RAIRO Operations Research
RAIRO Oper. Res. **40** (2006) 53–73
DOI: 10.1051/ro:2006010

BRANCH AND CUT BASED ON THE VOLUME ALGORITHM: STEINER TREES IN GRAPHS AND MAX-CUT

Francisco Barahona[1] and László Ladányi[1]

Abstract. We present a Branch-and-Cut algorithm where the volume algorithm is applied instead of the traditionally used dual simplex algorithm to the linear programming relaxations in the root node of the search tree. This means that we use fast approximate solutions to these linear programs instead of exact but slower solutions. We present computational results with the Steiner tree and Max-Cut problems. We show evidence that one can solve these problems much faster with the volume algorithm based Branch-and-Cut code than with a dual simplex based one. We discuss when the volume based approach might be more efficient than the simplex based approach.

1. Introduction

Since the early eighties, the Branch-and-Cut (B&C) algorithm has been used for a variety of combinatorial optimization problems, see [11] for a survey. In B&C, families of cutting planes are added to the formulation to tighten the linear programming relaxation at each search tree node. Traditionally these linear programming (LP) relaxations are solved by the dual simplex method, since the dual simplex method is well suited to warmstart the subsequent LP relaxations. Many authors have reported difficulties due to degeneracy and to the number of dense rows in the LPs coming from combinatorial problems. In many cases most of the computing time in B&C methods is devoted to solve these LPs. The main result of our paper is that when solving the LP relaxations it can be advantageous

Received May 19, 2005. Accepted December 8, 2005.

[1] IBM T. J. Watson research Center, Yorktown Heights, NY 10589, USA; barahon@us.ibm.com; ladanyi@us.ibm.com

to replace the dual simplex method with a fast approximate method, the volume algorithm [6].

Note that our aim was to compare B&C codes based on the simplex method and the volume algorithm when all else are equal. The speed of a B&C code very heavily depends on the quality of cut generation and primal heuristics. While we have implemented reasonable procedures, our main focus was on the LP engine comparison. Our max-cut code happens to be very competitive while the Steiner tree code is slower than the one presented in [28].

The volume algorithm (VA) is an extension of the subgradient method that produces an approximate primal as well as a dual solution and lower bound for a minimization problem. Note that the primal solution delivered by the VA algorithm to an LP relaxation is a fractional approximate solution, which can then be used in primal heuristics to produce integer solutions. The traditional subgradient method produces only a dual solution and a lower bound for the LP relaxation. Like the subgradient method, the VA can be warmstarted using a dual feasible solution.

The subgradient method has been extensively used in combinatorial optimization, in cutting plane procedures it has been used under the name of "relax-and-cut", see [18, 22, 30, 31]. In relax-and-cut the cutting planes are produced based on a solution to the Lagrangean problem (see (4) in Sect. 2). This is a point usually outside the optimal face of the LP relaxation. The difference with our procedure is that we produce the cutting planes based on a vector near the center of the optimal face. We claim that this makes the cutting planes very effective. A further discussion appears in Section 5. Note that a B&C code using an interior point method based LP engine should exhibit similar behavior as one based on the VA, since both LP engines produce primal solutions near the center of the optimal face. Indeed, Mitchell [33] has shown that this is the case. However, interior point methods suffer from the lack of warmstarting ability unlike the VA, which can be warmstarted with the previous dual feasible solution.

For a B&C code to work efficiently with an LP solver the following conditions have to be met. The LP solver must provide a lower bound on the LP optimum (for bounding); it must be possible to warmstart the LP solver (for efficiency) and a primal solution of the linear program must be produced by the LP solver (for cut generation and branching). This primal solution need not necessarily be optimal; it is up to the the cut generation routine to separate the fractional solution from the integer programming (IP) polytope. If the cut generation routine cannot separate the primal solution (exact or approximate) from the IP polytope then branching is performed. The VA satisfies all three conditions, therefore a VA based B&C code should be successful for any problem which has an LP relaxation the VA is good at approximately solving. Besides the problem classes discussed in this paper set partitioning type problems also fall into this category (see [7]). We believe that other combinatorial optimization problems (where the matrix is a 0/1 matrix and the problem is dual and/or primal degenerate) should also exhibit this behavior.

Our B&C implementation uses BCP, a state of the art Branch-Cut-Price framework designed to aid problem specific mixed integer programming implementations. BCP handles all general B&C related tasks allowing us to focus on problem specific issues like cut generation. Also, using BCP made the comparison of the dual simplex and VA based B&C easy, since BCP already has interfaces to both of these methods for solving the LP relaxations.

We have chosen two combinatorial problems for which simplex-based B&C algorithms are used as the common practice, these are the Max-Cut and the Steiner tree problems. We show that although the almost universal choice is the dual simplex method, our VA based code outperforms the other for the problem classes studied here.

We found that the VA was useful to deal with the LP relaxation at the root node of the B&C tree, we switched to the simplex method once branching was needed. We believe that this approach is useful for many other combinatorial problems for which solving the LP relaxation at the root node is computationally very intensive. There are other alternatives to the VA, for instance some experiments based on the bundle method have been presented in [20]. We used CLP an open source LP solver, we believe that our findings are independent of the LP solver used.

This paper is organized as follows. In Section 2 we describe the volume algorithm. In Section 3 we describe the BCP framework. Sections 4 and 5 are devoted to the Steiner tree problem in graphs and to computational results. Similarly, Sections 6 and 7 deal with the Max-Cut problem. Finally, conclusions and insights into when it might be advantageous to jump-start the branch-and-cut algorithm with the volume algorithm are given in Section 8.

2. Solving the linear programming relaxations with the volume algorithm

We use Lagrangean relaxation to deal with the arising linear programming relaxations during the B&C run. Lagrangean relaxation has been used in combinatorial optimization since the seminal papers of Held and Karp [23, 24] and Held, Wolfe and Crowder [25]. Lagrangean relaxation for solving LPs has been implemented *via* the subgradient algorithm. We use an extension of the subgradient algorithm, the volume algorithm [6]. This method produces not only a lower bound and a dual solution, but also an approximate (fractional) primal solution to the linear program. The convergence properties of the VA have been studied in [4].

At any stage of the the B&C algorithm the linear relaxation of the combinatorial problem can be stated as

$$\min cx \tag{1}$$

$$Ax \geq b \tag{2}$$

$$0 \leq x \leq 1. \tag{3}$$

We use Lagrangian relaxation on inequalities (2). This approach has also been used in [7] for solving set partitioning problems. For a vector of dual multipliers $\pi \geq 0$, a lower bound on the minimum of (1) is given by

$$L(\pi) = \min_x (c - \pi A)\, x + \pi b \qquad (4)$$
$$0 \leq x \leq 1.$$

In order to maximize the function $L(\cdot)$ we apply the volume algorithm sketched out in Algorithm 1, see also [6].

