree in Fig. 13(c) (thin solid lines) is:

$$\langle 1\ d\ 0\ b\ 1\ a\ 1\ g\ 0\ e\ 1 \rangle$$

While decoding the spanning tree defined by $\langle d\ b\ a\ g\ e \rangle$, edges are added in sequence as $c-d,\ d-b,\ b-a,\ a-g,\ f-e,\ e-g$. The first 1 in the chromosome denotes that the Steiner point is on the right side of the edge

$c - d$. Similarly, the next 0 denotes that the Steiner point is on the left of the edge $d - b$, and so on.

Genetic Operations

Crossover: By crossover operations we join parts of two solutions in a meaningful way to form a new solution. Here the genotype, i.e. the chromosome representation, and phenotype, i.e. the actual realization, are quite different. If conventional crossover is done, though the resulting offsprings will still be valid solution of the problem, they may be completely different from their parents. They will not combine parts of their parents, which is the basic idea of search in GA (building block hypothesis). Crossover will result in something like mutation, jumping to a new location far away from both parents. So, a new crossover method, which enable to retain parts from the parent trees, and generate a single offspring, is defined. It is named *spanning tree crossover.*

Spanning tree crossover is not done on genotypes but on phenotypes, i.e. the actual trees. The idea is to retain common parts from both parents as much as possible. If two parents have common edges, and the corresponding Steiner points are on the same side, those rectilinear edges are copied to the offspring. If the steiner points are on different sides, any one of the two is chosen randomly. The rest of the offspring tree is formed by selecting randomly from the remaining rectilinear segments of the parents. Any cycles created should be broken. Thus, the crossover operations are not on chromosomes but on substructures of trees, and computationally heavy.

Mutation: Mutation operation is like in traditional GA. It modifies the symbol at any position with a low probability of 0.01. For the 0/1 entries, it flips. For the node labels, mutation could change a label to any of the rest $(n - 1)$ possibilities randomly with uniform probability.

Overall execution: The initial population is created by random combination of the symbols. Crossover and mutation are performed on the population with probabilities of 0.4 and 0.01 respectively. As explained, both crossover and mutation produce a single new chromosome. After crossover and mutation, to keep the population size constant,

1. Duplicated members are deleted.

2. The tree lengths for different members are calculated. Chromosomes corresponding to longer trees are deleted with higher probabilities.

This creation of new chromosomes by crossover and mutation, and shedding of bad members to maintain constant population size are repeated. Other more deterministic shcemes for deletion, like deleting only worst members, were also tried.

7.2 Simulation, Results and Discussion

The experiments were done with a set of 32 randomly set points. The author tried different variations of population size and different crossover operations. Every experiments were performed such that 50,000 different chromosomes were created during the whole execution of the algorithm, and thus took more or less the same computation time, even though the population size was different. Traditional two-point crossover, one-point crossover and the proposed spanning tree crossover were tried. *Spanning tree crossover* could produce better (about 10%) results, whereas one-point crossover was only slightly better than two-point crossover. Population size of 40 produced the best Steiner tree, when different population sizes of 30, 40, 50, 60, 70, 80, 100, 150, 250 and 500 were tried. Of course, as mentioned, the total number of different chromosomes created during the whole period of execution was fixed at 50,000.

Hwang in [49] proved that no rectilinear Steiner tree could be shorter than 2/3 rd the distance of a minimum rectilinear spanning tree (*MRSpT*) on the same points. It is easy to find the minimum spanning tree using Prim's [4], or Kruskal's [5] algorithm. The distance of minimum rectilinear spanning tree is obtained, when the distances of edges are not measured as $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, but as $|(x_i - x_j)| + |(y_i - y_j)|$. Here (x_i, y_i) and (x_j, y_j) are the rectilinear coordinates of the two connected nodes. Thus it is easy to calculate a lower bound for the *MERStT*, which is 67% of the *MRSpT*. Most of the hueristics could achieve on an average around 92% of the *MRSpT* for most of the problems tried. But the best *MERStT* for the 32-point problem, found by this algorithm, is only 99% of *MRSpT*. The average is 104.5%. It is seen that for smaller problems it works better, though the overall performance is worse than all the heuristics. On an average, it is even worse than *MRSpT*. As *MRSpT* is easy to find, the proposed algorithm is of little practical use.

