SOLVING STEINER TREE PROBLEMS IN GRAPHS WITH LAGRANGIAN RELAXATION

LAURA BAHIENSE, FRANCISCO BARAHONA, AND OSCAR PORTO

ABSTRACT. This paper presents an algorithm to obtain near optimal solutions for the Steiner tree problem in graphs. It is based on a Lagrangian relaxation of a multi-commodity flow formulation of the problem. An extension of the subgradient algorithm, the *volume algorithm,* has been used to obtain lower bounds and to estimate primal solutions. Due to the high quality of the bounds, it was possible to solve many difficult instances from the literature to proven optimality, without preprocessing or branching. Computational results are reported for many problems drawn from the literature, including those in the *SteinLib* library.

1. INTRODUCTION

Given an undirected graph $G = (V, E)$ and a subset of the nodes $T \subseteq V$ called *terminals*, a *Steiner tree* for T in G is a tree that spans T. Let c_{ij}, for each $(i, j) \in E$, be nonnegative *costs* associated to the edges of G. The *Steiner tree problem in graphs* (STPG) asks for the Steiner tree of minimum total edge cost. The vertices in $V \backslash T$ are called *non-terminals*. Non-terminal vertices that end up in an optimum Steiner tree are called *Steiner vertices*. The STPG is known to be *NP*-Hard [27] for a general graph, remaining *NP*-Hard for particular classes of graphs such as grid graphs [18], among others. The STPG has been widely applied in the design of communication, distribution and transportation systems. It is also applied to problems such as the wire routing phase in physical VLSI design [31], network design [35], etc.

Different integer programming formulations of the STPG can be found in the works of Maculan [11, 34] and Goemans & Myung [19]. In the latter, the equivalence between some of the given formulations is shown. The literature includes the following solution methods: approximation algorithms [7, 45, 20]; dynamic programming [14, 29]; branch-and-cut [8, 9, 10, 33, 28]; dual ascent [49]; Lagrangian relaxation [6, 5], Lagrangian relaxation and polyhedral methods [32]; neural networks [38]; meta-heuristics (genetic algorithms [17, 26], simulated annealing [13], tabu-search [15, 50, 51, 40], and GRASP [36, 37]). Stand alone heuristics for the STPG have also been developed; see for example [43, 44, 47, 16]. Due to its outstanding performance, the one developed in [43] by Takahashi & Matsuyama is the basis of several meta-heuristics and primal heuristics developed for the STPG.

Many reduction techniques for the STPG have been developed, several of them can be found in [28]. A good reference is the book [24]. Very good surveys are also given in [46, 24, 34, 25].

This paper presents a new algorithm to obtain near optimal solutions for the STPG. This algorithm is based on a multicommodity flow formulation. Due to the large size of the model obtained, it is tackled with Lagrangian relaxation, and the *Volume Algorithm* (VA) is used in the relaxed problem. The VA is an extension of the subgradient algorithm that produces primal as well as dual solutions. The primal vectors are used as inputs for several heuristics that construct integer solutions. Due to the high quality of the bounds, it was possible to solve many difficult instances from the literature to proven optimality, without preprocessing or branching. In the instances tested where optimality could not be proven, the resulting gaps have been fairly small. Computational results are reported for problems drawn from the literature, including those contained in the *SteinLib* library.

This paper is organized as follows. Section 2 describes the integer programming formulation of the STPG used in this work. Dual bounds and the Volume Algorithm are presented in Section 3. Section 4 describes the primal heuristics used in this work. Finally, Section 5 provides computational results.

2. Formulation

In this section, two formulations of the STPG are discussed. As mentioned above, the STPG is defined on an undirected graph. A directed version of the problem, the Steiner arborescence problem, is defined as follows. Given a digraph $D = (V, A)$, a set of terminals $T \subseteq V$ and a root vertex $r \in T$, a *Steiner arborescence* is a tree directed away from the root r spanning T. The *Steiner arborescence problem* (SAP) asks for the minimum total arc cost Steiner arborescence. Any instance of the STPG can be formulated as a bidirected SAP (see [9] for properties of both formulations).

In this study, the bidirected SAP is formulated as a *non-simultaneous multi-commodity flow problem* in a digraph $D = (V, A)$. This directed graph has the same set of vertices of the original graph $G = (V, E)$ and its set of arcs A is obtained as follows: for each edge $(i, j) \in E$, two arcs, $(i,\vec{j})$ and $(j,\vec{i})$, both with the same weight c_{ij} of edge (i, j), are included in A.

The formulation below was simultaneously introduced by Claus & Maculan [11] and Wong [49]. A vertex $r \in T$ is chosen to be the *source* offering $|T_0| := |T \backslash \{r\}|$ commodities, one demanded by each of the remaining $|T_0|$ terminal vertices. Variables f_{ij}^k, $(i,\vec{j}) \in A$ and $k = 1, \ldots, T_0$, indicate the flow amount of commodity k going through the arc $(i,\vec{j})$. Binary variables x_{ij}, $(i,\vec{j}) \in A$, control the inclusion ($x_{ij} = 1$) or not ($x_{ij} = 0$) of arc $(i,\vec{j})$ in the solution. Naming $I(i)$ the set of vertices $j \in V$ such that $(j,\vec{i}) \in A$ and $O(i)$ the set of vertices $j \in V$ such that $(i,\vec{j}) \in A$, the SAP can be formulated as follows:

$$
(1)\qquad
\begin{cases}
\min\limits_{x:=(x_{ij})_{(i,\vec{j})\in A}} cx & \\
\sum\limits_{j\in O(r)} f^k_{rj} - \sum\limits_{j\in I(r)} f^k_{jr} = 1\,, \quad k\in T_0 & (a_1)\\
\sum\limits_{j\in O(k)} f^k_{kj} - \sum\limits_{j\in I(k)} f^k_{jk} = -1\,, \quad k\in T_0 & (a_2)\\
\sum\limits_{j\in O(i)} f^k_{ij} - \sum\limits_{j\in I(i)} f^k_{ji} = 0\,, \quad i\in V\setminus\{r,k\},\quad k\in T_0 & (a_3)\\
x_{ij}\in\{0,1\}\,, \quad (i,\vec{j})\in A\,. & (a_4)\\
0\le f^k_{ij}\le x_{ij}\,, \quad (i,\vec{j})\in A\,, \quad k\in T_0\,. & (a_5)
\end{cases}
$$

Equations $(a_1), (a_2)$ and (a_3) represent flow conservation for the terminal vertex chosen to be the source, for the remaining terminal vertices and for the non-terminal vertices, respectively. Constraints (a_5) allow a non-zero flow f^k_{ij} of any commodity k through an arc $(i,\vec{j})$ only if the latter is included in the solution. Finally, in the objective function, the total sum of the costs for the arcs included in the solution (a Steiner tree) is minimized.

