computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

SOLVING THE RECTILINEAR STEINER MINIMAL TREE PROBLEM WITH A BRANCH AND CUT ALGORITHM

Nahit Emanet [1], Can Ozturan [2]

[1] Computer Engineering Department, Bogazici University, emanetn@boun.edu.tr, asma.cmpe.boun.edu.tr/~emanetn
[2] Computer Engineering Department, Bogazici University, ozturaca@boun.edu.tr, asma.cmpe.boun.edu.tr/~ozturan

Abstract: *This paper presents a new branch-and-cut algorithm that allows us to reduce the solution time of the concatenation phase of the rectilinear Steiner minimal tree problem in the plane. Our branch-and-cut algorithm is used on an integer programming formulation using what we call cutsec, and sec constraints. We present implementation details of our branch-and-cut program called NEOSteiner and provide computational results on test instances from the SteinLib library.*

Keywords: *Combinatorial problems, Minimum spanning hypertree, Submodular functions, Branch-and-cut algorithm, Separation of cutting planes, Maximum flow network.*

1. INTRODUCTION

Given a set V of points in a plane, the rectilinear Steiner minimal tree (RSMT) problem asks for a set S of Steiner points such that the total length of the minimum spanning tree (MST) over $S \cup V$ is minimized. The points are connected in the plane by using horizontal and vertical line segments. The decision version of this problem has been shown to be NP-complete by Garey and Johnson [1].

Much work has been devoted to the solution of Steiner tree problems including exact algorithms and heuristic procedures. Hwang, Richards and Winter [2] surveyed the Steiner tree problem in detail in their book. Surprisingly, in the benchmark tests, the fastest algorithms in terms of average running time are exact algorithms due to Warme, Winter and Zachariasen [3], and Polzin and Daneshmand [4]. The average running time of their branch-and-cut algorithm is better than the average running time of the best heuristic algorithms: The Batched Iterated 1-Steiner (BI1S) algorithm of Robins [5], and the primal-dual approximation algorithm of Mandiou, Vazirani and Ganley [6].

The GeoSteiner code by Warme et al. [3] divides the RSMT problem into two phases. In the FST generation phase, a set of *full Steiner trees* (FSTs) are identified by using a recursive sweep-line algorithm. In the FST concatenation phase, an RSMT is constructed from this set. Warme [7] shows that the FST concatenation problem can be reduced to finding an MST in a hypergraph, and formulates the MST problem in a hypergraph as an integer linear program using subtour elimination constraints (*sec*), and *cut* constraints, and solves it via branch-and-cut algorithm.

Our branch-and-cut algorithm differs considerably from the GeoSteiner algorithm not only in the formulation, but in the implementation, as well. Although sec and cut contraints are used in GeoSteiner to improve the relaxation problem, we use sec and *cutsec* constraints, and to find these constraints, we use the same maximum flow network without using any heuristic procedures. This approach enables us not only to write a simple program, but also to reduce the time to find the solution.

In the literature, two hyperedges are called *incompatible* if they have two or more points in common. If they have exactly one common point, they are called *compatible*. We introduce the concept of *strong incompatibility* that is defined as a maximal clique of incompatible hyperedges. Strong incompatibility refers to a complete subgraph consisting of only incompatible hyperedges and not contained in any other complete subgraph of incompatible hyperedges. By means of strong incompatibility, we improve the initial integer formulation.

Section 2 gives the integer formulation of the problem and defines certain aspects of the problem. Section 3 derives the formulations that allow us to use the same maximum flow network to solve the separation problem easily and efficiently. Section 4 gives the implementation details of the program. Section 5 compares our program with GeoSteiner

3.1 and STEINER software packages. GeoSteiner 3.1 was developed by Warme et al. [3] for solving Euclidean and rectilinear Steiner problems, and STEINER was developed by Polzin and Daneshmand [7] for network Steiner problems.