Algorithm 1: Volume algorithm

1 $t \leftarrow 1$
2 Initilize $\bar{\pi} \geq 0$.
3 Solve (4) with $\pi = \bar{\pi}$ to obtain $\bar{x} = x^0$ and $\bar{z} = L(\bar{\pi})$.
4 **while** *new iteration should be done* **do**
5 Compute $v^t = b - A\bar{x}$ and $\pi' = \bar{\pi} + sv^t$ for a step size s given by (6).
6 $\pi_i^t \leftarrow \max(0, \pi_i')$
7 Solve (4) with $\pi = \pi^t$ to get its solution x^t and $z^t = L(\pi^t)$.
8 $\bar{x} \leftarrow \alpha x^t + (1-\alpha)\bar{x}$, where $0 < \alpha < 1$.
9 **if** $z^t > \bar{z}$ **then**
10 $\bar{\pi} \leftarrow \pi^t, \quad \bar{z} \leftarrow z^t$
11 $t \leftarrow t + 1$

Notice that in Step 10 we update $\bar{\pi}$ only if $z^t > \bar{z}$, so this is an ascent method. We are trying to mimic the bundle method [29], but we want to avoid the extra effort of solving a quadratic problem at each iteration.

One difference between this and the subgradient algorithm is the use of the primal update formula in Step 8. If $x^0, \ldots, x^t$ is the sequence of vectors produced by problem (4), then VA yields

$$\bar{x} = \alpha x^t + (1-\alpha)\alpha x^{t-1} + \cdots + (1-\alpha)^t x^0. \qquad (5)$$

So we should look at $\bar{x}$ as a convex combination of $\{x^0, \ldots, x^t\}$. The assumption that this sequence approximates an optimal solution of (1)–(3) is based on a theorem in linear programming duality that appears in [6]. This says that the values of the primal variables correspond to the volumes below the faces of the dual problem. With the primal update we estimate these volumes. Notice the exponential decrease of the coefficients of this convex combination; later vectors thus receive much larger weights than earlier ones. At every iteration the direction of movement is being updated based on $\bar{x}$, so this is a method with "memory" that does not have the same zig-zagging behavior as the subgradient method.

Here the formula for the step size is

$$s = \lambda \frac{T - \bar{z}}{\|v^t\|^2}, \qquad (6)$$

where λ is a number between 0 and 2, and T is a *target* value. We start with a small value for T, and each time that $\bar{z} \geq 0.95\,T$, we increase T to $T = 1.05\,\bar{z}$. In our implementation we set the value of λ based on counting how many iterations of the following types are encountered:

Red: Each time that we do not find an improvement (*i.e.* $z^t \leq \bar{z}$), we call this iteration red. A sequence of red iterations suggests the need for a smaller step-size.

Yellow: If $z^t > \bar{z}$ we compute

$$d = v^t \cdot \left(b - Ax^t\right).$$

If $d < 0$ it means that a longer step in the direction v^t would have given a smaller value for z^t, we call this iteration yellow.

Green: If $d \geq 0$ we call this iteration green. A green iteration suggests the need for a larger step-size.

At each green iteration, we multiply λ by 1.1. If the result is greater than 2, we set $\lambda = 2$. After two consecutive yellow iterations we also multiply λ by 1.1 and we set it to $\min\{\lambda, 2\}$. After a sequence of 20 consecutive red iterations, we multiply λ by 0.66, unless $\lambda < 0.0005$, in which case we keep it constant.

The value of α in the primal update formula is chosen as the solution of the following 1-dimensional problem:

$$\min \left\| b - A\left(\alpha x^t + (1-\alpha)\bar{x}\right)\right\| \tag{7}$$

subject to

$$u/10 \leq \alpha \leq u. \tag{8}$$

The value u is originally set to 0.1 and then every 100 iterations we check if $\bar{z}$ has increased by at least 1%. If not, we divide u by 2, unless u is already less than 10^{-5}, in which case it is kept constant. This choice of α is very similar to the one proposed in [38]; the difference is in the bounds $u/10$ and u.

3. The Branch-and-Cut framework

We used BCP, a Branch-and-Cut-and-Price framework. BCP handles all general B&C related tasks allowing us to focus on problem specific issues like cut generation. Also, using BCP made the comparison of the dual simplex and VA based B&C easy, since BCP already has interfaces to both of these methods for solving the LP relaxations. To solve LPs with the simplex method we have used CLP [19]. All these codes (an implementation of the Volume Algorithm, CLP and BCP) are open source codes and available from the COIN-OR project [13].

For the Steiner tree problem, when we branched, on one side we forced a non-terminal node to become a terminal, on the other branch we forced that node to not be included in the Steiner tree. We used the *strong branching* technique introduced in [2]. Each time, five nodes were used as branching candidates.

For the max-cut problem we branched on a fractional variable. We also used strong branching, choosing five variables close to 0.5 as branching candidates.

We also used *reduced cost fixing*, see [34]. Namely, given a dual vector $\bar{\pi}$, let $L(\bar{\pi})$ be the lower bound given by (4). Let UB be an upper bound and denote $\bar{c} = c - \bar{\pi}A$. With this notation if $L(\bar{\pi}) + \bar{c}_j > UB$ for an index j then $x_j = 0$ can be fixed and, respectively, if $L(\bar{\pi}) - \bar{c}_j > UB$ then $x_j = 1$ can be fixed.

We allowed a maximum of 2000 inequalities to be added at any particular iteration. An inequality was removed if its dual variable had been zero for two consecutive iterations and the lower bound had increased.

Branching was used each time the gap between lower and upper bounds did not decrease by at least 0.1% in the last three iterations.

In the volume based code we switched to the simplex method when tailing off (through several consecutive iterations the gap between the lower and upper bounds does not shrink sufficiently) in the root node was detected and from then on we have used the simplex method. This is needed, since the approximate lower bound given by the volume algorithm is 1–2% off of the exact LP value and this inaccuracy prevents proving optimality.

4. Steiner tree problems in graphs

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 an acyclic connected subgraph of G that spans T. Let $c_{ij} \geq 0$ for each edge $ij \in E$, be an edge-cost. The *Steiner tree Problem in graphs* asks for the Steiner tree of minimum total edge cost. A natural formulation comes from looking at cuts in the graph that separate terminals and write an inequality saying that at least one edge in the cut should be taken. It has been shown in [21] that the "cut formulation" associated with a directed graph gives a stronger linear programming relaxation than a similar formulation associated with an undirected graph. For that reason we work with directed graphs, we choose one terminal as a *root* r and look for a Steiner arborescence. The linear programming relaxation is

$$\min cx \tag{9}$$

$$\sum_{i \notin S,\ j \in S} x_{ij} \geq 1, \text{ for all } S \subset V,\ r \notin S,\ S \cap T \neq \emptyset \tag{10}$$

$$0 \leq x \leq 1. \tag{11}$$

We deal with this relaxation in a cutting plane fashion. This formulation was proposed in [1] and has been used in [12,28]. An equivalent formulation has been used in [32]. Computational results using the VA with an equivalent multicommodity flow formulation have been presented in [3].