The last part of this section analyzes the relationship between the continuous relaxations of two different integer programming formulations for the SAP, the one used in this work and the classical dicut formulation for the problem. Notice that the formulation above has a polynomial number of constraints and a polynomial number of variables. Let P_{xf} be the polytope defined by $(a_1), (a_2), (a_3)$, the continuous relaxation of (a_4) and (a_5). It can be shown (see [34]) that the polytope P_x obtained by projecting P_{xf} onto the x variables can be characterized as follows:

$$P_x = \{x\,:\, x(\delta^+(S))\ge 1, \text{for all } S\subset V \text{ with } r\in S \text{ and } (V\setminus S)\cap T\neq\emptyset;\ 0\le x_{ij}\le 1,\ (i,\vec{j})\in A\}$$

with $\delta^+(S) := \{(i,\vec{j})\in A : i\in S, j\in V\setminus S\}$ and $x(\delta^+(S)) := \sum\limits_{(i,\vec{j})\in\delta^+(S)} x_{ij}$.

The *dicut formulation* for the SAP, based on the polytope P_x and proposed first in [1], is as follows:

$$
(2)\qquad
\begin{cases}
\min\limits_{x:=(x_{ij})_{(i,\vec{j})\in A}} cx & \\
x(\delta^+(S))\ge 1\,, \quad \text{for all } S\subset V\,,\ r\in S\,,\ (V\setminus S)\cap T\neq\emptyset & (a_6)\\
x_{ij}\in\{0,1\}\,, \quad (i,\vec{j})\in A &
\end{cases}
$$

Since P_x is the projection onto the x variables of P_{xf}, the continuous relaxations of (1) and (2) are equivalent. It is important to mention that even with an exponential number of constraints, the continuous relaxation of the dicut formulation for the SAP can be solved in polynomial time with the ellipsoid algorithm. The reason for that is the polynomiality of the separation problem over the *Steiner dicut inequalities* (a_6); it can be tackled by solving a maximum flow problem for each terminal vertex in the set T_0. The dicut formulation for the SAP is the one most used in the literature. Examples include the approximation algorithm presented in [20] and branch and cut algorithms presented in [8, 28]. The latter combine a minimum cut algorithm and the Simplex method, used iteratively. Due to its size, formulation (1) has been mostly ignored. To our knowledge, only Wong [49] has used it with a dual ascent method. He reports results on graphs with up to 60 nodes and 120 edges.

In this paper, Lagrangian relaxation is used to tackle (1), and subgradient techniques are used in such a way that at every iteration the resulting subproblem is trivial to solve. A difference that should be stressed is that the algorithm presented in this study does not require the use of the Simplex method at any stage.

3. Dual Bounds

This section describes how lower bounds for the SAP are computed. Flow conservation constraints (a_1)-(a_3) of (1) are dualized in a Lagrangian way with multipliers π_i^k, for $i \in V$ and $1 \leq k \leq |T_0|$. Let $\pi = (\pi_i^k) \in \mathbb{R}^{|V||T_0|}$ be the resulting vector of Lagrangian multipliers. Finally the integrality constraints (a_4) are linearly relaxed and the following *Lagrangian subproblem* is obtained:

$$(3) \qquad (\theta(\pi)) \begin{cases} \min\limits_{\substack{x:=(x_{ij})_{(i,\vec{j})\in A} \\ f:=(f_{ij})_{(i,\vec{j})\in A}} & cx + \sum\limits_{k\in T_0} \ell^k f^k + \xi(\pi) \\ \quad x_{ij} \in [0,1], \quad (i,\vec{j}) \in A\,. & (a_4') \\ \quad 0 \leq f_{ij}^k \leq x_{ij}, \quad (i,\vec{j}) \in A, \quad k \in T_0, & (a_5) \end{cases}$$

where $\ell^k := (\ell_{ij}^k) = (\pi_i^k - \pi_j^k)$ is the vector of *Lagrangian costs* of variables f_{ij}^k and $\xi(\pi)$ is a constant factor easily computed for any fix π. For any value of π, the solution of $(\theta(\pi))$ gives a valid lower bound on the optimal solution of (1). The best of these bounds is given by a solution of the following Lagrangian dual problem:

$$(4) \qquad \max_{\pi\in\mathbb{R}^{|V||T_0|}} \theta(\pi)$$

Subsections 3.2 and 3.3 show how the volume algorithm is used to solve the Lagrangian dual problem (4). As mentioned before, special care has been taken to use Lagrangian

relaxation and subgradient techniques in such a way that the resulting Lagrangian subproblems are trivial to solve. The next subsection describes a simple inspection algorithm that solves the Lagrangian subproblems.

3.1. **Solving the Lagrangian subproblem.** For any fixed vector of Lagrangian multipliers π, the resulting Lagrangian subproblem (3) decomposes into a set of independent problems, one for each arc $(i,\vec{j}) \in A$. After dropping the indices i, j the solution is:

- If $\sum_{k:\,\ell^k<0} |\ell^k| \;>\; c$, then:
  - $x = 1$,
  - $f^k = 1$, for every k such that $\ell^k < 0$,
  - $f^k = 0$, for every k such that $\ell^k \geq 0$;
- If $\sum_{k:\,\ell^k<0} |\ell^k| \;\leq\; c$, then:
  - $x = 0$,
  - $f^k = 0$, for every k.

The rest of this section is dedicated to the solution of the Lagrangian dual problem (4).

3.2. **Solving the Lagrangian dual problem.** A similar relaxation has been used in [23], where the subgradient method [21] is used in the traditional way. Since this only produces dual variables, branching decisions are based on this dual information. In this paper the Lagrangian dual problem (4) is solved with the volume algorithm. This is an extension of the subgradient method that produces primal solutions as well as dual solutions, see [3]. It can be seen as a fast way to approximate Dantzig-Wolfe decomposition [12]. The name "volume" is inspired by the fact that the primal values come from computing the volumes below the faces of the dual problem. See [2] for a study of its convergence. A similar approach for uncapacitated facility location problems appears in [4]. A description of the algorithm is below.

Volume algorithm (VA)

STEP 0. [*Initialization*]

Let $\pi^* \in \mathbb{R}^m$ be a vector of Lagrangian multipliers. Solve (3) with $\pi := \pi^*$. Let $(\hat{x}_0, \hat{f}_0)$ be an optimal solution of (3), and z^* the value of the objective function. Set $t := 0$ and $(\bar{x}_0, \bar{f}_0) := (\hat{x}_0, \hat{f}_0)$.

STEP 1a. [*"Supergradient" Displacement*]

Let $\bar{v}_t = b - A\bar{f}_t$, where $Af = b$ is the system (a_1), (a_2), (a_3). Perform a ``*supergradient*'' *displacement*:

$$\bar{\pi} := \pi^* + s_t \bar{v}_t\,,$$

where the step size s_t is given by (7).

STEP 1b. [*Solving the Subproblem*]
Solve (3) with $\pi := \bar{\pi}$.
Let: $(\hat{x}_{t+1}, \hat{f}_{t+1})$ be an optimal solution of (3), and z_{t+1} the value of the objective function.

STEP 2a. [*Primal Update*]
Compute

$$(\bar{x}_{t+1}, \bar{f}_{t+1}) := \alpha\,(\hat{x}_{t+1}, \hat{f}_{t+1}) \;+\; (1-\alpha)\,(\bar{x}_t, \bar{f}_t)\,. \tag{5}$$

The value of α is discussed below.