2. INTEGER PROGRAMMING FORMULATION

In this section, we present the integer formulation of MST problem in hypergraphs that is solved with our branch-and-cut algorithm.

Given a set V of points, and set of edges E for V. Let $H = (V, E)$ denote a hypergraph with the point set V and hyperedge set E. The polyhedron P defined by the following inequalities formulates the MST problem in hypergraph:

$$\sum_{i \in I} (|e_i| - 1) x_i = |V| - 1, I = \{0,1,2,...|E|-1\} \quad (1)$$

$$\sum_{i \in I} \max(0, |e_i \cap S| - 1) x_i \leq |S| - 1, S \subseteq V \quad (2)$$

$$\sum_{i \in \delta_i(S)} x_i \geq 1, S \subset V \quad (3)$$

$$\sum_{i \in I} \max(0, |e_i \cap S| - 1) x_i + \sum_{i \in \delta_i(S)} x_i \geq |S|, S \subseteq V \quad (4)$$

$$\sum_{i \in C} x_i \leq 1, C \subseteq I \quad (5)$$

$$0 \leq x_i \leq 1, \forall i \in I \quad (6)$$

For each hyperedge $e_i \in E$, a variable x_i indicates whether e_i is in the MST ($x_i = 1$), or not ($x_i = 0$). Let $\overline{S} = V - S$. Thus, $\delta(S)$ and $\delta_i(S)$ can be defined as $\delta(S) = \{e_i \mid e_i \cap S \neq 0 \; and \; e_i \cap \overline{S} \neq 0\}$ and $\delta_i(S) = \{i \mid i \; is \; the \; index \; of \; e_i \in \delta(S)\}$, respectively. Similarly, C can be defined as the index set of maximal clique of incompatible hyperedges.

The solution to the integer program, $\min(c^T x \mid x \in P \cap \mathrm{Z}^{|E|})$ where each element of c^T is the length of each hyperedge, gives us the MST in hypergraph.

Two well-known constraints for the MST, subtour elimination constraints (sec) and cut constraints, are shown in Equations (2) and (3), respectively. We call the Equation (4) as the *cutsec* constraints since cutsec constraints resemble a combination of both cut and sec constraints. We include all but the cut constraints in our integer programming formulation.

Actually, sec constraints are *facet defining*. Therefore, when we include them in our formulation, we know that cut and cutsec constraints become redundant. Initially, we included cut and cutsec constraints into our formulation so that they could speed up the convergence of the cutting plane algorithm. Although we have observed speed improvements by adding cutsec constraints into our integer formulation, we have not gained any improvement from cut constraints. Therefore, we decided to exclude the cut constraints from our integer formulation.

As an example, Fig. 1 shows rectilinear full Steiner trees and the corresponding hypergraph $H = (V, E)$ for seven points and five hyperedges. Cut, sec and cutsec constraints for the point set $S = \{3, 4, 6\}$ of Fig. 1 are defined as follows:

$$\text{Cut constraint}: x_2 + x_3 + x_4 + x_5 \geq 1 \quad (7)$$

$$\text{Sec Constraint}: 2x_1 + x_3 \leq 2 \quad (8)$$

$$\text{Cutsec Constraint}: 2x_1 + x_2 + 2x_3 + x_4 + x_5 \geq 3 \quad (9)$$

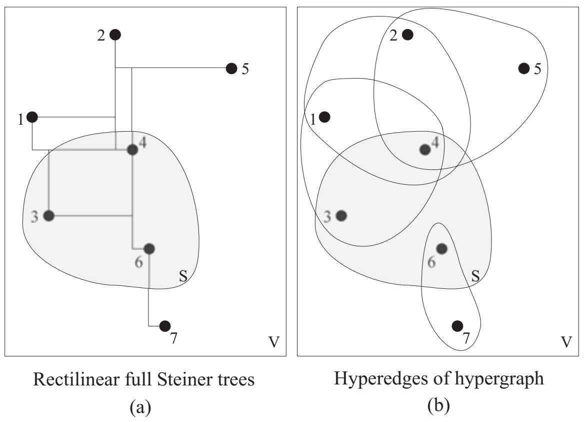

Fig.1 - Full Steiner trees and its hypergraph

3. GENERATING CUTTING PLANES

One of the crucial points in the branch-and-cut algorithm is to exploit strong cutting planes. Due to the theorem by Nemhauser and Wolsey [8], the separation problem is equivalent to submodular function minimization. In this section, we will give a maximum flow formulation that solves the cutsec separation problem in polynomial time. We define the following function:

$$f(S) = -|S| + \sum_{i \in I} \max(0, |e_i \cap S| - 1) x_i + \sum_{i \in \delta_i(S)} x_i \quad (10)$$

It is trivial to check that $f(S)$ is a submodular function. Thus, any $S \neq 0,\ ve\, f(S) < 0$ violates the cutsec constraints.

Due to a proposition by Nemhauser and Wolsey [8], when the submodular function $f(S)$ has the following form:

$$f(S) = -\sum_{T \subseteq S} c_T + \sum_{T \cap S \neq 0} r_T \qquad r_T, c_T \geq 0 \quad T \subseteq V \tag{11}$$

The *submodular function minimization* or the *separation problem* can be solved as a *maximum flow-minimum cut network* problem.

Specifically, we focus on only the nonlinear polynomial over binary programming problems of the following form:

$$\begin{gathered} f(S) = -\sum_{T \subseteq S} q_t \left(\prod_{j \in T} s_j \right) + \sum_{j \in S} a_j s_j \\ s_j = \{0,\ 1\}, \quad q_t, a_j \geq 0 \end{gathered} \tag{12}$$

As can be seen easily, Equation (11) can be shown to be equivalent to Equation (12) by taking

$$\begin{gathered} c_T = q_t \prod_{j \in T} s_j,\ T \subseteq S \\ r_T = a_j s_j, \leftrightarrow T = \{j\} \end{gathered} \tag{13}$$

Therefore, what we want to find is a way to express Equation (10) in the form of Equation (12).

$f(S)$ in Equation (10) can be rewritten as

$$\begin{gathered} f(S) = \sum_{i \in I} (|e_i| - 1) x_i - \\ \left(\sum_{i \in I} \max(|e_i \cap \overline{S}| - 1, 0) x_i + |S| \right) \end{gathered} \tag{14}$$

Let $f(S) = p - k$, where,

$$p = \sum_{i \in I} (|e_i| - 1) x_i \tag{15}$$

$$k = \sum_{i \in I} \max(|e_i \cap \overline{S}| - 1, 0) x_i + |S| \tag{16}$$

Let variable $s_j = 1$ if $j \in S$, and $s_j = 0$ otherwise. Then, we can write k in terms of the s_j as follows:

$$k = \sum_{i \in I} \left(\left(\sum_{j \in e_i} (1 - s_j) \right) - 1 + \prod_{j \in e_i} s_j \right) x_i + \sum_{i \in V} s_i$$

$$= \sum_{i \in I} \left(\sum_{j \in e_i} 1 - \sum_{j \in e_i} s_j - 1 + \prod_{j \in e_i} s_j \right) x_i + \sum_{i \in V} s_i$$

$$= \sum_{i \in I} \left((|e_i| - 1) - \sum_{j \in e_i} s_j + \prod_{j \in e_i} s_j \right) x_i + \sum_{i \in V} s_i$$

$$= \sum_{i \in I} (|e_i| - 1) x_i - \sum_{i \in I} \left(\sum_{j \in e_i} s_j \right) x_i + \sum_{i \in I} \left(\prod_{j \in e_i} s_j \right) x_i + \sum_{i \in V} s_i$$

$$= \sum_{i \in I} (|e_i| - 1) x_i - \sum_{j \in V} \left(s_j \sum_{i \in \delta_i(\{j\})} x_i \right) + \sum_{i \in I} \left(\prod_{j \in e_i} s_j \right) x_i + \sum_{i \in V} s_i$$

$$= \sum_{i \in I} (|e_i| - 1) x_i - \sum_{j \in V} b_j s_j + \sum_{i \in I} \left(\prod_{j \in e_i} s_j \right) x_i + \sum_{i \in V} s_i$$

$$= \sum_{i \in I} (|e_i| - 1) x_i - \sum_{j \in V} (b_j - 1) s_j + \sum_{i \in I} \left(\prod_{j \in e_i} s_j \right) x_i$$