Although the separation problem for inequalities (10) reduces to a minimum st-cut problem, we use a faster heuristic as follows. Let $\bar{x}$ be the current solution vector in the cutting plane procedure. Define the weights on the arcs as

$w_{ij} = d_{ij}(1 - \bar{x}_{ij})$ (we will specify the multiplier d_{ij} later) and find a minimum weight arborescence A. From A we derive cutting planes in two different ways. First, for each arc of A we remove it, thus dividing the node set of the graph into two parts, and test whether the associated inequality is violated. Second, while we execute Edmonds' [17] algorithm for finding a minimum weight arborescence we repeatedly contract cycles creating a sequence of supernodes. We always test whether the inequality corresponding to the original nodes in a newly created supernode is violated or not.

For selecting the multipliers d_{ij} we employ three different methods: set all of them to 1, set $d_{ij} = c_{ij}$ and set $d_{ij} = \bar{c}_{ij}$, the reduced cost of variable x_{ij}. We have found that we got the best results when we generated cuts based on each setting; none of them could be left out without taking a performance hit.

Another advantage of using this heuristic instead of exact cut generation (besides much faster execution) is that we can also derive upper bounds simultaneously with the cut generation. Given an arborescence A we construct a Steiner tree by simply recursively deleting all arcs leading to leaves that are non-terminal nodes.

To compensate for possible misses of our heuristic we do run the exact cut generation routine for every terminal node that is not separated from the root by any of the heuristically generated cuts. In our experience this rarely had to be done.

Another way to produce upper bounds is based on a heuristic due to Takahashi-Matsuyama [36] and is described in Algorithm 2. We have used an extended version of this algorithm (see, e.g., [28]) as follows:

- Given a starting vertex v, run the Takahashi-Matsuyama heuristic for the digraph $D = (V, A)$ with arc weights $w_{ij} = (1 - \bar{x}_{ij})c_{ij}$. The resulting Steiner tree is T.
- Find a minimum spanning tree in the subgraph induced by the vertices included in the tree obtained in the previous step, considering original costs c_{ij} as weights.
- Prune all non-terminal leaves.

Algorithm 2: Takahashi-Matsuyama Steiner tree heuristics

```
1 Chose an initial terminal vertex v.
2 Let T be a tree containing only v.
3 while there exists a terminal node t ∉ T do
4 |   foreach terminal node t ∉ T do
5 |   |   Compute the shortest path from t to T.;
6 |   Add the terminal node closest to T and the corresponding path to T.
```

Since this heuristic is computationally expensive, we ran it every 5 iterations of the cutting plane procedure.

5. Computational results on the Steiner tree problem

We ran all experiments on an IBM RS/6000 44P Model 270 workstation with a 375 MHz processor and 1GB of core memory.

Our implementation is very similar to the one presented in [28], the main difference is that we use the VA to handle the linear programs. In Table 1 we present a comparison between our simplex based code and our volume based code. We took instances from *Steinlib* [27], a library for Steiner tree problem in graphs. We did not treat the instances in Tables VI and VII of [28] because they were solved in [28] with a different technique. This consisted of running a branch and cut code to fix many variables and then restart the code from scratch when a certain percentage of the variables can be fixed. From the remaining problems treated in [28] we chose those instances that took more than 50 seconds to solve with either of our codes. A few problems (alue7065, alue7080, alut2610, and alut2625) were too big even after preprocessing to fit into our matchine. We have used a pre-processing code, obtained from Koch, which is a successor of the one used in [28]. Since the publication of [28] much stronger preprocessing methods were designed (see [37] and [35]). With those methods many problems in Steinlib can be reduced to near trivial size. However, we believe that preprocessing is orthogonal to comparing the volume and simplex based branch and cut methods, thus we have not implemented these very sophisticated preprocessing techniques. Therefore our computational results are not comparable to the ones in [37] and [35].

In the first column of Table 1 we have the name of the instance, then we present the number of nodes, number of edges, and number of terminals after preprocessing. These numbers are sometimes different than those in [28] since the preprocessing code has been updated since the publication of that paper. Then we present the time, lower bound, upper bound, the number of search tree nodes and the number of iterations in the root node for the volume based code. Finally we have the same data for the simplex-based code. Note that two numbers are listed for the number of iterations in the root node for the volume based code. The first is the total number of iterations while the second is the number of times we used the volume algorithm to solve the LP relaxation. We mark with a bullet (•) all cases where we have a proof of optimality. We had set a limit of 3 hours. Since the code checked the time only after solving each LP, the times reported are sometimes slightly larger than 10 800 s.

Observe that the volume based code is nearly always faster than the simplex based one, frequently significantly faster. The number of instances solved to optimality with the volume based code is larger than the number of instances solved to optimality by the simplex based code. There is only one problem (e12) the simplex based code solves but the volume based one does not. However when we have examined the log files, we found that when the simplex based run was faster the primal heuristics in the volume based run found the optimal solution (or in the case of e12, did not happen to find it at all) very late. If the optimal solution value were given to both runs then the volume based runs would always have finished

faster. Also notice, that for those cases not solved to optimality, the volume based code always produced a better lower bound.

Figure 1 gives an even more striking picture of the superiority of the volume based code. Here we chose several instances from Steinlib and plotted how the lower bound progressed in the root node (plots for other instances would exhibit the same structure). On these charts the results based on the VA are drawn with bold lines, those based on dual simplex are drawn with thin lines.

The charts on the left depict the value of the lower bound with respect to the elapsed time. It is clear that at any particular time the volume based lower bound is significantly better.

In charts on the right we have plotted the lower bounds against the iteration count. Notice, that the simplex based runs almost always do fewer iterations showing that the volume based runs take less time per iteration, *i.e.*, the VA solves these LP relaxations faster. Also, we can see that the lower bounds are better with VA at any given iteration as well (not only at any given time).

The fact that the volume based lower bounds are superior indicates that the cuts generated from the primal solution provided by the VA are stronger. Notice that the primal vector used for producing cutting planes is the convex combination showed in (5). This is a point near the center of the optimal face. We believe that this explains the superiority of the cuts produced by the volume based code. A similar observation has been made in [33] when using a cutting plane method based on an interior point algorithm.