STEP 3. [*Dual-Test*]
Perform the *Dual-Test*:

$$\text{if } z_{t+1} > z^*, \text{ then update } z^* := z_{t+1},\ \pi^* := \bar{\pi}. \tag{6}$$

STEP 4. [*Loop*]
Check stopping criteria. Do $t := t+1$ and go to STEP 1a. □

The step size is given by

$$\lambda \frac{T - z^*}{\|\bar{v}_t\|^2}, \tag{7}$$

where $0 < \lambda < 2$, and T is a *target* value. A small value of T is used at the beginning and each time that z^* is within 5% of T, the value of T is increased by 5%. In order to set the value of lambda, three types of iterations are defined:

- If there is no improvement in the value of z^*, the iteration is called *red*. A sequence of red iterations suggest the need for a smaller step-size.
- If $z_{t+1} > z^*$ the following number is computed:

  $$a = \bar{v}_t \cdot (b - A\hat{f}_t).$$

  If $a < 0$, it means that a longer step-size would have given a smaller value for z_{t+1}. This iteration is called *yellow*.
- If $a \geq 0$, this iteration is called *green*. A green iteration suggests the need for a larger step-size.

The initial value of λ was 0.1. After a sequence of 20 consecutive red iterations and if $\lambda > 10^{-4}$, the value of λ is multiplied by 0.67. After each green iteration and if $\lambda < 1$, this value is multiplied by 2.

To chose the value of α, an upper bound $\alpha_{\max} = 0.001$ is set. Then the following value is computed:

$$\gamma_{\text{opt}} := \operatorname*{Argmin}_{\gamma \in \mathbb{R}} \|\gamma \hat{v}_{t+1} + (1-\gamma)\bar{v}_{t+1}\|^2\,.$$

Then if $\gamma_{\text{opt}} < 0$ the value of α is given by $\alpha := \frac{\alpha_{\max}}{10}$; otherwise $\alpha := \min\{\gamma_{\text{opt}}, \alpha_{\max}\}$. After every 250 iterations, the progress in the objective value is monitored. If the increase is less than 1% and $\alpha_{\max} > 10^{-5}$, then $\alpha_{\max}$ is divided by 2. This idea was first used in conjugate subgradient methods (see [30, 48]). However, one difference here is the use of $\alpha_{\max}$, and another important difference is that this is used to produce primal solutions.

As can be seen in step [*Dual-Test*] the dual vector π^* is updated only when a dual improvement has been obtained. In a regular subgradient method the update is done at every iteration. In the VA, the multiplier π^* plays the same role as the *stability center* in *bundle methods* [22].

Note that in [***Primal Update***], $(\bar{x}_{t+1}, \bar{f}_{t+1}) := \alpha(\hat{x}_{t+1}, \hat{f}_{t+1}) + (1-\alpha)(\bar{x}_t, \bar{f}_t) = \alpha(\hat{x}_{t+1}, \hat{f}_{t+1}) + (1-\alpha)\alpha(\hat{x}_t, \hat{f}_t) + \ldots + (1-\alpha)^{t+1}(\hat{x}_0, \hat{f}_0)$. This should be seen as a convex combination of $\{(\hat{x}_0, \hat{f}_0), \ldots, (\hat{x}_{t+1}, \hat{f}_{t+1})\}$. The assumption that this sequence approximates an optimal solution of the continuous relaxation of (1) is based on a theorem in linear programming duality that appears in [3]. Due to the exponential decrease of the coefficients of this convex combination, later vectors receive a much larger weight than those that appeared only in earlier iterations.

Three stopping criteria were used, and they are as follows:

1. *Linear-Optimality*. If the maximum absolute value of the components of $b - A\hat{f}$ was less than 0.001 and
$$\frac{|z^* - c\bar{x}_t|}{z^*} < 0.001.$$
2. *Integer-Optimality*. Under the assumption that all costs are integer, if the difference between the best upper bound and z^* is less than 1.
3. *Iterations-Time*. If the number of iterations was more than 300000 or the number of CPU seconds was more than 10000.

4. Primal Solutions

The vector $(\bar{x}_t, \bar{f}_t)$ is an approximation of an optimal solution of the linear programming relaxation of (1). The idea in this section is to use this primal information to derive an integer solution.

Denote by $\bar{x}$ the vector $\bar{x}_t$. For every edge (i,j) set $\bar{y}_{ij} = \bar{x}_{ij} + \bar{x}_{ji}$. Three heuristics for finding Steiner trees are presented below.

I. Minimum spanning tree with volumetric weights (MSTV).

- `Find a minimum spanning tree in the graph` $G = (V, E)$ `with arc weights:`

. c_{ij} , when $t = 0$;

. $-\bar{y}_{ij}$, when $t > 0$.

- Prune all non-terminal leaves.

II. Minimum spanning tree in a modified graph (MSTM).

The second heuristic is applied in a subgraph of G. For a given $\beta, 0 \leq \beta \leq 1$, let $G_\beta = (V_\beta, E_\beta)$ be the graph induced by the set of vertices V_β consisting of all terminal vertices and the nonterminal vertices i such that $\sum_j \bar{y}_{ij} \geq \beta$. The second heuristic is below:

- Find the largest value of $\beta \in \{0, 0.1, 0.2, ..., 0.9, 1\}$, for which the graph $G_\beta(V_\beta, E_\beta)$ is connected.
- Find a minimum spanning tree in the graph $G_\beta = (V_\beta, E_\beta)$ with arc weights:
  . c_{ij} , when $t = 0$;
  . $(1 - \bar{y}_{ij})\, c_{ij}$, when $t > 0$.
- Prune all non-terminal leaves.

III. T&M heuristic with volumetric weights (T&MV).

Given a graph $G = (V, E)$ with nonnegative arc costs c_{ij}, for each $(i, j) \in E$, the Takahashi & Matsuyama (T & M) heuristic for the STPG [43] is as follows:

1. Choose an initial terminal vertex v_i; let $k = 1$;
2. Connect by the shortest path v_i with the terminal vertex v_j, $j \neq i$, that is closest to v_i; let T_1 be the subtree obtained;
3. Connect by the shortest path the subtree T_k with the terminal vertex v_l, $l \notin T_k$, that is closest to T_k; let T_{k+1} be the subtree obtained;
4. Stop if all terminals are connected; otherwise do $k \leftarrow k + 1$ and go to step 3.

The third heuristic is as follows:

- Given a starting vertex v_i, run the Takahashi & Matsuyama heuristic for the digraph $D = (V, A)$ with arc weights:
  . $c_{\vec{ij}} = c_{\vec{ji}} = c_{ij}$, when $t = 0$;
  . $c_{\vec{ij}} = c_{\vec{ji}} = (1 - \bar{y}_{ij})\, c_{ij}$, when $t > 0$.

5. Computational Results

This section presents extensive computational experience. The code was implemented in C++ and all instances were run on an IBM RS6000 workstation with a 332 Mhz

processor. The instances were obtained from the *SteinLib* library, available at
`ftp://ftp.zib.de/pub/mp-testdata/SteinLib/` or
`http://www.zib.de/pub/mp-testdata/SteinLib/`.