Since $f(S) = p - k$, and $b_j = \sum_{i \in \delta_i(\{j\})} x_i$ then

$$f(S) = -\sum_{i \in I} \left(x_i \prod_{j \in e_i} s_j \right) + \sum_{j \in V} (b_j - 1) s_j \tag{17}$$

This result concludes that Equation (17) has the same form as Equation (12), and thus the separation problem for the cutsec constraints can be solved by finding maximum flow in the network D where

$$\begin{aligned} & D = (V_1 \cup V_2 \cup \{s\} \cup \{t\}, A_1 \cup A_2 \cup A_3) \\ & V_1 = \{e_i \in E \mid x_i > 0\} \\ & V_2 = \{j \in V\} \\ & A_1 = \{(s, X) \mid X \in V_1\} \text{ with capacity } c_{A1} = x_i \\ & A_2 = \{(X, Y) \mid X \in V_1, Y \in V_2\} \text{ with capacity } c_{A2} = \infty \\ & A_3 = \{(Y, t) \mid Y \in V_2\} \text{ with capacity } c_{A3} = b_j - 1 \\ & s \ = source \\ & t \ = t\arg et \end{aligned}$$

Variables s_j which take on value 1 in an optimal solution to $f(S)$ correspond to the labelled vertices when Ford and Furkerson [9] labelling procedure is applied after the maximum flow has been found in

D. An example network can be seen in Fig. 2.

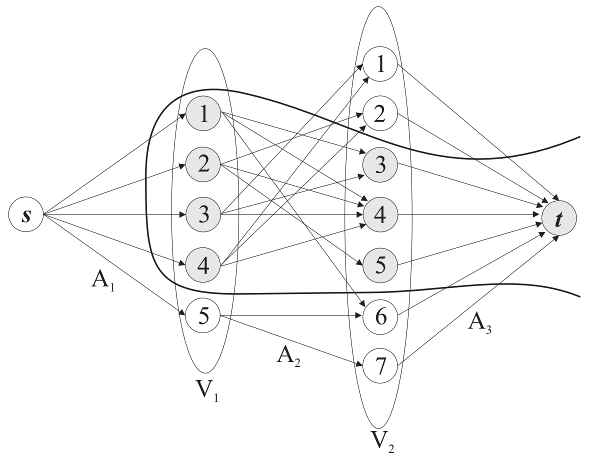

Fig.2 - A maximum flow network for Fig. 1.

Warme derived a similar result for the sec constraints. He shows that minimizing $f(S)$ for the sec constraints is equivalent to finding the minimum of the nonlinear polynomial over 0-1 variables of the form:

$$f(S) = -\sum_{i\in I}\left(x_i \prod_{j\in e_i}(1-s_j) \right) + \sum_{j\in V}(b_j - 1)(1-s_j) \quad (18)$$

By means of equation (17) and (18), we observe that the separation problem for the sec constraints can be solved by using the same network D that we use to find cutsec constraints. Thus, after solving the maximum flow network problem in D, the variables which are on the source and target side constructs the sec and cutsec constraints, respectively. This feature allows us to use the same maximum flow network D to obtain both cutsec and sec constraints.