Finally, although we do not have exact figures, we have monitored the memory usage of the running programs and the simplex based runs used 2–4 times the memory than the corresponding volume based runs (before they switched over to the simplex based bounding).

6. The max-cut problem

Given a graph $G = (V, E)$ with edge weights, the max-cut problem is defined as partitioning the set of nodes into two sets that maximize the total weight of the edges between the two sets. B&C algorithms for complete or highly dense graphs have been presented in [9, 16]. Max-cut in complete graphs has also been treated with Semidefinite Programming in [26]. In this paper we treat sparse graphs. The linear programming relaxation of the Max-Cut problem for sparse graphs is

$$\begin{aligned}
&\max c^T x \\
&\text{subject to} \\
&\sum_{e\in F} x_e - \sum_{e\in C\setminus F} x_e \le |F| - 1, \text{where } C \text{ is a cycle, and } F \subseteq C \text{ with } |F| = 2k+1 \qquad (12)\\
&0 \le x \le 1.
\end{aligned}$$

Table 1: Computational Results on the Steiner tree problem.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Volume					Simplex				
				Time	LB	UB	Tree	Root	Time	LB	UB	Tree	Root
alue3146	2951	5088	64	1616	2240.00•	2240	3	76/75	3071	2240.00•	2240	3	84
alue5067	2766	4693	68	843	2586.00•	2586	1	41/40	1963	2586.00•	2586	1	53
alue5345	4168	7060	68	11886	3501.68	3537	5	244/239	11027	3298.95	3590	1	79
alue5623	3499	5869	68	10839	3406.33	3470	1	272/218	10900	3297.46	3474	1	98
alue5901	9650	16415	68	10805	3666.62	4075	1	66/66	10959	3327.83	4039	1	50
alue6179	2630	4400	66	846	2447.00•	2447	3	45/44	1537	2447.00•	2447	3	56
alue6457	3167	5307	68	3728	3057.00•	3057	1	130/129	10870	3043.69	3104	1	135
alue6735	3422	5910	68	1888	2696.00•	2696	1	70/69	3302	2696.00•	2696	1	65
alue6951	2188	3723	67	598	2386.00•	2386	1	48/47	1014	2386.00•	2386	1	58
alue7066	5444	9400	14	11051	1846.49	2278	1	136/131	11021	1747.61	2278	1	103
alut1181	2894	5486	64	5283	2353.00•	2353	1	227/226	10877	2262.52	2420	1	97
alut2010	5710	10535	68	11546	3307.00•	3307	1	174/173	11075	3072.52	3391	1	71
alut2288	8758	16227	68	10809	3548.58	3955	1	76/76	11159	3095.18	3973	1	49
alut2566	4697	8656	67	10802	3055.27	3108	1	228/228	10935	2852.31	3126	1	82
diw0234	5258	9968	25	10901	1974.59	2010	1	129/94	10914	1933.71	2017	1	105
diw0445	1709	3186	33	3008	1363.00•	1363	3	209/146	3285	1363.00•	1363	3	128
diw0459	3491	6595	25	3546	1362.00•	1362	1	234/113	6270	1362.00•	1362	1	171
diw0473	2097	3986	25	1188	1098.00•	1098	1	95/61	1466	1098.00•	1098	1	108
diw0487	2250	4168	25	1684	1419.00•	1419	1	151/147	2306	1419.00•	1419	1	148
diw0559	3597	6837	18	11170	1453.84	1577	11	279/252	10858	1440.44	1583	1	168
diw0778	7134	13612	24	10833	2018.96	2200	1	130/130	10926	1794.12	2190	1	93
diw0779	11680	22346	50	11012	3232.49	4686	1	48/48	10866	2682.56	4710	1	43
diw0795	3076	5752	10	10960	1435.63	1593	1	215/195	10834	1399.16	1598	1	139
diw0801	2848	5350	10	10849	1475.26	1594	1	334/212	10868	1442.15	1594	1	147
diw0819	10427	19914	32	10908	2752.93	3429	1	65/65	10917	2278.28	3457	1	52
diw0820	11604	22202	37	11025	2890.46	4310	1	48/48	11073	2036.99	4332	1	50
dmxa0368	1926	3532	18	3173	1017.00•	1017	1	319/110	3369	1017.00•	1017	1	300
dmxa0454	1707	3101	16	710	914.00•	914	1	109/109	1012	914.00•	914	1	96
dmxa1010	3680	6733	23	4709	1488.00•	1488	1	335/194	8800	1488.00•	1488	3	225
dmxa1801	2087	3840	17	2449	1365.00•	1365	1	201/201	9940	1365.00•	1365	1	227

Table 1: Continued.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Volume					Simplex				
				Time	LB	UB	Tree	Root	Time	LB	UB	Tree	Root
e12	2487	11362	10	10800	66.99	68	1	49/49	1000	67.00•	67	1	77
e13	1884	4300	395	1335	1190.00•	1190	1	26/24	2262	1190.00•	1190	1	51
e14	1514	3058	484	1031	1333.00•	1333	1	27/21	1686	1333.00•	1333	3	33
e16	2500	25184	5	573	15.00•	15	1	11/11	4810	15.00•	15	5	11
e17	2500	21508	10	581	25.00•	25	1	18/18	11186	24.00	25	21	13
e18	2382	7368	408	10989	552.47	554	25	38/35	10814	552.25	554	15	89
e19	2127	5352	574	4209	697.00•	697	7	42/35	11583	693.60	709	17	34
e20	904	1618	534	20	589.00•	589	1	5/5	1728	589.00•	589	7	20
es30c	649	1246	30	947	42016079.00•	42016079	1	270/159	1074	42016079.00•	42016079	1	244
es30f	589	1126	30	553	36989394.00•	36989394	1	211/189	1076	36989394.00•	36989394	1	285
es40a	1169	2268	39	1436	43793623.00•	43793623	1	217/215	2609	43793623.00•	43793623	1	204
es40b	1123	2173	39	1689	44560187.00•	44560187	1	221/184	3186	44560187.00•	44560187	1	235
es40c	1147	2225	40	7292	49185709.00•	49185709	3	203/141	5250	49185709.00•	49185709	1	310
es40d	1120	2166	40	1574	43587584.00•	43587584	1	219/203	2734	43587584.00•	43587584	1	208
es40e	1280	2488	40	11195	50631467.99	51486717	11	265/227	10812	50620796.96	51886525	7	203
es40f	1098	2129	40	6037	49328246.00•	49328246	5	277/179	5100	49328246.00•	49328246	1	474
es40g	1162	2251	39	1152	44546137.00•	44546137	1	159/138	1867	44546137.00•	44546137	1	144
es40h	1248	2424	40	1777	46505796.00•	46505796	1	212/201	4493	46505796.00•	46505796	1	255
es40i	1220	2369	40	10806	50159681.46	50546305	3	255/239	10802	50521142.57	50548442	1	410
es40j	1246	2419	40	4021	56583112.00•	56583112	1	297/222	8116	56583112.00•	56583112	1	275
es40k	1184	2293	40	2679	46282378.00•	46282378	1	314/286	5737	46282378.00•	46282378	1	310
es40l	1252	2430	40	1521	41423579.00•	41423579	1	164/133	2542	41423579.00•	41423579	1	141
es40m	1373	2671	40	3545	51316954.00•	51316954	1	277/194	5986	51316954.00•	51316954	1	281
es40n	1300	2528	40	2228	48340363.00•	48340363	1	220/183	4429	48340363.00•	48340363	1	191
es40o	1303	2530	40	2573	50023470.00•	50023470	3	197/178	10800	49992197.90	50023470	13	220
gap3128	9612	17171	104	10893	4127.10	4377	1	73/73	11110	3689.85	4403	1	56