The instances treated consist of the following series:

- B, C, D and E: test cases introduced by J. Beasley [5]. They are defined on sparse random graphs with Euclidean costs.
- R: test cases introduced by J. Soukup and W. F. Chow [42], defined on a grid with no holes.
- ALUE, ALUT, DIW, DMXA, GAP, MSM and TAQ: cases derived from VLSI circuits introduced by T. Koch e A. Martin [28]. They are defined on grids with holes.

For series B, C, D, E and R, the heuristics $MSTV$ and $MSTM$ are run every 20 iterations; the $T\&MV$ heuristic is run every 200 or 500 iterations depending on the problem size. For the other instances, the $MSTV$ and $MSTM$ heuristics are run every 100 or 200 iterations, and the $T\&MV$ heuristics is run every 1000, 2500, or 5000 iterations, depending on the size of the instance.

The appendix contains tables for all cases treated. Each line corresponds to an instance. The first column gives the name of the instance, and the next three columns give the number of nodes, edges and terminals respectively. The fourth column (LB) gives the lower bound. The next column (UB) contains the best upper bound found by the primal heuristics. The sixth column (GAP %) gives the gap in percentage defined by $100(UB - LB)/LB$. The next column gives the CPU time. The next three columns $(MSTV, MSTM, T\&MV)$ show the best upper bound produced by each of the primal heuristics. Finally, the last column (PC) compares performance between the algorithm proposed in this paper and the one presented in [28]. The comparison is not based on CPU time, but on the bounds produced as follows:

- $*^+$ means that optimality was proved by this algorithm and not by [28];
- $*^-$ means that optimality was proved by [28]; and not in the present paper;
- $+$ means that neither algorithm proved optimality, but the one in this paper gave a smaller gap.
- $-$ means that neither algorithm proved optimality, and the algorithm of [28] produced a smaller gap.
- $\sim$ implies that both algorithms gave similar results: either both proved optimality, or both obtained the same gap.

Table 1 below contains one line for each series of instances. The first column ($Series$) shows the name of the series, the second column (N) indicates the number of instances in the series. The third column ($OPT(\%)$) displays the number of instances for which optimality was proved. The next column ($Gap_m(\%)$) shows the average gap in percentage, for the cases not solved to optimality. The last five columns ($*^+(\%)$, $+(\%)$, $*^-(\%)$, $-(\%)$ e $\sim(\%)$) have the same meaning described previously, and the results are presented in percentage within each series.

$Series$	N	$OPT(\%)$	$Gap_m(\%)$	$*^+(\%)$	$+(\%)$	$*^-(\%)$	$-(\%)$	$\sim(\%)$
B	18	100.00	0.00	0.00	0.00	0.00	0.00	100.00
C	20	90.00	0.31	0.00	0.00	10.00	0.00	90.00
D	19	52.63	0.56	0.00	0.00	36.84	0.00	63.16
E	14	57.14	1.06	0.00	0.00	42.86	0.00	57.14
R	46	100.00	0.00	0.00	0.00	0.00	0.00	100.00
ALUE	13	23.08	7.00	0.00	7.69	23.08	38.46	30.77
GAP	13	84.61	14.04	0.00	0.00	15.39	0.00	84.61
TAQ	14	64.29	17.44	0.00	7.14	0.00	21.43	71.43
ALUT	07	42.86	6.05	0.00	42.86	0.00	14.28	42.86
DMXA	14	92.86	3.95	0.00	7.14	0.00	0.00	92.86
MSM	30	90.00	1.94	6.67	6.67	0.00	3.33	83.33
DIW	21	76.19	14.03	9.52	23.81	0.00	0.00	66.67

TABLE 1. Results in percentage: comparison with the algorithm of [28].

Before starting a branch & cut procedure, the algorithm of [28] uses a pre-processing phase to reduce the size of the graph. In this paper no pre-processing has been used. Although in practice this is an important phase, the authors feel that no pre-processing should be used when evaluating an algorithm.

In the D series, the instance d20 was not tested, due to its large size. In the E series, 6 out of 20 instances were not tested, due to their size. For this series, the pre-processing procedure of [28] greatly reduced size.

In the ALUE series, which is the hardest for the algorithm in this paper, the algorithm of [28] performs better in 62% of the cases. The best performance of the present algorithm is in the series MSM and DIW. In the DIW series, seven instances remained with no proven optimality; in two, optimality was proved, in the other five, the gap was reduced.

Appendix

Name	$\|V\|$	$\|E\|$	$\|T\|$	*LB*	*UB*	*Gap*(%)	$CPU_{\text{Time}}(s)$	*MSTV*	*MSTM*	*T&MV*	*PC*
b01	50	63	9	81.563	82	0.000	0.350	82	83	85	~
b02	50	63	13	82.766	83	0.000	0.440	83	84	86	~
b03	50	63	25	137.585	138	0.000	1.100	139	138	138	~
b04	50	100	9	59.000	59	0.000	0.430	59	59	62	~
b05	50	100	13	60.977	61	0.000	0.890	61	62	62	~
b06	50	100	25	121.984	122	0.000	2.740	122	124	128	~
b07	75	94	13	110.406	111	0.000	0.650	111	111	111	~
b08	75	94	19	103.139	104	0.000	1.750	104	108	104	~
b09	75	94	38	219.397	220	0.000	4.160	226	220	222	~
b10	75	150	13	85.568	86	0.000	0.650	91	86	90	~
b11	75	150	19	87.997	88	0.000	1.870	88	90	90	~
b12	75	150	38	173.913	174	0.000	8.200	174	175	177	~
b13	100	125	17	165.000	165	0.000	2.140	165	172	178	~
b14	100	125	25	235.000	235	0.000	2.980	235	238	241	~
b15	100	125	50	317.639	318	0.000	11.660	318	323	322	~
b16	100	200	17	126.997	127	0.000	3.460	127	142	136	~
b17	100	200	25	130.111	131	0.000	4.340	131	132	132	~
b18	100	200	50	217.992	218	0.000	14.490	218	219	223	~
c01	500	625	5	84.272	85	0.000	1.790	85	85	85	~
c02	500	625	10	143.528	144	0.000	6.720	144	175	144	~
c03	500	625	83	753.039	754	0.000	192.950	754	758	769	~
c04	500	625	125	1078.020	1079	0.000	449.920	1079	1079	1101	~
c05	500	625	250	1578.980	1579	0.000	5379.220	1579	1586	1587	~
c06	500	1000	5	54.055	55	0.000	2.630	55	74	55	~
c07	500	1000	10	101.809	102	0.000	7.170	107	127	102	~
c08	500	1000	83	508.900	509	0.000	216.420	516	509	521	~
c09	500	1000	125	706.987	707	0.000	4058.820	707	709	708	~
c10	500	1000	250	1092.990	1094	0.092	10501.200	1094	1094	1094	$*^{-}$
c11	500	2500	5	31.995	32	0.000	11.320	32	33	34	~
c12	500	2500	10	45.892	46	0.000	26.240	46	48	48	~
c13	500	2500	83	257.351	258	0.000	1700.650	259	261	258	~
c14	500	2500	125	322.039	323	0.000	664.410	333	323	334	~
c15	500	2500	250	555.473	556	0.000	3506.140	562	557	556	~
c16	500	12500	5	10.388	11	0.000	17.820	20	12	11	~
c17	500	12500	10	17.959	18	0.000	100.270	18	20	19	~
c18	500	12500	83	112.063	113	0.000	5043.900	114	115	113	~
c19	500	12500	125	145.031	146	0.000	3325.490	146	148	159	~
c20	500	12500	250	265.566	267	0.540	10425.400	267	267	268	$*^{-}$

Table 2. Series B and C: random graphs with Euclidean costs.