4. IMPLEMENTATION DETAILS

Branch-and-cut algorithms are cutting plane algorithms combined with branch-and-bound algorithms. Actually, cutting plane algorithms improve the relaxation of the integer programming problem by starting with a small subset of constraints. They try to find violated constraints for the current LP solution. If they find one or more violated constraints, they add them to the current LP problem, and resolve the problem. If no constraint is violated and the current solution is an integer solution, then the current solution is the optimum solution to the problem.

Sometimes, this method can not find the optimum integer solution to the problem, or it needs large amount of time to solve it. In that case, branch-and-bound algorithms are applied by dividing the main problem into smaller problems such that each subproblem can be solved by cutting plane algorithms individually in the same manner. Fig. 3 presents the main steps of our branch-and-cut algorithm.

```
Algorithm BRANCH_AND_CUT()

(1) int status;
(2) Node node;
(3) begin
(4)   Initialize LP solver, and construct main node
(5)   repeat forever
(6)       status = process_node();
(7)       if status = INFEASIBLE then
(8)          delete node;
(9)       else if status  = INTEGRAL then
(10)            Save best solution and change upper bound
(11)            delete node;
(12)      else if status = FRACTIONAL then
(13)            Choose branching variable, and branch current node
(14)      end if
(15)      if (there is node to be processed)
(16)            select(node);
(17)      else
(18)         break;
(19)      end if
(20)  end repeat
(21) end.
```

Fig. 3 - Branch-and-cut algorithm

In step (4) of the branch-and-cut algorithm, the LP solver is fed with the total degree constraints of Equation (1), strong incompatibility constraints of Equation (5), and cutsec constraints of Equation (4) for each point with $|S| = 1$. We use Bron and Kerbosch [10] algorithm to find strong incompatibility constraints. Since the incompatible hyperedges graph is sparse, finding these constraints is not time consuming. We start separation and branch-and-bound sequence in step (5). Step (6) tries to find new violated constraints until the node in process is either infeasible as in step (7), or a new violated constraint is not found. If a new constraint is not found and the solution vector is integer as in step (9), we store the current solution vector as the best solution, and update the upper bound. If the solution vector is not integer as in step (12), we try to find a good branching variable such that the number of runs needed to solve the problem will be small. We divide the current node at the branching variable and add the child nodes into the branch-and-bound tree. Finally, in step (16), we try to find the best node to be processed next as the current node until no node is available.

The algorithm in Fig. 4 helps us to process each node of branch-and-cut algorithm. Processing a node continues until the solution is infeasible or no new constraint is found. In step (6), the current LP problem is solved. If the current solution is feasible, we try to exploit violated constraints by solving maximum flow networks as in step (12). If we find at least one violated constraint after solving the maximum flow network, we add it to the LP table, and resolve the newly constructed table in step (6). Otherwise, we either find a new integer solution or the solution is fractional. As can be seen in Fig. 4, we don't use any procedure to fix the variables. Any attempt to fix variables to either one or zero turned out to be ineffective in our tests. So, we decided to exclude variable fixing procedures in our program.

```
Algorithm int process_node()

(1) int status;
(2) int number_violated_constraints;
(3) bool is_solution_integer;
(4) begin
(5)   repeat forever
(6)     status = lp_solve(is_solution_integer);
(7)     if status = INFEASIBLE then
(8)       return INFEASIBLE; (* LP problem is infeasible *)
(9)     end if
(10)    (* Find violated constraints by solving maximum flow network *)
(12)    number_violated_constraints = generate_constraints();
(13)    if number_violated_constraints = 0 and is_solution_integer then
(14)      return INTEGRAL;
(15)    end if
(16)    if number_violated_constraints = 0 then
(17)      return FRACTIONAL;
(18)    end if
(19)  end repeat
(20) end.
```

Fig. 4 - Processing a node in branch-and-cut algorithm

Given an LP solution $x \in P$, the separation problem asks for finding the most violated sec and cutsec constraints (or prove that there is no such violated constraint) by solving at most $|V|$ maximum flow problems. Since the same maximum flow network is used to find both sec and cutsec constraints, the time to find the rectilinear Steiner tree is reduced. In our implementation, we use maximum flow network to solve the separation problem without using any heuristic procedures.