Table 1: Continued.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Volume					Simplex			
				Time	LB	UB	Tree	Root	Time	LB	UB	Tree	Root
msm2152	1938	3453	37	726	1590.00●	1590	1	70/70	2454	1590.00●	1590	1	93
msm2492	3761	6733	12	4446	1459.00●	1459	3	270/191	9710	1459.00●	1459	15	141
msm2525	2726	4868	12	4038	1290.00●	1290	1	281/83	4353	1290.00●	1290	1	287
msm2601	2668	4759	16	10810	1418.32	1469	3	348/261	10825	1395.16	1469	1	222
msm2846	3091	5564	89	7027	3135.00●	3135	5	213/189	10861	3127.24	3147	1	124
msm3727	4354	7942	21	3922	1376.00●	1376	1	131/121	7296	1376.00●	1376	1	167
msm3829	3849	6811	12	10858	1473.90	1624	7	158/119	10802	1468.68	1609	11	125
msm4312	4733	8374	10	11572	1789.71	2085	1	146/144	10833	1668.89	2073	1	102
taq0014	5886	10360	128	10840	5167.19	5420	1	109/109	10823	4533.48	5510	1	58
taq0365	3751	6569	22	10888	1896.42	1931	1	472/460	10800	1769.99	1966	1	97
taq0377	6316	11135	136	10847	6157.72	6579	1	83/83	10801	5458.20	6630	1	52
taq0903	5503	9690	130	10840	4888.24	5181	1	112/112	10859	4391.66	5227	1	53

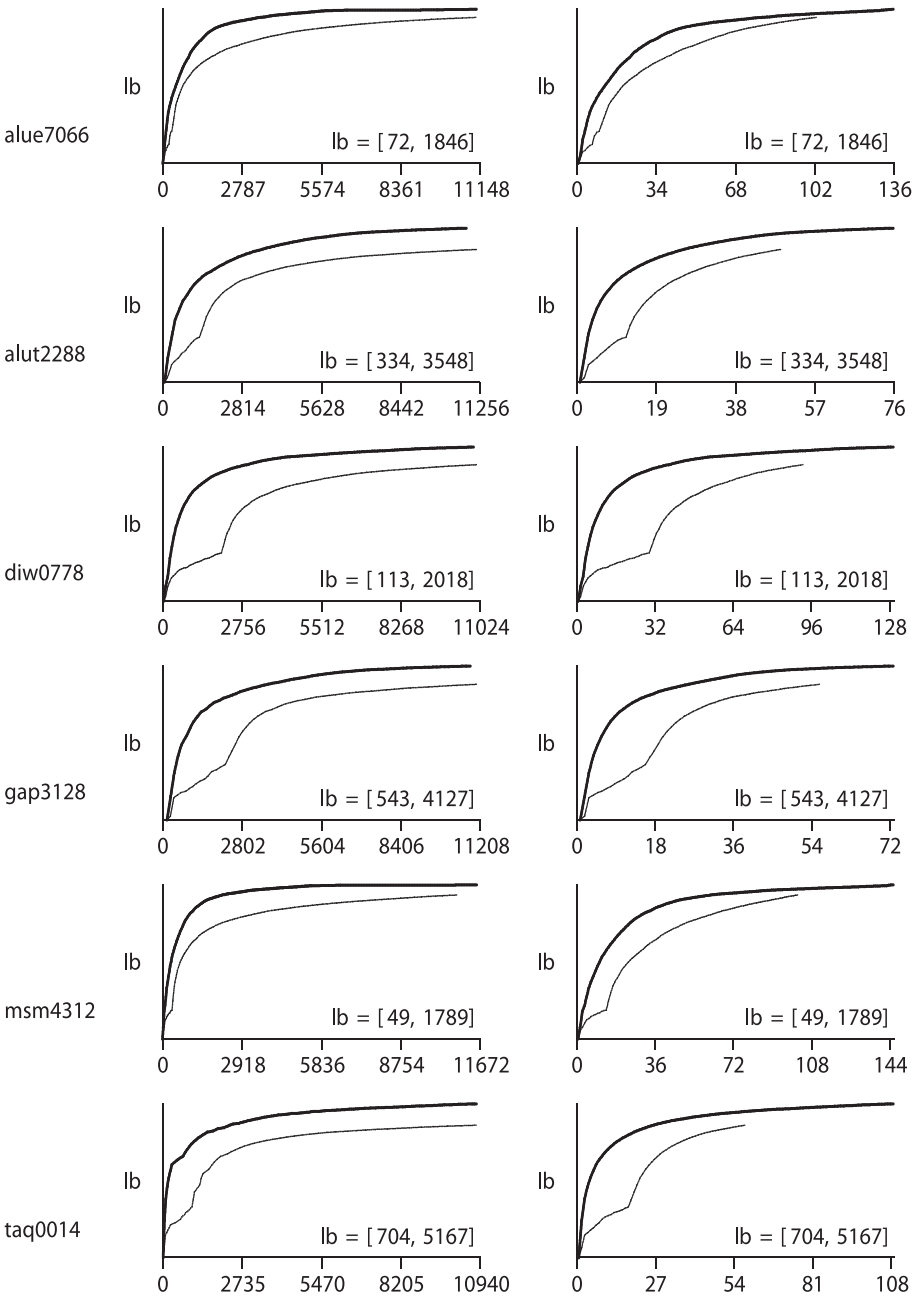

FIGURE 1. Lower bound *vs.* seconds elapsed (left) in the root and Lower bound vs. iteration count (right) in the root. Bold lines: Volume Algorithm. Thin lines: simplex.