Name	$\|V\|$	$\|E\|$	$\|T\|$	*LB*	*UB*	*Gap*(%)	$CPU_{\text{Time}}(s)$	*MSTV*	*MSTM*	*T&MV*	*PC*
d01	1000	1250	5	106.000	106	0.000	8.740	106	108	107	$\sim$
d02	1000	1250	10	219.413	220	0.000	14.940	229	220	220	$\sim$
d03	1000	1250	167	1564.990	1565	0.000	4840.800	1565	1567	1565	$\sim$
d04	1000	1250	250	1934.530	1935	0.000	3370.140	1937	1940	1935	$\sim$
d05	1000	1250	500	3237.340	3252	0.453	10424.300	3266	3260	3252	$*^-$
d06	1000	2000	5	66.634	67	0.000	8.120	67	84	70	$\sim$
d07	1000	2000	10	102.458	103	0.000	19.900	103	105	103	$\sim$
d08	1000	2000	167	1071.990	1074	0.188	10427.800	1076	1079	1074	$*^-$
d09	1000	2000	250	1447.080	1449	0.133	10509.800	1454	1454	1449	$*^-$
d10	1000	2000	500	2107.840	2113	0.245	10505.100	2129	2113	2114	$*^-$
d11	1000	5000	5	29.000	29	0.000	24.140	29	31	30	$\sim$
d12	1000	5000	10	41.114	42	0.000	36.220	42	47	42	$\sim$
d13	1000	5000	167	499.960	500	0.000	10509.800	504	504	500	$\sim$
d14	1000	5000	250	665.168	667	0.275	10501.700	677	669	667	$*^-$
d15	1000	5000	500	1110.180	1116	0.524	10504.800	1136	1122	1116	$*^-$
d16	1000	25000	5	12.029	13	0.000	42.430	20	14	13	$\sim$
d17	1000	25000	10	22.121	23	0.000	165.740	31	25	23	$\sim$
d18	1000	25000	167	222.188	223	0.000	10502.900	228	223	246	$\sim$
d19	1000	25000	250	307.578	314	2.088	10500.300	329	314	315	$*^-$
e01	2500	3125	5	110.937	111	0.000	19.310	111	136	115	$\sim$
e02	2500	3125	10	214.000	214	0.000	53.400	214	218	227	$\sim$
e03	2500	3125	417	4002.990	4031	0.700	10508.400	4049	4045	4031	$*^-$
e04	2500	3125	625	5086.800	5123	0.712	10508.400	5240	5217	5123	$*^-$
e05	2500	3125	1250	8047.540	8164	1.447	10503.800	8527	8164	8212	$*^-$
e06	2500	5000	5	73.000	73	0.000	20.220	73	85	78	$\sim$
e07	2500	5000	10	145.000	145	0.000	86.360	145	145	149	$\sim$
e08	2500	5000	417	2630.210	2658	1.057	10502.000	2739	2760	2658	$*^-$
e09	2500	5000	625	3594.350	3612	0.491	10501.000	3680	3612	3737	$*^-$
e11	2500	12500	5	33.499	34	0.000	35.990	46	34	38	$\sim$
e12	2500	12500	10	67.000	67	0.000	7066.980	67	72	68	$\sim$
e13	2500	12500	417	1270.230	1295	1.950	10505.000	1443	1295	1308	$*^-$
e16	2500	62500	5	14.993	15	0.000	263.270	15	18	17	$\sim$
e17	2500	62500	10	24.879	25	0.000	760.210	25	26	26	$\sim$

TABLE 3. Series D and E: sparse random graphs with Euclidean costs.

$Name$	$\|V\|$	$\|E\|$	$\|T\|$	LB	UB	$Gap(\%)$	$CPU_{\text{Time}}(s)$	MSTV	MSTM	T&MV	PC
r01	15	22	5	186.123	187	0.000	0.060	187	189	187	~
r02	12	17	6	163.625	164	0.000	0.050	168	164	168	~
r03	28	45	7	235.970	236	0.000	0.240	236	236	237	~
r04	64	112	8	253.061	254	0.000	0.800	255	264	254	~
r05	12	17	6	225.425	226	0.000	0.100	229	229	226	~
r06	24	38	12	241.161	242	0.000	0.290	242	242	245	~
r07	30	49	12	247.420	248	0.000	0.450	251	248	254	~
r08	24	37	12	235.924	236	0.000	0.410	239	236	242	~
r09	15	22	7	164.000	164	0.000	0.130	164	172	172	~
r10	36	60	6	176.261	177	0.000	0.230	192	184	177	~
r11	30	49	6	143.275	144	0.000	0.180	144	144	147	~
r12	27	42	9	179.143	180	0.000	0.340	180	180	180	~
r13	42	71	9	149.100	150	0.000	0.500	150	150	150	~
r14	36	60	12	259.279	260	0.000	0.670	260	260	260	~
r15	100	180	14	147.100	148	0.000	2.580	150	148	148	~
r16	9	12	3	159.436	160	0.000	0.040	160	160	160	~
r17	48	82	10	199.127	200	0.000	0.600	200	200	200	~
r18	182	337	62	403.896	404	0.000	246.260	404	405	408	~
r19	168	310	14	187.954	188	0.000	5.010	190	188	190	~
r20	6	7	3	111.573	112	0.000	0.000	142	142	112	~
r21	15	22	5	192.000	192	0.000	0.050	196	196	192	~
r22	16	24	4	62.164	63	0.000	0.080	63	63	63	~
r23	16	24	4	64.045	65	0.000	0.040	68	65	65	~

TABLE 4. Series R: grids with no holes.