Most of the time, it is helpful to decompose a large graph into manageable pieces so that it can be processed one component at a time. Warme [7] suggested two theorems for connected and bi-connected components that can be used to find sec constraints. We have extended these theorems to find cutsec constraints. If a graph has a violated cutsec constraint, at least one of the connected or bi-connected components of the graph also has a violated constraint. Since solving a maximum flow network problem is very costly and in our program it is the only mechanism used to find violated constraints, we try to reduce the time to solve the maximum flow problem by reducing the size of the problem.

In addition to these decomposition algorithms, we applied a decomposition algorithm for both sec and cutsec constraints similar to *terminal elimination algorithm* which was originally suggested by Padberg and Wolsey [11]. However, our decomposition algorithm does not eliminate any point from consideration. We can define the congestion level of a point $j \in V$ as defined in Equation (17), $b_j = \sum_{i \in \delta_i(\{j\})} x_i$. Thus, congested and uncongested points can be defined as points with $b_j \geq 1$ and $b_j < 1$, respectively. Our decomposition method splits the components into two subcomponents such that while one component that is composed of only uncongested points is used to find the violated cutsec constraints, the other component that is composed of only the congested points is used to find the violated sec constraints.

In step (4) of the Fig. 5, we find all subcomponents of the current LP solution. These include connected, bi-connected, and congested and uncongested components. For each component generated, we first construct a maximum flow network in step (9), and try to find at least one violated sec and cutsec constraints in step (11). After examining all components, we add violated constraints into the LP solver and return the number of violated constraints. One important thing that should be taken into consideration is the constant value of MAX_COMPONENT_SIZE. It discards components whose sizes are greater than a certain value. This allows us to design a branch-and-cut algorithm such that we emphasize either cutting plane algorithm or branch-and-bound algorithm. When we set the MAX_COMPONENT_SIZE to 1, we are bound to use branch-and-bound algorithm, and when we set the MAX_COMPONENT_SIZE to the size of the original graph, we are bound to use the cutting plane algorithm. Thus, MAX_COMPONENT_SIZE allows us to adjust the branch-and-cut algorithm.

```
Algorithm int generate_constraints()

(1) int number_violations = 0;
(2) begin
(3)   (* Generate all subcomponents, and put them into List *)
(4)   Iterator p = make_components();
(5)   (* Iterate all subcomponents in the list *)
(6)   while (there is an element in the List)
(7)     if p.component_size < MAX_COMPONENT_SIZE then
(8)       (* Construct a max-flow network *)
(9)       max_flow = construct_maxflow_network();
(10)      (* Find violated constraints if one exists *)
(11)      number_violations += minimize_submodular_function(p);
(12)    end if
(13)    p.move(); (* Move iterator to the next element *)
(14)  end while
(15)  add_violated_constraints_to_LP();
(16)  return number_violations;
(17) end.
```

Fig. 5 - Generating violated constraints

There is a wide variety of branching variable selection strategies in the literature. We apply a strategy similar to the one suggested by Applegate, Bixby, Chvatal and Cook [12]. We choose a set of variables $C = x_i \mid 0.4 \le x_i \le 0.6$ as a branching variable candidate set. We then solve two LP problems $\min(c^T x \mid x \in P)$ for each value of the variable $x_i = 0$ or $x_i = 1$ using a limited number of simplex iterations, and represent the objective function values as v_i^0 and v_i^1, respectively. We finally select the branching variable b such that it satisfies the following condition:

$$\max\{v_b^0, v_b^1\} = \min_{x_i \in C}\{v_i^0, v_i^1\} \qquad (19)$$

For the node selection strategy, we use the best node selection strategy. We always choose the node with the lowest objective value.

