Efficient use of Genetic Algorithms for the Minimal Steiner Tree and Arborescence Problems with applications to VLSI Physical Design

Mark Rabkin
Design Technology – Process Verification and Physical Design
Intel Corporation
Santa Clara, California 95054
mark.rabkin@intel.com

Abstract: Given an undirected graph **G** = (**V**, **E**), a set of algorithms exist that will produce a new graph **G'** = (**V'**, **E'**), such that **V** ⊆ **V'** and **E** ⊆ **E'**, all nodes *v'* ∈ **V'** are strongly connected by the edges of **E'**, and that the length of edges *e'* ∈ **E'** is minimized according to constraints given by the variations of the problem. This set of algorithms includes the minimum rectilinear shortest path Steiner arborescence (MRSPSA) problem and the rectilinear minimum Steiner tree (RMST) problem. Applications of these algorithms allow timing, current, and various other analyses to be performed on partially complete layout for a design, in order to estimate VLSI design progress.; these algorithms have been heavily studied in literature. This paper proposes a genetic algorithm with direct applications to these two minimization algorithms. Experiments indicate that using the genetic algorithm for these two problems produces nearly optimal results, with times on the same order of magnitude as existing "conventional" algorithms for the RMST Problem.

I. Introduction and Overview

Given an undirected graph G = (V, E), a set of terminals to be connected N ⊆ V, the minimum rectilinear Steiner tree (RMST) is an unrooted Steiner tree spanning all the terminals in N such that the sum of edge weights in the tree is minimized [7], and the edge weights are equal to the rectilinear distances between their two vertices. This is an NP-Complete problem, as shown by Hwang and others in 1976.

Given an undirected graph undirected graph G = (V, E), a set of terminals (sinks) N ⊆ V, and a unique root (source) node *r* ∈ V, a minimum shortest path Steiner arborescence (MSPSA) is a Steiner tree rooted at *r* spanning all the terminals in N such that every source-to-sink path is a shortest path in G, and the weights of the arborescence is minimized [4]. In this paper, for this and all the other algorithms, we are discussing the rectilinear (Manhattan) version of the problem (MRSPSA). Cong *et al.* discuss various types of exact algorithms for this MRSPSA problem, including dynamic programming, integer programming, branch-and-bound enumeration, and point out that the MRSPSA problem is also NP-complete. They also cover several heuristic algorithms derived from Rao *et al.* [8]. We are interested in the RSA/G and k-IA/G heuristics, because they produce the best performance/accuracy tradeoff as demonstrated in [4]. It is important to note that while the RMST problem minimizes the total length of the tree (each edge weight is counted once); in the MRSPSA problem, each path between *r* and all the terminals *n* ∈ N, *n* ≠*r* is minimized, and thus the weight of the tree in MRSPSA is given by adding the total length (weight) of each such path (each edge could be counted multiple times).

In the scope of this paper, we are only interested in the subset of input graphs G that are "rectilinear-complete". This means that the points in V have assigned plane coordinates, and that the weight of an edge between any two points in V is equal to the rectilinear distance between those two points. Furthermore, any two points in V can be connected to each other (the graph is free of "unusable blocks" or "obstructions"). We can assume this without loss of applicability, because (1) at the beginning stages of VLSI design no such obstructions are present, and (2) the edges produced can later have physical layers assigned to them so that they do not conflict with the objects they pass above or beneath. Given this assumption, the input to the algorithm can consist simply of the set N for the MRST problem, and the pair I = (N, *r*), for the MRSPSA problem. The equivalent graph G = (V, E) can be generated from N by the procedure described in Section III of this paper.