Here x_e should take the value 1 if the edge e appears in the cut, and 0 otherwise. Constraints (12) are called *cycle inequalities* and they ensure that the intersection of a cycle and a cut has even cardinality. A polynomial time separation algorithm for this class of inequalities has been given in [10], however we use a faster heuristic from [8]. Let $\bar{x}$ be the vector to separate, define weights

$$w_e = c_e \cdot \max(\bar{x}_e, 1 - \bar{x}_e).$$

Then find a maximum weighted spanning tree T with weights w. For an edge $e \in T$, if $\bar{x}_e \geq 1 - \bar{x}_e$ then the end-nodes of e should be on opposite sides of the cut. We give the label "A" to this edge. Otherwise, if $\bar{x}_e < 1 - \bar{x}_e$ then the end-nodes should be on the same side of the cut and we give the label "B" to the edge. Once every edge in T has been labeled the edges with label "A" define a heuristic cut K. Then for an edge $e \notin T$ we add it to T and look at the created cycle C. If $e \in K$ then we test the violation of the inequality (12), where the set F is given by the A-edges. If $e \notin K$ the set F is given by the A-edges and the edge e.

7. Computational results on the max-cut problem

We have run our experiments on an IBM RS/6000 44P Model 270 workstation with a 375 MHz processor and 1GB of core memory and 1 hour time limit. We have considered six different classes of sparse graphs.

The first four classes are max-cut problems arising from variations of the so called *Ising spin glass model*, a model in statistical physics, that has been studied in [5, 8, 14, 15, 33]. In one case the corresponding problem is to find the maximum cut in a toroidal grid where half of the edge weights in the grid are 1, the other half are –1. This problem class has been identified as a challenging case for simplex based cutting plane algorithms. In [15] the authors report that a 70×70 grid takes on the order of 16 h on a Sun SPARC 10 machine. (Our running times are on the order of a minute on a roughly 30 times faster machine.) Toroidal grids have also been treated with a cutting plane algorithm based on an interior point algorithm in [33]. In that paper times of the order of 2200 s and 12 000 s are reported for 70×70 and 100×100 grids on a Sun SPARC 20/71 (which corresponds to roughly 110 and 600 s on our machine).

We have studied four variations of this problem. First the edge weights are either randomly generated from $\{1, -1\}$ or generated with a gaussian distribution (keeping 6 significant digits). Also, an external magnetic field can be present or not, this is modelled by an extra node connected to every node in the grid (all the new edges have the same weight, which represents the strength of the field). All these models have significance in physics.

Tables 2–5 show the results of the Ising experiments. To somewhat smooth out the randomness in the experiments we have generated 8 instances for each entry

TABLE 2. Ising spinglass problems: ±1 weights, no external field (8 instances/type).

	solved by both			solved by vol		solved by sim	
size	#	vol time	sim time	#	time	#	time
70	0	–	–	8	90.72	0	–
80	0	–	–	8	238.35	0	–
90	0	–	–	8	313.13	0	–
100	0	–	–	8	491.14	0	–
110	0	–	–	8	785.48	0	–
120	0	–	–	6	1701.34	0	–
130	0	–	–	5	2295.42	0	–

TABLE 3. Ising spinglass problems: gaussian weights, no external field (8 instances/type).

	solved by both			solved by vol		solved by sim	
size	#	vol time	sim time	#	time	#	time
40	8	34.50	32.39	8	34.50	8	32.39
50	8	62.47	79.28	8	62.47	8	79.28
60	8	129.55	186.40	8	129.55	8	186.40
70	8	171.88	448.72	8	171.88	8	448.72
80	8	282.21	1267.05	8	282.21	8	1267.05
90	6	408.68	2671.25	8	427.68	6	2903.43
100	0	–	–	8	868.23	0	–
110	0	–	–	8	1254.86	0	–

where there is no external field and (due to the large number of distinct problem characteristics) 4 instances when there was an external field. First the size of the toroidal grid is given (possibly followed by the field strength). Then we present the instances that could be solved to optimality by both algorithms. For them we present the the number of instances solved and for both codes the average running time of the solved instances. Then for the volume based algorithm we present the number of instances solved to optimality, then we present the average running time among all instances regardless of whether they were solved or not (for those not solved we use 3600 seconds as their running time). Finally, the same information for the simplex based code is presented.

The fifth set of instances are quadratic 0–1 programming problems that have been transformed into max-cut problems (for details see [9]). The results are displayed in Table 6 which has the same layout as the previous tables.

Finally, we have received a set of eight spin glasses problems that were reported to be very difficult in [15]. Table 7 gives our running times for these problems.

TABLE 4. Ising spinglass problems: ±1 weights, external field present (4 instances/type).

		solved by both			solved by vol		solved by sim	
size	field	#	vol time	sim time	#	time	#	time
70	0.3981	4	264.62	572.78	4	264.62	4	572.78
	0.5012	4	155.10	194.61	4	155.10	4	194.61
	0.6310	4	125.63	134.67	4	125.63	4	134.67
	0.7943	4	128.89	79.59	4	128.89	4	79.59
	1.0000	4	108.77	61.62	4	108.77	4	61.62
	1.2589	4	77.39	36.42	4	77.39	4	36.42
	1.5849	4	46.13	28.44	4	46.13	4	28.44
	1.9953	4	75.17	28.78	4	75.17	4	28.78
	2.5119	4	17.99	8.69	4	17.99	4	8.69
80	0.3981	4	318.55	1171.19	4	318.55	4	1171.19
	0.5012	4	260.86	517.97	4	260.86	4	517.97
	0.6310	4	221.11	236.93	4	221.11	4	236.93
	0.7943	4	150.10	116.44	4	150.10	4	116.44
	1.0000	4	178.69	110.98	4	178.69	4	110.98
	1.2589	4	120.92	61.18	4	120.92	4	61.18
	1.5849	4	75.42	43.63	4	75.42	4	43.63
	1.9953	4	106.33	42.98	4	106.33	4	42.98
	2.5119	4	30.92	12.39	4	30.92	4	12.39
90	0.3981	3	583.74	2361.65	4	609.09	3	2671.24
	0.5012	4	457.04	896.88	4	457.04	4	896.88
	0.6310	4	339.04	446.67	4	339.04	4	446.67
	0.7943	4	240.82	186.76	4	240.82	4	186.76
	1.0000	4	360.57	185.18	4	360.57	4	185.18
	1.2589	4	142.91	87.89	4	142.91	4	87.89
	1.5849	4	124.27	68.16	4	124.27	4	68.16
	1.9953	4	160.64	66.48	4	160.64	4	66.48
	2.5119	4	46.98	18.45	4	46.98	4	18.45
100	0.3981	2	725.50	3249.31	4	827.65	2	3424.65
	0.5012	4	553.16	1755.30	4	553.16	4	1755.30
	0.6310	4	428.70	848.11	4	428.70	4	848.11
	0.7943	4	342.21	331.11	4	342.21	4	331.11
	1.0000	4	318.21	250.64	4	318.21	4	250.64
	1.2589	4	252.32	130.23	4	252.32	4	130.23
	1.5849	4	185.65	101.45	4	185.65	4	101.45
	1.9953	4	219.37	107.28	4	219.37	4	107.28
	2.5119	4	52.33	25.86	4	52.33	4	25.86
110	0.3981	0	–	–	4	1385.94	0	–
	0.5012	3	794.66	2486.20	4	889.40	3	2764.65
	0.6310	4	548.28	1020.89	4	548.28	4	1020.89
	0.7943	4	554.24	502.86	4	554.24	4	502.86
	1.0000	4	579.62	368.93	4	579.62	4	368.93
	1.2589	4	309.13	195.74	4	309.13	4	195.74
	1.5849	4	203.00	137.80	4	203.00	4	137.80
	1.9953	4	330.64	145.39	4	330.64	4	145.39
	2.5119	4	78.09	37.40	4	78.09	4	37.40