$Name$	$\|V\|$	$\|E\|$	$\|T\|$	LB	UB	$Gap(\%)$	$CPU_{\text{Time}}(s)$	MSTV	MSTM	T&MV	PC
r24	16	24	4	29.407	30	0.000	0.060	30	30	30	$\simeq$
r25	9	12	3	22.880	23	0.000	0.020	23	23	23	$\simeq$
r26	9	12	3	14.572	15	0.000	0.010	17	17	15	$\simeq$
r27	16	24	4	132.052	133	0.000	0.100	133	133	133	$\simeq$
r28	12	17	4	23.446	24	0.000	0.060	24	24	24	$\simeq$
r29	9	12	3	199.387	200	0.000	0.050	200	200	200	$\simeq$
r30	28	45	12	109.210	110	0.000	0.540	110	110	110	$\simeq$
r31	130	237	14	258.084	259	0.000	4.660	260	259	259	$\simeq$
r32	210	391	19	313.000	313	0.000	41.280	313	314	317	$\simeq$
r33	132	241	18	267.024	268	0.000	6.030	268	270	269	$\simeq$
r34	272	511	19	240.991	241	0.000	26.430	251	246	241	$\simeq$
r35	240	449	18	150.028	151	0.000	12.110	154	152	151	$\simeq$
r36	6	7	4	90.000	90	0.000	0.000	98	90	90	$\simeq$
r37	49	84	8	89.103	90	0.000	0.330	91	92	90	$\simeq$
r38	100	180	14	165.217	166	0.000	2.430	183	166	181	$\simeq$
r39	100	180	14	165.623	166	0.000	2.170	176	166	181	$\simeq$
r40	64	112	10	154.210	155	0.000	0.850	175	158	155	$\simeq$
r41	144	263	20	223.988	224	0.000	9.970	230	226	224	$\simeq$
r42	81	144	15	152.185	153	0.000	1.980	157	159	153	$\simeq$
r43	195	362	16	254.926	255	0.000	8.840	255	256	256	$\simeq$
r44	196	364	17	251.996	252	0.000	14.700	256	252	253	$\simeq$
r45	270	507	19	220.000	220	0.000	147.370	220	224	224	$\simeq$
r46	16	24	16	149.137	150	0.000	0.400	150	150	150	$\simeq$

TABLE 5. Series R: grids with no holes.

Name	$\|V\|$	$\|E\|$	$\|T\|$	*LB*	*UB*	*Gap*(%)	$CPU_{\text{Time}}(s)$	*MSTV*	*MSTM*	*T&MV*	*PC*
alue2087	1244	1971	34	1049.000	1049	0.000	1433.910	1054	1063	1049	$\sim$
alue2105	1220	1858	34	1031.002	1032	0.000	734.340	1032	1032	1032	$\sim$
alue3146	3626	5869	64	2004.110	2242	11.870	10520.200	2380	3346	2242	–
alue5067	3524	5560	68	2586.000	2586	0.000	10519.800	2586	3012	2623	$\sim$
alue5345	5179	8165	68	3238.840	3566	10.101	10550.500	3781	4463	3566	–
alue5623	4472	6938	68	3204.560	3480	8.595	10546.300	3839	3913	3480	–
alue5901	11543	18429	68	3322.830	4011	20.710	10761.600	5221	4831	4011	–
alue6179	3372	5213	67	2386.330	2457	2.961	10531.200	2567	2707	2457	$*^{-}$
alue6457	3932	6137	68	2997.970	3067	2.250	10554.200	3398	3450	3067	–
alue6735	4119	6696	68	2651.440	2704	1.982	10547.200	2776	2887	2704	$*^{-}$
alue6951	2818	4419	67	2381.470	2403	0.904	10516.200	2465	2445	2403	$*^{-}$
alue7066	6405	10454	16	2194.760	2275	3.655	10520.300	2540	2340	2275	+
alue7229	940	1474	34	823.202	824	0.000	83.460	827	899	824	$\sim$
taq0014	6466	11046	128	4295.260	5459	27.094	10457.700	6994	6681	5459	–
taq0023	572	963	11	621.000	621	0.000	207.370	621	623	623	$\sim$
taq0365	4186	7074	22	1911.110	1914	0.151	10506.500	1981	1963	1914	+
taq0377	6836	11715	136	5403.270	6551	21.241	10489.800	8242	8130	6551	–
taq0431	1128	1905	13	896.002	897	0.000	382.910	897	946	897	$\sim$
taq0631	609	932	10	581.000	581	0.000	85.950	588	588	581	$\sim$
taq0739	837	1438	16	847.500	848	0.000	8537.940	848	885	848	$\sim$
taq0741	712	1217	16	846.998	847	0.000	699.930	847	856	855	$\sim$
taq0751	1051	1791	16	938.420	939	0.000	10501.200	939	949	939	$\sim$
taq0891	331	560	10	318.124	319	0.000	10.300	331	321	319	$\sim$
taq0903	6163	10490	130	4284.170	5195	21.260	10486.800	6809	6783	5195	–
taq0910	310	514	17	369.165	370	0.000	8.690	427	407	370	$\sim$
taq0920	122	194	17	209.077	210	0.000	2.100	217	212	210	$\sim$
taq0978	777	1239	10	565.175	566	0.000	48.200	649	613	566	$\sim$

TABLE 6. Series ALUE and TAQ: grids with holes (VLSI circuits).

Name	$\|V\|$	$\|E\|$	$\|T\|$	*LB*	*UB*	*Gap*(%)	$CPU_{\text{Time}}(s)$	*MSTV*	*MSTM*	*T&MV*	*PC*
gap1307	342	552	17	548.896	549	0.000	15.180	549	551	551	$\sim$
gap1413	541	906	10	456.033	457	0.000	14.070	464	478	457	$\sim$
gap1500	220	374	17	253.108	254	0.000	10.490	301	266	254	$\sim$
gap1810	429	702	17	481.932	482	0.000	19.630	496	482	489	$\sim$
gap1904	735	1256	21	762.960	763	0.000	48.900	763	763	785	$\sim$
gap2007	2039	3548	17	1099.000	1104	0.455	10001.600	1104	1114	1104	$*^{-}$
gap2119	1724	2975	29	1243.010	1244	0.000	3073.950	1244	1244	1244	$\sim$
gap2740	1196	2084	14	744.997	745	0.000	1121.180	758	775	745	$\sim$
gap2800	386	653	12	385.477	386	0.000	21.170	401	400	386	$\sim$
gap2975	179	293	10	244.920	245	0.000	2.620	245	245	245	$\sim$
gap3036	346	583	13	456.985	457	0.000	15.740	457	457	465	$\sim$
gap3100	921	1558	11	639.999	640	0.000	605.100	656	649	640	$\sim$
gap3128	10393	18043	104	3453.180	4407	27.622	10445.700	5462	5724	4407	$*^{-}$
alut0787	1160	2089	34	981.005	982	0.000	1299.150	982	1064	982	$\sim$
alut0805	966	1666	34	957.001	958	0.000	2982.070	958	958	963	$\sim$
alut1181	3041	5693	64	2334.090	2377	1.838	10420.500	2612	2993	2377	$+$
alut2010	6104	11011	68	3190.870	3355	5.144	10472.500	3912	4390	3355	$-$
alut2288	9070	16595	68	3541.740	3892	9.889	10409.700	5825	4984	3892	$+$
alut2566	5021	9055	68	2952.420	3117	7.341	10449.900	3326	4559	3117	$+$
alut2764	387	626	34	639.995	640	0.000	53.450	642	663	640	$\sim$

TABLE 7. Series GAP and ALUT: grids with holes (VLSI circuits).