8 Discussion

In this article, we have covered all important genetic algorithm based approaches to solve different Steiner tree problems. Optimum Steiner tree is a constraint optimization problem. Straightforward use of genetic operations (crossover and mutation) may generate invalid solutions that does not comply with the required constraints. If invalid solutions are allowed in the chromosome, we invite two main problems: (1) the search space becomes too large to be able to find optimum solution, (2) it is difficult to assign proper fitnesses (i.e. penalties) to the invalid solutions. In Kapsalis et al. [24] work, one weak point is the fitness calculation. There constraint violation part (number of tree components) and optimization part (tree costs) were linearly added. In general, ranking the chromosomes, where constraint part is given more priority than optimization works better, as done by Zhang and Leung in [36]. Yet, better approach is to repair invalid solutions to their nearest valid solutions. This requires execution of some heuristic methods that uses problem domain knowledge, and therefore computationally costly. Heuristic algorithms using problem domain knowledge were extensively used by Esbensen in [25]. The quality of result was better or at least as good as the best heuristic algorithm, and the execution time was comparable. This work concludes without doubt that GA is a high performance, robust, and efficient algorithm for solving $MStTG$ problem, when the domain knowledge of the problem is effectively used.

Another important aspect of GA is to allow proper exploration of the search space during early generations, and quickly settle down to the optimum at later generations. The proposal of orthogonal crossover in [36] is to expedite this exploration. The crossover is designed so as to ensure that the whole problem domain is explored in a regular manner. The idea is adopted from proper designing of experiments [67]. The questionable point is that, by using longer genes the exploration is restricted to some extent. Moreover, the cut points of the chromosomes to different genes are arbitrary - not well defined as in case of real experiments discussed in [67]. It is not clear why the logic of orthogonal experiment design would work well here too. For different optimization problem, orthogonal GA may or may not work well, depending on the nature of the problem domain. The success of the idea, for the Steiner tree problem, is shown through experiments, where the speed gain using orthogonal crossover is appreciable. Another added advantage is the way the selection is done. Only the set of chromosomes involved in the orthogonal crossover, and offsprings generated from them, are considered for selection. Therefore, distributed implementation of crossover, mutation and selection is natural and easy. Thus, the increased computational cost for orthogonal crossover could be easily taken care of by a distributed implementation.

Other important aspect of GA is how the phenotype (the solution of the problem in the original form) is encoded into genotype i.e. the chromosome. When the original solution is in *pseudoboolean*, as in [24] and [25], where the solution is either to select a node or not and naturally translates to 1 or 0, a simple boolean encoding is sufficient. In [44], the chromosome represents a set of real decimal numbers. They converted these decimal coordinates to base - 2 binary numbers. Presently, it is common practice to use *Gray code* interpretation of the bitstring segments, an idea first indicated by Bethke in his Ph.D. thesis in 1981. It would have been more advantageous for their work too. The reason is that, in *Gray code* two consecutive numbers differ by only one bit [71].

Another important aspect of GA is the scaling of the fitness values, by which one can control the exploration and selection pressure. In [44], it is done simply by adding some positive number to the right side of Eq. (1). This would squeeze the range of fitness variation (as percentage of maximum fitness), and help in better exploration of the search space.

Julstrom's work [53] is a novel example of the varieties of ways a genotype could be designed. The algorithm basically searches derivatives from the minimum spanning tree, and the quality of result is poor. Though computationally efficient, on an average the result is even poorer than $MRSpT$, for which there is very simple and fast heuristics. So the usefulness of [53] is questionable, yet it is interesting to note.

We have covered most of the important genetic algorithm based works to solve Steiner tree problem. Though all of these works are interesting in their own right, they are yet to reach the efficiency of the heuristic algorithms for large problems, and there is ground for further improvements. Only the work by Esbensen on $MStTG$ performs well even for very large graphs. It should be mentioned that it borrowed ideas from heuristic algorithms and exploited various problem domain knowledge. In other works too, it has been shown that the use of problem domain knowledge improves efficiency. Thus, the general conclusion is that, a hybrid approach would be more efficient and could compete with heuristic algorithms.

References

[1] M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing Steiner minimal trees," *SIAM Journal of Applied Mathematics*, Vol. 34, pp. 477-495, 1977.

[2] 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.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1989.

[4] R. C. Prim, "Shortest connection networks and some generalizations,"*Bell System Technical Journal*, vol. 36, 1957.

[5] J. B. Kruskal Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings AMS*, vol. 7, no. 1, 1956.

[6] A. Levin, "Algorithm for the shortest connection of a group of graph vertices," *Soviet Math. Dokladi*, vol. 12, pp. 1477 - 1481, 1971.

[7] S. L. Hakimi, "Steiner problems in graphs and its implications," *Networks*, vol. 1, pp. 113 - 133, 1971.

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

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

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

[11] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for steiner trees," *Acta Informatica*, 15:141–145, 1981.