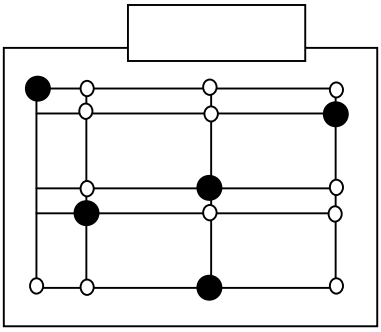
As described above, these two algorithms can produce graphs that solve two common VLSI design problems: (1) connecting a number of points in layout together using the minimum amount of wiring (such as for the power and ground nets in IC's), and (2) providing wiring to send a signal from a given source (or driver, or root) to one or more receivers (or sinks), with the minimum delay (and thus, usually, minimum length) between the driver and the receivers. This allows the designers, in the intermediate steps of VLSI Physical Design, to perform timing or current-limit analysis on missing or partially complete layout for a design, in order to estimate progress and to test and validate the results of previous steps such as block and cell placement.

Such analysis involves knowing the resistances, capacitances, and inductances of the full signal path, and thus the layout must be completed by automated means. Since these analyses are usually done (in the preliminary step) on a signal-by-signal basis, a full VLSI circuit is usually split into thousands of signal nets, which each signal net becoming a separate "completion" problem. The algorithms used to complete the layout net are very flexible – indeed, their goal is to produce a strongly connected net that would most resemble a net completed by an expert human design engineer. Thus, it is not important that the results of this algorithm are exact; it is more important that they run quickly, efficiently, and produce results within reasonable tolerances, which in some cases are up to 5-10% error.

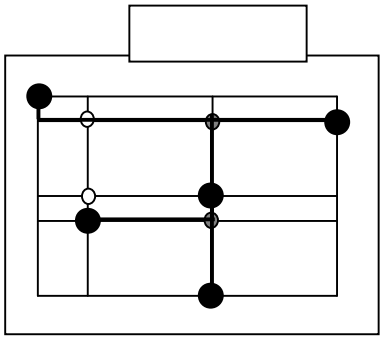
The size of the problems expected is greatly variable; the vast majority of nets encountered in real test cases (90%) is less than 30 input terminals in size ($|N| < 30$). However, nets of size $300 \leq |N| \leq 500$ are very common, and some extreme cases exist in which $|N|$ can reach 2000. The algorithms currently in use for RMST and MRSPSA (non-genetic) handle sizes of up to 400 fairly quickly, as is discussed in Section V.

III. Candidate Steiner Point Generator

As discussed above, our algorithms will take in only a set of input terminals N, and a root *r*, if required. Hanan [7] showed that an RMST spanning these points N must lie on a complete rectangular grid, where each input point adds a row and a column given its X and Y coordinate. In Figure 1, several Hanan points and the Hanan Grid induced by them are shown. It is easy to see that in the worst case, there are $O(N^2)$ such vertices. In Figure 1 above, twelve such vertices are created for the five input vertices.



Furthermore, Mandoi, Vazirani, and Ganley [10] provide a vertex reduction algorithm that reduces the number of vertices in the grid to O(N) vertices without any impact on the length of a minimum RMST or MRSPSA. Figure 2 shows the arrangement from Figure 1 with all but five of the initial twelve Hanan vertices removed according to the reduction algorithm. Figure 2 also shows the RMST that results from this arrangement.



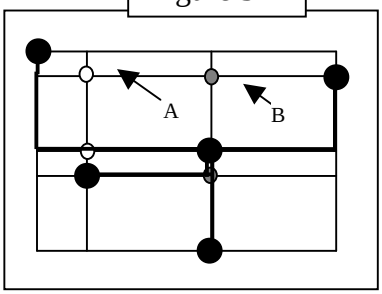
Note that only two candidate Hanan points (the gray circles in Figure 2) are located at an *intersection* of two black-outlined edges in the final RMST – in other words, their degree is greater than two. All such Hanan Grid points that appear in the final output RMST, or MRSPSA, with a degree greater than two will be called **Steiner Points**. The rest of the Hanan Grid points that remain after the vertex reduction algorithm (the white points in Figure 2 as an example) are called **Candidate Points**.