TABLE 5. Ising spinglass problems: gaussian weights, external field present (4 instances/type).

		solved by both			solved by vol		solved by sim	
size	field	#	vol time	sim time	#	time	#	time
40	0.01778	4	253.73	1160.88	4	253.73	4	1160.88
	0.03162	4	256.99	797.88	4	256.99	4	797.88
	0.05623	4	210.22	319.88	4	210.22	4	319.88
	0.10000	4	122.48	131.65	4	122.48	4	131.65
	0.17783	4	69.96	77.14	4	69.96	4	77.14
	0.31623	4	47.52	40.31	4	47.52	4	40.31
	0.56234	4	23.09	14.16	4	23.09	4	14.16
	1.00000	4	11.14	7.15	4	11.14	4	7.15
	1.77828	4	7.00	3.78	4	7.00	4	3.78
50	0.01778	0	–	–	4	506.07	0	–
	0.03162	2	442.59	1922.75	4	536.82	2	2761.38
	0.05623	2	212.81	1741.23	4	441.87	2	2670.61
	0.10000	4	291.66	617.21	4	291.66	4	617.21
	0.17783	4	163.26	283.26	4	163.26	4	283.26
	0.31623	4	83.02	77.17	4	83.02	4	77.17
	0.56234	4	45.67	28.00	4	45.67	4	28.00
	1.00000	4	25.52	15.89	4	25.52	4	15.89
	1.77828	4	12.08	6.65	4	12.08	4	6.65
60	0.01778	0	–	–	4	887.02	0	–
	0.03162	0	–	–	4	1039.82	0	–
	0.05623	1	348.39	2689.18	4	694.35	1	3372.30
	0.10000	3	463.64	1239.43	4	602.55	3	1829.58
	0.17783	4	270.89	549.34	4	270.89	4	549.34
	0.31623	4	202.14	328.31	4	202.14	4	328.31
	0.56234	4	80.47	58.01	4	80.47	4	58.01
	1.00000	4	44.69	30.05	4	44.69	4	30.05
	1.77828	4	19.64	11.87	4	19.64	4	11.87
70	0.01778	0	–	–	4	1923.28	0	–
	0.03162	0	–	–	3	2040.43	0	–
	0.05623	0	–	–	4	1671.77	0	–
	0.10000	0	–	–	4	935.32	0	–
	0.17783	3	653.49	2367.66	4	634.84	3	2675.75
	0.31623	4	260.91	499.81	4	260.91	4	499.81
	0.56234	4	137.87	104.60	4	137.87	4	104.60
	1.00000	4	60.71	50.23	4	60.71	4	50.23
	1.77828	4	38.26	21.87	4	38.26	4	21.87
80	0.01778	0	–	–	3	2996.72	0	–
	0.03162	0	–	–	3	2240.55	0	–
	0.05623	0	–	–	3	2592.82	0	–
	0.10000	0	–	–	4	1551.79	0	–
	0.17783	1	705.02	2718.90	4	829.74	1	3379.72
	0.31623	4	392.07	746.16	4	392.07	4	746.16
	0.56234	4	242.04	187.32	4	242.04	4	187.32
	1.00000	4	107.81	78.35	4	107.81	4	78.35
	1.77828	4	52.70	33.73	4	52.70	4	33.73

TABLE 6. Quadratic 0-1 programming based problems (4 instances/type).

		solved by both			solved by vol		solved by sim	
size	% of 1's	#	vol time	sim time	#	time	#	time
60	20	4	9.18	12.34	4	9.18	4	12.34
	30	4	137.31	123.89	4	137.31	4	123.89
	40	4	109.12	130.38	4	109.12	4	130.38
	50	4	349.85	417.58	4	349.85	4	417.58
	60	4	521.01	752.20	4	521.01	4	752.20
	70	3	491.15	650.46	3	1268.37	3	1387.85
80	20	4	231.84	272.00	4	231.84	4	272.00
	30	4	236.16	412.35	4	236.16	4	412.35
	40	4	2083.01	2442.76	4	2083.01	4	2442.76
	50	4	828.84	1445.77	4	828.84	4	1445.77
100	20	4	891.66	1118.72	4	891.66	4	1118.72
	30	2	1631.98	2361.07	3	2471.69	2	2980.53
	40	2	511.06	1879.82	2	2055.53	2	2739.91

TABLE 7. Instances from [15].

instance	simplex	volume
L_70_1	–	139.72
L_70_2	–	66.48
L_70_3	–	141.35
L_70_4	–	332.10
L_70_5	–	238.52
L_70_6	–	158.92
L_70_7	–	216.79
L_70_8	–	205.18

These tables also show that it is advantageous to use the volume algorithm until it starts to tail off, but the superiority is not as clear-cut as it was for the Steiner tree problem. This is especially apparent for smaller sized Ising spinglass problems when stronger external field is present. For those problems the purely simplex based code consistently outperforms the other one. Upon examining the log files we have found the reason for this behavior, a reason which is very natural in hindsight.

If a problem can be solved in the root node quickly (in about 10 iterations) then the volume based code is at a disadvantage since first the volume algorithm has to tail off, then we switch over to the simplex method, and although that nearly always proves optimality, the extra 3–5 LPs solved with the volume algorithm

slows down the code. On the other hand, if switching over to the simplex method does not prove optimality and we have to explore a reasonably large search tree then it is more or less random which code is faster (it depends on which one is luckier to find an optimal solution sooner).

8. Conclusions

We have presented B&C algorithms where the LP relaxations can be treated either with the VA or with the simplex method. We first considered the Steiner tree problem in graphs. The volume based approach was able to solve to optimality substantially more instances. For the instances not solved to optimality, the lower bounds produced by the volume based code were always better than the lower bounds produced by the simplex based approach.