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

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

[14] Y. F. Wu, P. Widmayer, C. K. Wong, "A Faster Approximation Algorithm for the Steiner Problem in Graphs," *Acta Informatica*, Vol. 23, pp. 223-229, 1986.

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

[16] A. Balakrishnan and N. R. Patel, "Problem reduction method and a tree generation algorithm for the Steiner network problem," *Networks*, Vol. 17, pp. 65 - 85, 1987.

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

[18] J. E. Beasley, "An SST-Based Algorithm for the Steiner Problem in Graphs," *Networks*, Vol. 19, pp. 1-16, 1989.

[19] S. Voss, "Steiner's problem in Graphs: Heuristic methods," *Discrete Applied Math.*, Vol. 40, pp. 45 - 72, 1992.

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

[21] S. Chopra, E. 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.

[22] A. Z. Zelikovsky, "A faster approximation algorithm for the Steiner tree problem in graphs," *Information Processing Letters*, Vol. 46, pp. 79-83, 1993.

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

[24] 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.

[25] H. Esbensen, P. Mazumder, "A Genetic Algorithm for the Steiner Problem in a Graph," Proc. of *The European Design and Test Conference*, pp. 402-406, 1994.

[26] C. Pornavalai, G. Chakraborty, and N. Shiratori, "Neural network for optimal steiner tree computation," *Neural Processing Letters*, Vol. 3, No. 3, pp. 139-149, August 1996.

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

[28] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, Vol. 22, pp. 55 - 89, 1992.

[29] V. P. Kompella, J. C. Pasquale, and G. C. Plyzos, "Multicast routing problems," *IEEE/ACM Trans. on Networking*, Vol. 1, No. 3, pp. 286-292, 1993.

[30] V. Kompella. *Multicast Routing Algorithms for Multimedia Traffic.* PhD thesis, University of California, Sandiego, 1993.

[31] R. Widyono, "The design and evaluation of routing algorithms for real-time channels," *Technical Report TR-94-024*, ICSI, University of California, Berkeley, June 1994.

[32] Q. Zhu, M. Parsa and J. J. Garcia-Luna-Aceves, "A source based algorithm for delay-constrained minimum-cost multicasting," In *Proceedings of IEEE INFOCOM*, Boston, M.A. pp. 377-385, 1995.

[33] C. Pornavalai, G. Chakraborty, and N. Shiratori, "A neural network approach to multicast routing in real-time communication networks," In *1995 Int. Conf. on Network Protocols (ICNP'95)*, Tokyo Japan, November 1995.

[34] C. Pornavalai, G. Chakraborty, and N. Shiratori, "Neural networks for solving constrained steiner tree problem," In *1995 IEEE Int. Conf. on Neural Networks (ICNN'95)*, Perth Australia, November 1995.

[35] Yee Leung, Guo Li, and Zong-Ben Zu, "A genetic algorithm for the multiple destination routing problems," *IEEE Trans. on Evolutionary Computation*, Vol. 2, No. 4, pp. 150-161, 1998.

[36] Q. Zhang and Y. W. Leung, "An orthogonal genetic algorithm for multimedia multicast routing," *IEEE Trans. on Evolutionary Computation*, Vol. 3, No. 1, pp. 53-61, 1999.

[37] D. Chakraborty, Goutam Chakraborty, P. Chotipat, N. Shiratori, “Optimal Routing for Dynamic Multipoint Connection”, *European Transactions on Telecommunication*, Special issue Architectures, Protocols and Quality of Service for the Internet of the Future, Vol.10, No.2, pp. 183-190, March-April 1999.

[38] E. N. Gilbert, H. O. Pollak, “Steiner minimal trees,” *SIAM Journal of Applied Mathematics*, Vol. 16, pp. 1-29, 1986.

[39] P. Winter, “An algorithm for the Steiner trees problem in the Euclidean plane,” *Networks*, Vol. 15, pp. 323-345, 1985.

[40] E. J. Cockayne, D. E. Hewgill, “Exact computation of Steiner trees in the plane,” *Information Processing Letters*, Vol. 22, pp. 151-156, 1986.

[41] E. J. Cockayne, D. E. Hewgill, “Improved computation of plane Steiner minimum trees,” *Algorithmica*, Vol. 7, pp. 219-229, 1992.

[42] J. E. Beasley, “A heuristic for the Euclidean and rectilinear Steiner problems,” *European Journal of Operation Research*, Vol. 58, pp. 284-292, 1992.

[43] M. Lundy, “Applications of the annealing algorithm to combinatorial problems in statics,” *Biometrika*, Vol. 72, pp. 191-198, 1985.