It can be seen even from the simple example above that adding certain Steiner points induces savings in the graphs because they allow groups of points to "share" paths to other points. Consider the example of Figure 3, which shows a simple Minimum Spanning Tree on the same graph as above. Notice the unnecessary duplication of vertical edges that is necessary to connect the top two terminals, as opposed to Figure 2. Also notice the slight duplicate edge just below the center terminal. Adding the bottom Steiner point eliminates this duplication; adding the top one consolidates the two vertical pieces into one central one, combining for large savings in overall graph length. Note that savings of close-proximity Steiner points tend to interfere with each other, however: adding point

A would also produce some savings, but these savings are nullified if point B is also added (point B introduces some savings of its own).

Salowe and Warme [11] showed that an RMST of the input terminals N that happens to uses a set S of Steiner Points chosen from the Hanan Candidates will be **identical** to the simple minimum spanning tree (MST), as found in any computer science textbook, of the vertex set V = N + S. Since the MST is an efficient algorithm, it remains that the difficulty (the NP-Completeness) of the RMST problem lies in the selection of this optimal Steiner point set S from the set of Hanan candidate points. The MST algorithm over V points can be found in O(E + V lg V) time [9].

Figure 3



In similar fashion, Cong *et al*. demonstrate an algorithm called RSA/G that will efficiently generate the MRSPSA for a terminal set N and root $r \in N$ given the optimal Steiner point set S [4]. The RSA/G algorithm runs in worst-case O(E + V lg V) time (identical to the MST algorithm above).

Thus, if only the set S of optimal Steiner points could be found, both the RMST and the MRSPSA become easy problems that can be solved very efficiently. Note, of course, that it is not the same S of Steiner points that is optimal for both minimization problems as the constraints are different in each case.

I now propose a Genetic Steiner Selector algorithm that will perform this selection of the optimal Steiner set.

IV. GSS Algorithm for use with RMST and RSA/G

Table 1: Tableau for the GSS Algorithm

Objective:	Given a set of terminals N, a possible root $r \in N$, and a set of candidate Steiner points C, find a subset $S \subseteq C$ of Steiner points such that: (1) the MST of the set V = N + S is the optimal RMST of N Or: (2) the result of the RSA/G algorithm when run on N, *r*, and S, returns the optimal MRSPSA for the pair (N, *r*).
Search Space Size:	$2^{\|N\|}$
Representation Scheme:	• Structure: Fixed-length bit string, one bit for each point in set C, so |C| bits. Each candidate point is either present (utilized) or absent (unused). • K = 2 (alphabet size, since we're using bits) • L = |C| • Mapping: The points in |C| will be pre-sorted (bucketed) in one particular order so that points close to each other in the string will be near each other geometrically (since distant candidates are relatively independent compared to close neighbors).
Fitness Cases:	One fitness case, the input points N (and *r* for the arborescence version).
Raw Fitness:	The total weight of either (1) the RMST obtained by running MST on (V+S) or (2) the MRSPSA obtained by running RSA/G on N+S and *r*.
Scaled Fitness:	A scaling window of 3 generations was used; thus, scaled fitness = F_MAX3 – f(x), where F_MAX3 is the largest fitness value seen in the last three generations.
Parameters:	• M = |N| • $P_c = 0.6$ • $P_m = 0.03$ • G = 50
Termination Criteria:	Number of generations G has been exceeded – there is no way of knowing what the optimal value of raw fitness is for this problem.

The GSS algorithm will use one bit in the encoding string to represent each candidate Steiner point. This is a natural encoding, because each Steiner point candidate can be present or absent in the final (and optimal) solution. Since all subsets $S \subseteq C$ are valid, all respective bit strings of length |C| are valid individuals as well; thus, no additional measures are necessary to penalize or repair invalid individuals. This is a very natural and effective encoding for this problem. It trivially follows that the search space for this problem is the set of all bit strings of length |C|, and thus the size of the search space is $2^{|C|}$. From Section III, the optimizations reduce the size of the candidate set |C| to O(|N|), where N is the input terminal set. Thus, the size of the search space is $2^{|N|}$. Considering that |N| often reaches values of 500 and above, the search space is very large indeed: $2^{500} \cong 10^{150}$. The typical search space, 2^{30} is also too large for an efficient exhaustive search: 1,073,741,824 possible configurations of the tree or arborescence.