5. COMPUTATIONAL RESULTS

In this section, we compare our program, NEOSteiner 2.0 (which is available for downloading at Emanet [13]), with GeoSteiner 3.1 and STEINER programs, which are written by Warme et al. [3], and Polzin and Daneshmand [4], respectively. As a test data, we use ES1000 test instances from the SteinLib library by Koch and Martin [14] which is accessible from the World Wide Web.

The FSTs that are used by the concatenation phase of the GeoSteiner 3.1 and NEOSteiner 2.0 are produced by applying the FST generation phase of GeoSteiner 3.1 with the pruning step enabled. Pruning step reduces the set of FSTs generated by the FST generation phase while retaining at least one optimal solution. This step reduces the solution time of the concatenation phase of the rectilinear Steiner minimal tree problem considerably.

All tests for GeoSteiner 3.1 and NEOSteiner 2.0 programs were performed on a PC with an AMD Athlon 1.4 GHz processor and 1 GB main memory running Mandrake Linux 9.0 operating system. We used the GNU gcc 3.2 compiler and CPLEX 8.0 LP solver. The same optimization options were used for both programs. The MAX_COMPONENT_SIZE was selected as 230 for NEOSteiner 2.0 program.

In Table 1, we compare the running time of GeoSteiner 3.1, STEINER and NEOSteiner 2.0 for the exact solution of the ES1000 test instances. We present the running time of STEINER as it is in Polzin and Daneshmand [4] article due to the unavailability of STEINER program. All tests in Polzin and Daneshmand article were performed on a PC with an AMD Athlon 1800+ processor and 1 GB of main memory. They used the gcc 2.94 compiler and CPLEX 7.0 LP solver on Linux 2.4.9 operating system. Our program is on average and on most of the test cases faster than GeoSteiner 3.1. Although results of NEOSteiner 2.0 are comparable with STEINER on most of the cases, there are a few cases where NEOSteiner 2.0 performs worse than STEINER.

Table 1. Comparison of GeoSteiner 3.1, STEINER and NEOSteiner 2.0 on ES1000FST instances

Instances	Optimum	GeoSteiner 3.1	STEINER	NEOSteiner 2.0
ES1000FST01	230535806	12.08	11.55	27.41
ES1000FST02	227886471	7.63	7.79	6.06
ES1000FST03	227807756	118.31	11.29	2.77
ES1000FST04	230200846	7.66	12.52	2.70
ES1000FST05	228330602	81.08	8.50	2.22
ES1000FST06	231028456	423.25	16.13	114.69
ES1000FST07	230945623	102.77	4.80	4.07
ES1000FST08	230639115	141.12	12.32	28.76
ES1000FST09	227745838	19.70	12.72	4.17
ES1000FST10	229267101	114.30	4.76	6.18
ES1000FST11	231605619	18.28	8.13	7.56
ES1000FST12	230904712	611.79	16.47	13.59
ES1000FST13	228031092	2.64	4.62	4.28
ES1000FST14	234318491	760.27	14.92	25.17
ES1000FST15	229965775	7.90	7.59	1.79
Averages:		**161.92 (s)**	**10.27 (s)**	**16.76 (s)**

6. CONCLUSION

Our aim in this work is twofold. One is to design simple and reusable branch-and-cut algorithm that can be used to solve the RSMT problem as a stand alone sequential program or as a main part of a parallel algorithm. In this paper, we only presented the sequential algorithm. The parallel algorithm is explained in detail in another paper by Emanet and Ozturan [15]. The other aim is to reduce the time to solve the problem.

Test results show us that our branch-and-cut algorithm outperforms the GeoSteiner 3.1 in most of the test cases, even though our code is implemented in C++ with liberal use of object oriented features which may introduce some overheads. The STEINER program performs better than NEOSteiner 2.0. However, this is mainly due to the extensive use of strong reduction techniques in STEINER program. STEINER can be considered as a middle phase between FST generation and FST concatenation phase of Steiner minimal tree problem with its extended reduction tests like terminal separator technique, which reduces the size of a test instance by ensuring at least one optimal solution. In the future, one can easily integrate these reduction techniques into the NEOSteiner program, which has already been designed and implemented for including new algorithms, to obtain better results for large instances of Steiner minimal tree problem.