[44] J. Hesser, R. Manner, and O. Stucky, “Optimization of Steiner trees using genetic algorithms,” Proceedings of *the 3th International Conference on Genetic Algorithm*, pp. 231-236, 1989.

[45] E. N. Gilbert, “Minimum cost communication networks,” *Bell Systems Technical Journal*, Vol. 9, pp. 2209-2227, 1967.

[46] 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.

[47] A. V. Aho, M. R. Garey, and F. K. Hwang, “Rectilinear Steiner trees: Efficient special-case algorithms,” *Networks*, Vol. 7, pp. 37-58, 1977.

[48] P. K. Agarwal and M. T. Shing, “Algorithms for special cases of rectilinear Steiner trees: I. Points on the boundary of a rectilinear rectangle,” *Networks*, Vol. 20, pp. 453-485, 1990.

[49] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM Journal of Applied Mathematics*, Vol. 30, No. 1, pp. 104-114, 1976.

[50] J. H. Lee, N. K. Bose, and F. K. Hwang, "Use of Steiner's problem in suboptimal routing in rectilinear metric," *IEEE Trans. on Circuits and Systems*, Vol. CAS-23, pp. 470-476, 1976.

[51] F. K. Hwang, "An O(n log n) algorithm for suboptimal rectilinear Steiner trees," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, No. 1, pp. 75-77, 1979.

[52] J. M. Ho, G. Vijayan, and C. K. Wong, "New algorithms for the rectilinear Steiner tree problem," *IEEE Trans. on Computer-Aided Design*, Vol. 9(2), pp. 185-193, 1990.

[53] B. A. Julstrom, "A Genetic Algorithm for the Rectilinear Steiner Problem," Proceedings of *the 5th International Conference on Genetic Algorithms*, pp. 474-480, 1993.

[54] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI., 1975.

[55] David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[56] Zbigniew Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag, 1995.

[57] Edited by T. Back, D. B. Fogel and Z. Michalewicz, *Evolutionary Computation 2*, Institute of Physics publishing. 2000.

[58] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Systems, Man and Cybernetics,* vol. SMC-16, No. 1, pp.122-128, Jan./Feb. 1986.

[59] M. Srinivas and L. M. Patnaik, "Adaptive Probabilities of crossover and mutation in Genetic algorithms," *IEEE Trans. Systems, Man and Cybernetics,* vol. SMC-24, No. 4, pp.656-666, April 1994.

[60] Goutam Chakraborty, and Kayo Hoshi, "Rank Based Crossover - A new technique to improve the speed and quality of convergence in GA", Proceedings of the 1999 Congress on Evolutionary Computation, Washington, USA, pp.1595-1602, July 6-9, 1999.

[61] H. Esbensen, "A Macro-Cell Global Router Based on Two Genetic Algorithms," Proc. of *The European Design Automation Conference*, pp. 428-433, 1994.

[62] H. Esbensen, "Finding (near-)optimal Steiner trees in large graphs," Proceedings of *the 6th International Conference on Genetic Algorithm*, pp. 485-491, 1995.

[63] H. Esbensen, "Computing (near-)optimal solutions to the Steiner problem in a graph using a genetic algorithm," *Networks* Vol. 26, No.4, pp. 173-185, 1995.

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

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

[66] Y. W. Leung, and Y. Wang, "An Orthogonal Genetic Algorithm with Quantization for Global Numerical Optimization," *IEEE Trans. on Evolutionary Computation*, Vol. 5, No. 1, pp. 41-53, 2001.

[67] D. C. Montgomery, *Design and Analysis of Experiments*, 3rd. edition, New York, Wiley, 1991.

[68] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and Complexity*, Englewood cliffs, NJ, Prentice-Hall, 1982.

[69] B. N. Khoury, P. M. Pardalos, and D. Z. Du, "A test problem generator for the Steiner problem in graphs," *ACM Transactions on Math. Soft.*, Vol. 19, No. 4, pp. 509-522, 1993.

[70] D. E. Goldberg, "Optimal initial population size for binary coded genetic algorithms," TGCA Report No. 85001, University of Alabama, 1985.

[71] T. Bäck,, *Evolutionary algorithms in theory and practice*, Oxford University Press. 1996.

[72] J. Gross, J. Yellen, *Graph theory and its applications*, CRC Press. 1999.

[73] J. M. Smith, D. T. Lee, and J. S. Liebman, "An O(n log n) heuristic algorithm for the rectilinear Steiner minimal tree problem," *Engineering Optimization*, Vol. 4, pp. 179-192, 1980.