The total number of schemata is $(2 + 1)^{|N|}$, equal to 3^{30} (10^{14}) for a typical test case and 3^{500} (10^{239}, an obscene amount) for an extremely large one. In a typical population, since we are using M = |N|, there are $|N| * 2^{|N|}$ schemata represented, or (approximately) $3*10^{11}$ for |N| = 30, $5*10^{152}$ for a large one.

The genetic operators active in this algorithm are the basic crossover, mutation, and reproduction operators that operate on bit strings. Since the mutation probability is non-zero, the algorithm is trivially able to explore the entire search space (mutation could set any bit to either 1 or 0).

The population size parameter M was chosen to be |N|. Given a problem such as this, where the length of the encoded individual L = O(|N|), the population size has to be directly dependent on |N|. Attempts to set the population size to a lower function of |N| (such as lg |N|, and $\sqrt{(|N|)}$) resulted in a more meandering algorithm that did not always converge within 50 generations.

The initial population of individuals was completely random; in a sense, it is already being primed by eliminating large amounts of completely useless Steiner point candidates during the pre-processing discussed in Section III. There was no easy way that I found to "start the algorithm off" with a couple well-chosen Steiner points.

Since the crossover operation is disruptive to schema of large defining length, we want alleles (bits in this case) that are dependent on each other to be in close proximity in the bit string. Since close-proximity Steiner points affect each other's savings, whereas distant Steiner points are practically completely independent, we have a simple scheme to sort candidate points to map them to close locations inside the bit string. We simply divide the bounding box of all the input terminals N into squares like a chessboard, which are numbered from 1 to A^2, where A is the size of the board. The points are then sorted according to the chessboard square number – the additional cost of O(N lg N) is irrelevant compared to later processing. Currently, A is chosen so that A^2 is on the same order of magnitude as |N|, but a more complicated scheme could be developed. Even this simple modification helped speed the convergence of the algorithm and allowed the decrease of the G parameter (number of generations allowed before termination).

The mutation parameter was set relatively high to promote convergence within 50 generations due to the large bit string lengths for this algorithm in the large test cases.

The fitness was initially set as a ratio between (Individual's Tree Weight) / (Tree Weight with no Steiner Points). However, this seemed to lead to poorer differentiation between individuals. The average length of an optimal Steiner tree, from experimental data, seems to be only 15-20% shorter than a simple MST on the same terminal input set. Practically all the ratios were thus between 1.0 and 0.80, and even with the additional fitness scaling performed by GENESIS, did not perform as well. When fitness was set to Max_Weight – IndividualWeight (linear fitness rather than a ratio), the algorithm converged quicker.

V. Experimental Results

The experiments were run using the GENESIS 5.0 software in C by John J. Grefenstette. The elitist strategy option was given, and the Fitness Scaling Window was set to 3 as described above. The machine being used for the results was a Pentium-4 1800 MHz with 2GB of memory, running Red Hat Linux kernel version 2.2.17, owned by Intel Corporation.

The fitness calculations (and thus the RSA/G and the MST algorithms) were implemented in C++ on the same platform and combined with GENESIS. Tests were performed on sets of points of given size |N|, with all the points evenly and randomly distributed in a square between (0,0) and (10000, 10000). This tends to be the worst-case scenario, as in randomly chosen points there are the most Hanan Grid Lines – points in actual layout tend to have large groupings along the same X or Y coordinate, and thus reduce the possible number of Hanan Candidate points.