$Name$	$\|V\|$	$\|E\|$	$\|T\|$	LB	UB	$Gap(\%)$	$CPU_{\text{Time}}(s)$	$MSTV$	$MSTM$	$T\&MV$	PC
dmxa0296	233	386	12	343.062	344	0.000	3.720	416	366	344	$\sim$
dmxa0368	2050	3676	18	1016.990	1017	0.000	4696.280	1017	1031	1021	$\sim$
dmxa0454	1848	3286	16	913.997	914	0.000	1535.710	941	939	914	$\sim$
dmxa0628	169	280	10	274.959	275	0.000	3.710	275	277	277	$\sim$
dmxa0734	663	1154	11	506.000	506	0.000	343.070	507	523	506	$\sim$
dmxa0848	499	861	16	593.997	594	0.000	40.180	599	594	599	$\sim$
dmxa0903	632	1087	10	580.000	580	0.000	126.850	580	595	585	$\sim$
dmxa1010	3983	7108	23	1436.290	1493	3.948	10503.600	1650	1619	1493	$+$
dmxa1109	343	559	17	453.954	454	0.000	15.310	454	454	462	$\sim$
dmxa1200	770	1383	21	749.045	750	0.000	94.750	754	750	750	$\sim$
dmxa1304	298	503	10	310.999	311	0.000	7.260	311	311	312	$\sim$
dmxa1516	720	1269	11	507.844	508	0.000	24.010	508	508	513	$\sim$
dmxa1721	1005	1731	18	779.998	780	0.000	132.400	780	781	781	$\sim$
dmxa1801	2333	4137	17	1364.002	1365	0.000	3226.410	1390	1365	1365	$\sim$
diw0234	5349	10086	25	1995.970	1996	0.000	10512.800	2192	2192	1996	$\sim$
diw0250	353	608	11	349.099	350	0.000	5.590	431	350	350	$\sim$
diw0260	539	985	12	467.152	468	0.000	54.890	751	475	468	$\sim$
diw0313	468	822	14	396.063	397	0.000	14.390	409	399	397	$\sim$
diw0393	212	381	11	301.097	302	0.000	3.960	302	302	302	$\sim$
diw0445	1804	3311	33	1362.030	1363	0.000	7268.660	1414	1458	1363	$\sim$
diw0459	3636	6789	25	1361.020	1362	0.000	7013.390	1808	1554	1362	$\sim$
diw0460	339	579	13	344.002	345	0.000	5.040	376	388	345	$\sim$
diw0473	2213	4135	25	1097.030	1098	0.000	3891.350	1254	1098	1100	$\sim$
diw0487	2414	4386	25	1423.970	1424	0.000	8630.050	1434	1438	1424	$\sim$
diw0495	938	1655	10	615.223	616	0.000	92.040	1036	616	634	$\sim$
diw0513	918	1684	10	604.000	604	0.000	140.710	609	702	604	$\sim$
diw0523	1080	2015	10	560.112	561	0.000	58.590	647	587	561	$\sim$
diw0540	286	465	10	373.088	374	0.000	3.370	397	431	374	$\sim$
diw0559	3738	7013	18	1489.940	1570	5.373	10503.100	1778	1860	1570	$+$
diw0778	7231	13727	24	2101.110	2173	3.421	10420.300	2933	2570	2173	$+$
diw0779	11821	22516	50	3346.010	4598	37.417	10404.200	7349	6422	4598	$+$
diw0795	3221	5938	10	1549.140	1550	0.000	2604.880	1757	2072	1550	$*^{+}$
diw0801	3023	5575	10	1586.020	1587	0.000	2793.660	1602	1795	1587	$*^{+}$
diw0819	10553	20066	32	3038.360	3404	12.034	10454.900	5937	4581	3404	$+$
diw0820	11749	22384	37	3789.430	4240	11.890	10428.500	6420	6634	4240	$+$

TABLE 8. Series DMXA and DIW: grids with holes (VLSI circuits).

Name	$\|V\|$	$\|E\|$	$\|T\|$	*LB*	*UB*	*Gap*(%)	$CPU_{\text{Time}}(s)$	*MSTV*	*MSTM*	*T&MV*	*PC*
msm0580	338	541	11	466.999	467	0.000	8.940	467	467	480	$\simeq$
msm0654	1290	2270	10	822.989	823	0.000	1153.420	833	833	823	$\simeq$
msm0709	1442	2403	16	883.002	884	0.000	396.170	897	896	884	$\simeq$
msm0920	752	1264	26	805.053	806	0.000	93.590	806	806	808	$\simeq$
msm1008	402	695	11	494.000	494	0.000	169.930	498	498	494	$\simeq$
msm1234	933	1632	13	550.000	550	0.000	733.080	550	563	563	$\simeq$
msm1477	1199	2078	31	1067.010	1068	0.000	1108.140	1086	1089	1068	$\simeq$
msm1707	278	478	11	563.037	564	0.000	4.860	564	564	564	$\simeq$
msm1844	90	135	10	187.947	188	0.000	1.180	188	215	188	$\simeq$
msm1931	875	1522	10	603.002	604	0.000	168.070	634	735	604	$\simeq$
msm2000	898	1562	10	593.179	594	0.000	88.080	712	670	594	$\simeq$
msm2152	2132	3702	37	1590.000	1590	0.000	6868.560	1603	1631	1590	$\simeq$
msm2326	418	723	14	398.454	399	0.000	10.780	407	399	399	$\simeq$
msm2492	4045	7094	12	1456.480	1459	0.173	10501.800	1525	1459	1459	$+$
msm2525	3031	5239	12	1289.980	1290	0.000	5523.380	1417	1310	1290	$\simeq$
msm2601	2961	5100	16	1439.990	1440	0.000	6929.590	1450	1583	1440	$*^{+}$
msm2705	1359	2458	13	713.120	714	0.000	155.710	805	727	714	$\simeq$
msm2802	1709	2963	18	925.014	926	0.000	2251.200	933	933	926	$\simeq$
msm2846	3263	5783	89	3036.580	3153	3.834	10442.700	3384	3548	3153	$-$
msm3277	1704	2991	12	868.024	869	0.000	1755.490	930	942	869	$\simeq$
msm3676	957	1554	10	606.001	607	0.000	82.250	627	622	607	$\simeq$
msm3727	4640	8255	21	1375.020	1376	0.000	6279.820	1557	1409	1376	$\simeq$
msm3829	4221	7255	12	1564.490	1593	1.822	10502.600	1973	1729	1593	$+$
msm4038	237	390	11	352.029	353	0.000	4.460	356	353	358	$\simeq$
msm4114	402	690	16	392.292	393	0.000	9.440	429	393	393	$\simeq$
msm4190	391	666	16	380.106	381	0.000	24.380	423	381	381	$\simeq$
msm4224	191	302	11	310.001	311	0.000	3.100	311	311	311	$\simeq$
msm4312	5181	8893	10	2015.980	2016	0.000	8943.140	2016	2095	2016	$*^{+}$
msm4414	317	476	11	407.141	408	0.000	11.910	437	408	408	$\simeq$
msm4515	777	1358	13	630.000	630	0.000	231.700	640	640	630	$\simeq$

TABLE 9. Series MSM: grids with circuits (VLSI circuits).

REFERENCES

[1] Y. P. Aneja. An integer linear programming approach to the Steiner problem in graphs. *Networks*, 10:167–178, 1980.

[2] L. Bahiense, N. Maculan and C. Sagastizábal. On the convergence of the volume algorithm. To appear.

[3] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87:385-399, 2000.

[4] F. Barahona and F. Chudak. Solving large scale uncapacitated facility location problems. In: *Approximation and Complexity in Numerical Optimization*, P. Pardalos (ed.), Kluwer, 48-62, 2000.