While developing NEOSteiner program, we do not sacrifice software issues such as modifiability, reusability, maintainability, and portability for performance. These are the issues often a branch-and-cut program fails to satisfy due to the complexity of the problem.

7. REFERENCES

[1] M.R. Garey. D.S. Johnson. The rectilinear Steiner problem is NP-Complete, *SIAM J. Appl. Math.* 32 (1977). p. 826-834.

[2] F.K. Hwang. D.S. Richards. P. Winter. *The Steiner tree problem*. Elsevier Science Publishers. Amsterdam, The Netherlands, 1990.

[3] D.M. Warme. P. Winter. M. Zachariasen. Exact algorithms for plane Steiner tree problems: A computational study. in: D-Z. Du. J.M. Smith. J.H. Rubinstein. (Editors), *Advances in Steiner Trees*. Kluwer Academic Publishers. Massachusetts, 2000. p. 81-116.

[4] T. Polzin. S.V. Daneshmand. On Steiner trees and minimum spanning trees in hypergraphs, *Operations Research Letters* 31 (2003). p. 12-20.

[5] G. Robins. On optimal interconnections, PhD thesis, Department of Computer Science, UCLA, 1992.

[6] I. Mandiou. V. Vazirani. J. Ganley. A new heuristic for rectilinear Steiner trees. *IEEE Transactions on CAD*, 19 (2000). 1129-1139.

[7] D.M. Warme. Spanning trees in hypergraphs with applications to Steiner trees, PhD thesis, Department of Computer Science, The University of Virginia, 1998.

[8] G.L. Nemhauser. L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Publication, 1988.

[9] L.R. Ford. D.R. Fulkerson. *Flows in Network*. Princeton University Press, New Jersey, 1962.

[10] C. Bron. J. Kerbosch. Finding of all cliques of an undirected graph, *Comm. ACM.* 16 (1973) p. 575-577.

[11] M. Padberg. L. Wolsey. Trees and cuts, *Annals of Discrete Mathematics*, 17 (1983).

[12] D. Applegate. R. Bixby. V. Chvatal. W. Cook. Finding cuts in the TSP, Technical Report, Mathematics, AT&T Bell Laboratories, NJ, 1994.

[13] N. Emanet. NEOSteiner program. http://asma.cmpe.boun.edu.tr/~emanetn/NEO.html.

[14] T. Koch. A. Martin. SteinLib. http://elib.zib.de/steinlib/steinlib.php.

[15] N. Emanet. C. Ozturan. Dogrulu Steiner agac problemlerinin seri ve paralel algoritmalar ile cozumu, in: F.E. Sevilgen. H. Sadikouglu. (Editors), *Yuksek Performansli Bilisim Sempozyumu*, Gebze YTE, Kocaeli, 2002. p. 35-38.

After graduating from Istanbul Ataturk Science High School, ***Nahit Emanet*** *enrolled Bogazici University Computer Engineering department. He earned B.S. and M.Sc. degrees from this department. Today, he is pursuing his Ph.D. thesis on "Parallel and Sequential Algorithms for Steiner Tree Problems". At the same time, he is doing research on computer controlled machine automation and optimization in a private company. His research interests include parallel and sequential algorithm design, real-time operating systems, embedded systems, computer architecture and organization, VLSI design automation, and compiler design.*

Dr. ***Can Özturan*** *earned his PhD degree from the Computer Science Department of Rensselaer Polytechnic Institute in 1995. After spending one year as a postdoc staff scientist at ICASE, NASA Langley Research Center, he joined Bogazici University in Istanbul, Turkey as a faculty member in 1996. Today, he is an associate professor in the Department of Computer Engineering. His research interests are parallel processing, grid computing, scientific computing and graph algorithms.*