On all various sizes, the GSS times are significantly slower than the already existing (and optimized) 1-IBS Steiner algorithm implemented in C++ derived from Robins [1], [3] (See Chart 1). However, the trend-lines seem to demonstrate that the 1-IBS algorithm increases at a faster rate: the best-fit power trend-line for the GSS times is $O(|N|^{2.993})$, whereas the best-fit power trend-line for the 1-IBS algorithm is $O(|N|^{3.92})$. This is an encouraging result for further study. The error performance of this GSS algorithm, however, was very good; the average error margin between the Total Tree Length on 1-IBS and the GSS algorithm was 1.2%, with the GSS algorithm usually getting slightly lower lengths. This level of error is nearly insignificant given the applications of these algorithms as discussed in this paper.

The best, average, and worst fitness curves of a single run on a population size of 240 (and a |N| size of 240) is given in Figure 2 (for the DSS-MST version). The worst-of-population starts out at 101% of the MST length (adding some Steiner points actually increased the total length), and then increases to 102% for the duration of the run. The best-of-population drops to 84% in Generation 1 from 89% in Generation 0, and then has several step-like drops, the biggest of which happen at generation 1, 12, and 39. For most of the end of the run, it is completely converged and does not change after Generation 40. The average population fitness varies up and down a bit, but it seems there are no huge meanderings of the curve – this is to be expected, because the fitness function here is inherently not deceptive – there is very rarely one particular point that produces huge gains and requires a set of others be turned off. Overall, the average decreases smoothly. The average is very close to the best-of-generation near the end of the run, which can suggest either (1) a lack of population diversity in that run toward the end, or (2) that for this particular problem, many individuals that include a set of several Steiner critical points produce nearly identical and nearly optimal results.

For the RSA/G version of the algorithm, DSS-RSA/G actually ran slightly faster than the comparable DSS-MST algorithm for the same input sizes. However, the MRSPSA 2-IdeA/G heuristic shown in Cong *et al.* [4], produces results nearly indistinguishable from DSS-RSA/G in accuracy (average error was less than 3.0%), at a time that is several orders of magnitude faster than DSS could produce. Despite several attempts to modify the genetic algorithm parameters for this problem run, the times achieved were nowhere near enough to compete with the 2-IdeA/G (or even other, slower heuristics) presented in [4].

Chart 1.

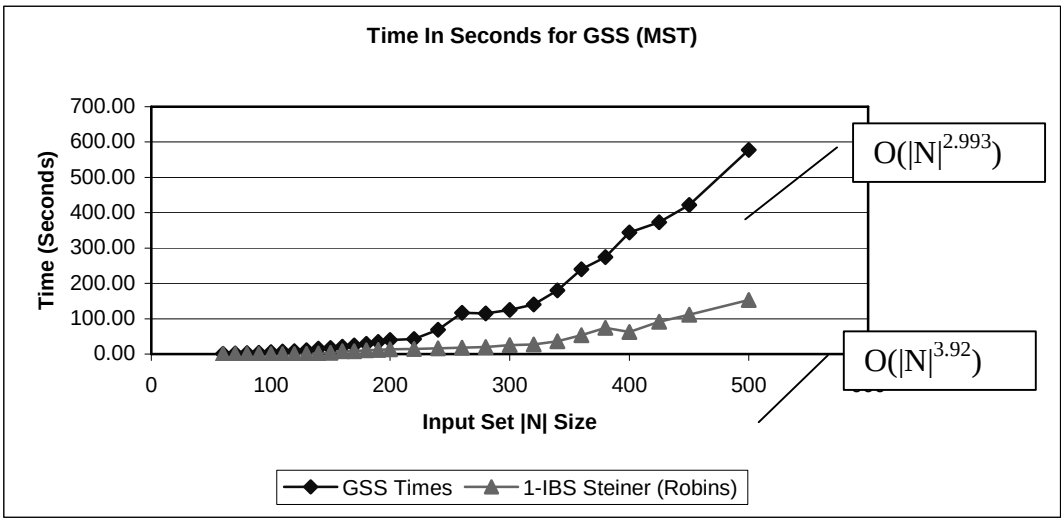


Chart 2.