[5] J. Beasley. An sst-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.

[6] J. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14:147–159, 1984.

[7] P. Bermann and V. Ramaiyer. Improved approximations for the Steiner tree problem. *J. Alg.*, 17:381–408, 1994.

[8] S. Chopra, E. R. Gorres and M. R. Rao. Solving the Steiner tree problem in graphs using branch-and-cut. *ORSA J. Comput.*, 4:320–335, 1992.

[9] S. Chopra and M. R. Rao. The Steiner tree problem I: formulations, compositions and extension of facets. *Mathematical Programming*, 64:209–229, 1994.

[10] S. Chopra and M. R. Rao. The Steiner tree problem II: properties and classes of facets. *Mathematical Programming*, 64:231–246, 1994.

[11] A. Claus and N. Maculan. Une nouvelle formulation du Problème de Steiner sur un graphe. Technical Report 280, Centre de Recerche sur les Transports, Université de Montréal, 1983.

[12] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[13] K. A. Dowsland. Hill-climbing, simulated annealing and the Steiner tree problem in graphs. *Eng. Opt.*, 17:91–107, 1991.

[14] S. E. Dreyfus and R. A. Wagner. The Steiner Problem in Graphs. *Networks*, 1:195–207, 1971.

[15] C. W. Duin and S. Voß. Steiner tree heuristics: a survey. In M. Salomon H. Dyckhoff, U. Derigs and eds. H. C. Tijms, editors, *Operations Research Proceedings 1993*, pages 485–496. Springer Verlag, Berlin, 1994.

[16] C. W. Duin and S. Voß. Efficient path and vertex exchange in Steiner tree algorithms. *Networks*, 29:89–105, 1997.

[17] H. Esbensen. Computing near-optimal solutions to the Steiner tree problem in a graph using a genetic algorithm. *Networks*, 26:173–185, 1995.

[18] M. R. Garey and D. S. Johnson. The Rectilinear Steiner tree problem is *NP*-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[19] M. X. Goemans and Y s. Myung. A catalog of Steiner tree formulations. *Networks*, 23:19–28, 1993.

[20] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal of Computing*, 24:296–317, 1995.

[21] M. Held, P. Wolfe and H. P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[22] J. B. Hiriart-Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms*. Springer Verlag, 1991.

[23] K. Holmberg and J. Hellstrand. Solving the uncapacitated network design problem by a Lagrangian heuristic and branch-and-bound. *Operations Research*, 46:247–259, 1998.

[24] F. K. Hwang, D. S. Richards and P. Winter. The Steiner tree problem. *Annals of Discrete Mathematics*, volume 53. North Holland, Amsterdam, 1992.

[25] F. K. Hwang and D. S. Richards . Steiner tree problems. *Networks*, 22:55–89, 1992.

[26] A. Kapsalis V. J. Rayward-Smith and G. D. Smith. Solving the graphical Steiner tree problem using genetic algorithms. *Journal of the Operational Research Society*, 44:397–406, 1993.

[27] R. M. Karp. Reducibility among combinatorial problems. R. E. Miller and J. W. Thatcher, eds., *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[28] T. Koch and A. Martin. Solving Steiner Tree Problems in Graphs to Optimality. *Networks*, 32:207–232, 1998.

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

[30] C. Lemaréchal. An extension of Davidon methods to nondifferentiable problems. *Mathematical Programming Study*, 3:95–109, 1975.

[31] T. Lengauer. Combinatorial Algorithms for integrated circuit layout. Wiley, Chichester.

[32] A. Lucena. Steiner problem in graphs: Lagrangian relaxation and cutting-planes. *COAL Bull.*, 21:2–7, 1992.

[33] A. Lucena. and J. Beasley A branch-and-cut algorithm for the Steiner problem in graphs. *Networks*, 31:39–59, 1998.

[34] N. Maculan. The Steiner Problem in Graphs. *Annals of Discrete Mathematics*, 31:185–212, 1987.

[35] T. L. Magnanti and T. Wong. Network design and transportation planning: models and algorithms. *Transp. Science*, 18:1–55, 1984.

[36] S. L. Martins, C. C. Ribeiro and M .C. Souza. A parallel GRASP for the Steiner problem in graphs. *Lecture Notes in Computer Science*, 1457:285–297, 1967. Proceedings of IRREGULAR'98 - 5th International Symposium on Solving Irregularly Structured Problems in Parallel.

[37] S. L. Martins, P. M. Pardalos, M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptative search procedures for the Steiner problem in graphs. S. Rjasekaram P. M. Pardalos and J. Rolim, eds., *Randomization Methods in Algorithm Design*, pages 133–145. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 43, 1998.

[38] C. Pornavalai N. Shiratori and G. Chakraborty. Neural network for optimal Steiner tree computation. *Neural Process. Lett.*, 3:139–149, 1996.

[39] V. J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.

[40] C. C. Ribeiro and M .C. Souza. Tabu search for the Steiner problem in graphs. 1997. Working paper.

[41] N. Shor. *Minimization methods for non-differentiable functions*. Springer-Verlag, Berlin, 1985.

[42] J. Soukup and W. F. Chow. Set of test problems for the minimum length connection networks. ACM/SIGMAP *Newsletters*, 15:48–51, 1973.

[43] H. Takahashi and A. Matsuyama. An approximated solution for the Steiner tree problem in graphs. *Math. Japonica*, 254:573–577, 1980.

[44] S. Voß. Steiner's problem in graphs: heuristic methods. *Discrete Applied Mathematics*, 40:45–72, 1992.

[45] D. P. Williamson, M. X. Goemans, M. Mihail and V. V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15 (3):435–454, 1995.

[46] P. Winter. Steiner problems in networks: a survey. *Networks*, 17:129–167, 1987.

[47] P. Winter and J. M. Smith. Path-distance heuristics for the Steiner problem in undirected networks. *Algorithmica*, 7:309–327, 1992.

[48] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Study*, 3:145–173, 1975.

[49] R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[50] J. Xu, S. Y. Chiu and F. Glover. A probabilistic tabu search for the telecommunications network design. *Combinatorial Optimization: Theory and Pratice*, 1:69–94, 1996.

[51] J. Xu, S. Y. Chiu and F. Glover. Using tabu search to solve Steiner tree-star problems in telecommunications network design. *Telecommunication Systems*, 6:117–125, 1996.

(L. Bahiense) UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COPPE-SISTEMAS E COMPUTAÇÃO, P.O. BOX 68511, RIO DE JANEIRO, RJ 21945-970, BRAZIL

Supported by grant from Brazilian agency CNPq

E-mail address, L. Bahiense: laura@cos.ufrj.br

(F. Barahona) IBM T. J. Watson research Center, Yorktown Heights, NY 10589, USA (Corresponding Author)

E-mail address, F. Barahona: barahon@us.ibm.com

(O. Porto) PUC-Rio, Dept. de Engenharia Elétrica, Rua Marquês de São Vicente 225, Predio Cardeal Leme, Sala 401, CEP 22453-900, Rio de Janeiro, RJ, Brazil

Partially supported by Brazilian agency CNPq, grant 301261/91-1

E-mail address, O. Porto: oscar@ele.puc-rio.br