Then we presented similar experiments with the max-cut problem. Again the superiority of the volume based approach is clear as we could handle sizes that had not been solved before. However this problem class presented a better insight to when it is advantageous to jump-start the B&C algorithm by solving the LP relaxations with the Volume Algorithm: whenever the bulk of the work is done in the root node (*i.e.*, the search tree is reasonably small), but the problem cannot be solved in just a few cutting plane iterations.

Most B&C methods are being implemented based on the simplex method. For many combinatorial problems the resulting linear programs are extremely difficult to solve due to degeneracy and numerical difficulties. We expect that in these cases a volume based approach will be more advantageous.

Acknowledgements. We are grateful to Dr. T. Koch for providing us with his preprocessing code for Steiner tree problems and to Dr. Michael Jünger for sending us some max-cut instances from [15]. We are also grateful to Dr. Márta Eső for providing helpful comments when writing this paper.

References

[1] Y.P. Aneja, An integer linear programming approach to the Steiner problem in graphs. *Networks* **20** (1980) 167–178.

[2] D. Applegate, R. Bixby, V. Chvátal and W. Cook, On the solution of traveling salesman problems, in *Proc. of the International Congress of Mathematicians* **III** (1998) 645–656.

[3] L. Bahiense, F. Barahona and O. Porto, Solving Steiner tree problems in graphs with Lagrangian relaxation. *J. Comb. Optim.* **7** (2003) 259–282.

[4] L. Bahiense, N. Maculan and C. Sagastizábal, The volume algorithm revisited: relation with bundle methods. *Math. Program.* **94** (2002) 41–69.

[5] F. Barahona, Ground state magnetization of Ising spin glasses. *Phys. Rev. B* **49** (1994) 2864–2867.

[6] F. Barahona and R. Anbil, The volume algorithm: producing primal solutions with a subgradient method. *Math. Program.* **87** (2000) 385–399.

[7] F. Barahona and R. Anbil, On some difficult linear programs coming from set partitioning. *Discrete Appl. Math.* **118** (2002) 3–11. Third ALIO-EURO Meeting on Applied Combinatorial Optimization (Erice, 1999).
[8] F. Barahona, M. Grötschel, M. Jünger and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design, *Oper. Res.* **36** (1988) 493–513.
[9] F. Barahona, M. Jünger and G. Reinelt, Experiments in quadratic 0-1 programming. *Math. Program.* **44** (1989) 127–137.
[10] F. Barahona and A. Mahjoub, On the cut polytope. *Math. Program.* **36** (1986) 157–173.
[11] A. Caprara and M. Fischetti, *Branch-and-cut algorithms*, in Annotated Bibliographies in Combinatorial Optimization, edited by M.D. Amico, F. Maffioli and S. Martello, Wiley (1997) 45–63.
[12] S. Chopra, E.R. Gorres and M.R. Rao, Solving the Steiner tree problem in graphs using branch-and-cut. *ORSA J. Comput.* **4** (1992) 320–335.
[13] COIN-OR, `http:www.coin-or.org`.
[14] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt and G. Rinaldi, Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. *J. Stat. Phys.* **80** (1995) 487–496.
[15] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt and G. Rinaldi, Exact ground states of two-dimensional +−J Ising spin glasses. *J. Stat. Phys.* (1996).
[16] C. De Simone and G. Rinaldi, A cutting plane algorithm for the max-cut problem. *Optim. Method. Softw.* **3** (1994) 195–214.
[17] J. Edmonds, Optimum branchings. *J. Res. Natl. Bur. Stand.* **71B** (1967) 233–240.
[18] L.F. Escudero, M. Guignard and K. Malik, A Lagrangean relax-and-cut approach for the sequential ordering problem with precedence relations. *Ann. Oper. Res.* **50** (1994) 219–237.
[19] J.J. Forrest, The COIN-OR Linear Program Solver (CLP), INFORMS Atlanta, (2003).
[20] A. Frangioni, A. Lodi and G. Rinaldi, Optimizing over semimetric polytopes, in Integer programming and combinatorial optimization, Springer, Berlin *Lect. Notes Comput. Sci.* **3064** (2004) 431–443.
[21] M.X. Goemans and Y.S. Myung, A catalog of Steiner tree formulations, *Networks* **23** (1993), pp. 19–28.
[22] M. Guignard, *Efficient cuts in lagrangean* "Relax-and-Cut" schemes. *Eur. J. Oper. Res.* **105** (1998) 216–223.
[23] M. Held and R.M. Karp, The travelling salesman problem and minimum spanning trees. *Oper. Res.* **18** (1970) 1138–1162.
[24] M. Held and R.M. Karp, The travelling salesman problem and minimum spanning trees: Part II. *Math. Program.* **1** (1971) 6–25.
[25] M. Held, P. Wolfe and H.P. Crowder, Validation of subgradient optimization. *Math. Program.* **6** (1974) 62–88.
[26] C. Helmberg and F. Rendl, Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Math. Program.* **82** (1998) 291–315.
[27] T. Koch and A. Martin, *Steinlib*, `http://elib.zib.de/steinlib/steinlib.php`.
[28] T. Koch and A. Martin, Solving Steiner tree problems in graphs to optimality, Networks **32** (1998) 207–232.
[29] C. Lemaréchal, Nondifferentiable optimization, in *Optimization, Hanbooks Oper. Res.*, edited by G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd, North Holland, 1989, pp. 529–572.
[30] A. Lucena, *Steiner problem in graphs: Lagrangian relaxation and cutting-planes*, *COAL Bull.* **21** (1992) 2–7.
[31] A. Lucena, Tight bounds for the Steiner problem in graphs, in *Proc. Netflow* **93** (1993) 147–154.
[32] A. Lucena. and J. Beasley, A branch-and-cut algorithm for the Steiner problem in graphs. *Networks* **31** (1998), pp. 39–59.
[33] J.E. Mitchell, Computational experience with an interior point cutting plane algorithm. *SIAM J. Optim.* **10** (2000) 1212–1227.

[34] M. Padberg and G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.* **33** (1991) 60–100.
[35] T. Polzin and S.V. Daneshmand, Improved algorithms for the Steiner problem in networks. *Discrete Appl. Math.* **112** (2001) 263–300. Combinatorial Optimization Symposium (Brussels, 1998).
[36] H. Takahashi and A. Matsuyama, An approximated solution for the Steiner tree problem in graphs, *Math. Japonica* **254** (1980) 573–577.
[37] E. Uchoa, M. Poggi de Aragão and C.C. Ribeiro, Preprocessing Steiner problems from VLSI layout, *Networks* **40** (2002) 38–50.
[38] P. Wolfe, A method of conjugate subgradients for minimizing nondifferentiable functions. *Math. Program. Study* **3** (1975) 145–173.