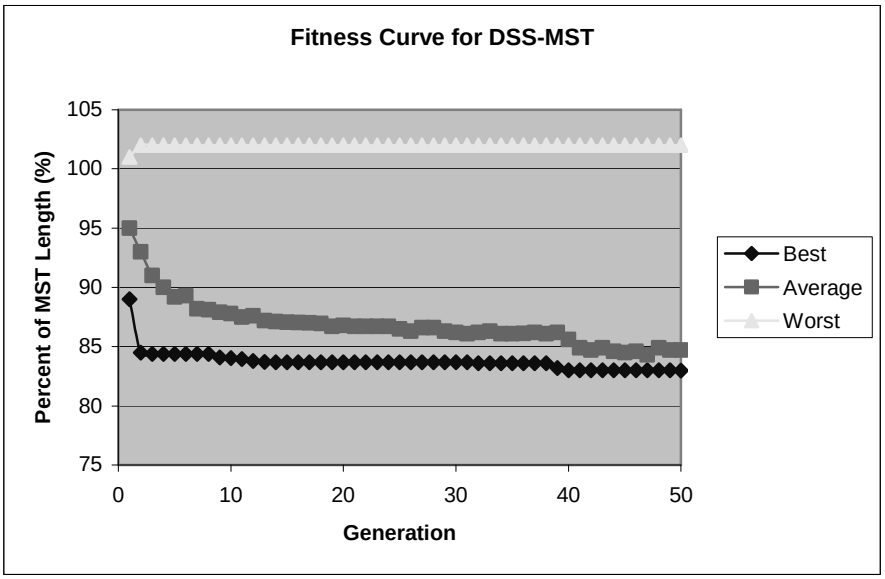
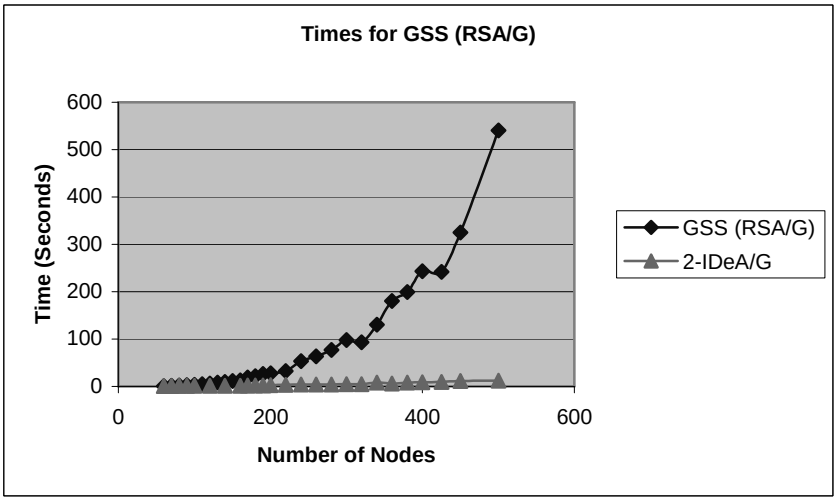


Chart 3.



VI. Conclusion

I have presented a genetic algorithm application to the common Steiner minimum tree and minimum shortest-path arborescence problem. The GSS algorithm is an entirely brand-new class of Steiner Tree and Arborescence algorithms, as far as the author is aware. For the RMST problem, it consistently produces results with average quality (error margin) just as good as other popular heuristic and/or randomized in the field (the 1-IBS Iterated Batch Steiner heuristic by Robins *et al.* is one of the most powerful results/time heuristics currently available and, on average, produces results within 5% from optimal). For the MRSPSA problem, it also produces results nearly indistinguishable in quality (length) from the existing heuristics outlined by Cong *et al.* [4].

The results generated by GSS show that this a problem that is well within reach of the genetic algorithm approach, and that the evolutionary operators are very well able to evaluate schemata for this problem in order to arrive at a nearly optimal solution.

The times taken to achieve these goals, however, is the only issue with the GSS algorithm. For the RMST problem, the algorithm seems to be relatively competitive, performing only 3.5 times slower for a input size of 500 terminals. With the rates of ascent of both algorithms compared, it is possible that for larger input sizes (|N| around 1000), the GSS algorithm will perform much better than 1-IBS. This shows that genetic algorithms can be applied in this area with relative success; the implementation of 1-IBS used has been heavily modified and optimized over a long period (over 6 months) by me and other developers; the actual fitness calculations performed in the GSS algorithm were very spottily optimized.

In the MRSPSA problem, however, the times produced by the GSS algorithm are nowhere near what is needed to compete with other existing algorithms in this field. A truly innovative new encoding and problem formulation would have to be found in order to produce results that will be applicable and useful.

VI. Future Work

The main concentration of further research in this area will be on the promising results demonstrated by DSS in the RMST problem field. There are two approaches which may very well yield very competitive results. First, the Genetic algorithm parameters and encoding may be re-examined, and perhaps a way will be found to achieve consistent convergence on slightly smaller populations sizes or within fewer generations. A 50% improvement in the population size required with bring the DSS-MST algorithm directly within striking distance of the 1-IBS heuristic for larger data sizes (|N| > 400). Furthermore, serious improvements (at the cost of memory) can be made using memoization due to the huge amount of identical computations that happens while evaluating the fitness of identical or extremely similar Steiner trees. Also, for large sizes of the input, various distant components of the tree are practically independent from each other; an addition of one Steiner node affects only one small area in the entire tree. Thus, an iterative or incremental evaluation method can be introduced (one such algorithm already exists, create by the author [unpublished], which computes in O(N) time the new RMST generated by adding a new Steiner point to a previous Steiner tree. This algorithm could be modified to handle the crossover and mutation operations to prevent recomputation of the entire tree, for a very considerable speedup (up to tenfold).

VII. References

[1] M. J. Alexander and G. Robins, “New performance-driven FPGA Routing Algorithms”, *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 15, No. 12, pp. 1505-1517, December 1996.

[2] A. B. Kahng, and G. Robins, "A New Class of Iterative Steiner Tree Heuristics With Good Performance", *IEEE Transactions on Computer-Aided Design*, 11 (7), 1992, pp. 893-902.

[3] C. S. Helvig, G. Robins, and A. Zelikovsky, “Improved Approximation Bounds for the Group Steiner Problem”, *In Proc. Conference on Design Automation and Test in Europe*, pages 406--413, 1998.

[4] J. Cong, A. B. Kahng, and K.-S. Leung, “Efficient Algorithms for the Minimum Shortest Path Steiner Arborescence Problem with Applications to VLSI Physical Design”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(1):24--39, 1998.

[5] John R. Koza, Genetic Programming, *MIT Press*, 1992

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

[7] M. Hanan, “On Steiner’s problem with rectilinear distance,” *SIAM J. Appl. Math.*, vol 14, pp. 255-265, 1966.

[8] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, “The rectilinear Steiner arborescence problem”, *Algorithmica*, vol 7. pp. 277-288, 1992.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, *MIT Press*, 1990.

[10] I.I. Mandoiu, V.V. Vazirani, and J.L. Ganley, “A New Heuristic for Rectilinear Steiner Trees”, *IEEE Transactions on CAD* 19 (2000), pp. 1129-1139

[11] J. S. Salowe and D. M. Warme, “An Exact Rectilinear Steiner Tree Algorithm”, in *Proc. IEEE Intl. Conf. on Computer Design*, Cambridge, MA, October 1993, pp. 472-475