RC22221 (W0110-084) October 31, 2001
Mathematics

IBM Research Report

Branch and cut based on the volume algorithm: Steiner trees in graphs and max-cut

Francisco Barahona, Laszlo Ladanyi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

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

FRANCISCO BARAHONA AND LÁSZLÓ LADÁNYI

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 arising at each node of the search tree. This means we use fast approximate solutions to these linear programs instead of exact but slower solutions. Our claim is that such a Branch-and-Cut code should work better than a simplex based code (everything else being equal) for problems whose linear programming relaxation is well suited to be solved with the volume algorithm. 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.

1. INTRODUCTION

Since the early eighties, the Branch-and-Cut (B&C) algorithm has been used for a variety of combinatorial optimization problems, see [10] 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 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 it can be advantageous to replace the dual simplex method with a fast approximate method, the volume algorithm [6].

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. The traditional subgradient method produces only a dual solution and a lower bound. 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 [27, 28, 17, 19]. In relax-and-cut the cutting planes are produced based on a solution to the Lagrangean problem (see (4) in Section 2). This is a point 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.

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 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 IP polytope.

Date: December 14, 2001.

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 solving. A number of combinatorial optimization problems fall into this category.

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 by a wide margin for the problem classes studied here.

This paper is organized as follows. In Section 2 we describe the volume algorithm. In Section 3 we describe the BCP framework. Section 4 is devoted to the Steiner tree problem in graphs. Section 5 presents computational results for the Steiner tree problem. Section 6 deals with the max-cut problem. Some conclusions are given in Section 7.

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 [20, 21] and Held, Wolfe and Crowder [22]. 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.

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). For a vector of dual multipliers $\pi \geq 0$, a lower bound on the minimum of (1) is given by

$$L(\pi) = \min(c - \pi A)\, x + \pi b \qquad 0 \leq x \leq 1 \tag{4}$$

In order to maximize the function $L(\cdot)$ we apply the volume algorithm described below, see also [6].

Volume algorithm

Step 0: Starting with a vector $\bar{\pi} \geq 0$, solve (4) with $\pi = \bar{\pi}$ to obtain its solution $\bar{x} = x^0$ and $\bar{z} = L(\bar{\pi})$. Set $t = 1$.

Step 1: Compute $v^t = b - A\bar{x}$ and $\pi' = \bar{\pi} + sv^t$ for a step size s given by (7). Set $\pi_i^t = \max(0, \pi_i')$.

Solve (4) with $\pi = \pi^t$ to get its solution x^t and $z^t = L(\pi^t)$. Update $\bar{x}$ as

$$\bar{x} \leftarrow \alpha x^t + (1-\alpha)\bar{x}, \tag{5}$$

where α is a number between 0 and 1.

Step 2: If $z^t > \bar{z}$ update $\bar{\pi}$ and $\bar{z}$ as

$$\bar{\pi} \leftarrow \pi^t, \quad \bar{z} \leftarrow z^t.$$

Let $t \leftarrow t + 1$ and go to Step 1.

Notice that in Step 2 we update $\bar{\pi}$ only if $z^t > \bar{z}$, so this is an ascent method. We are trying to mimic the bundle method [26], 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 formula (5). If $x^0, \ldots, x^t$ is the sequence of vectors produced by problem (4), then VA yields

$$\bar{x} = \alpha x^t + (1-\alpha)\alpha x^{t-1} + \cdots + (1-\alpha)^t x^0. \tag{6}$$

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 formula (5) 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

$$s = \lambda \frac{T - \bar{z}}{\|v^t\|^2}, \tag{7}$$

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 $\bar{z} \geq 0.95T$, 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

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

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 (5) is chosen as the solution of the following 1-dimensional problem:

$$\min \|b - A(\alpha x^t + (1-\alpha)\bar{x})\| \tag{8}$$

subject to

$$u/10 \leq \alpha \leq u. \tag{9}$$

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 [33]; the difference is in the bounds $u/10$ and u.

This implementation of the VA is available as an open source program under the auspices of the COIN-OR project [12] and has been also used in [5] for solving set partitioning problems.

The convergence properties of the VA have been studied in [3].

3. The Branch-and-Cut framework

We used BCP, a state of the art 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. BCP is also available from the COIN-OR project [12].

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, two nodes were used as branching candidates.

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

We also used *reduced costs fixing*, see [31]. 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 can be fixed to 0 and, respectively, if $L(\bar{\pi}) - \bar{c}_j > UB$ then x_j can be fixed to 1.

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 3 iterations.

In the volume based code we switched to a *hybrid method* if the gap had not decreased by at least 0.1 in 2 successive cutting plane rounds. This hybrid method consisted of running the VA that produced a dual vector $\bar{\pi}$ and then solving the LP below with the dual simplex algorithm.

$$\begin{aligned} &\min(c - \bar{\pi}A)x + \bar{\pi}s \\ &Ax - Is = b \\ &0 \leq x \leq 1 \\ &0 \leq s. \end{aligned}$$

Usually this improved the lower bound by a small amount that was sufficient to fathom a node. Our experience is that solving this modified LP is much faster than solving the original LP from scratch, see also [5]. In a typical case processing the root node took 70 cutting plane iterations, and the hybrid method was used in the last 4 or 5 before resorting to branching.

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 [18] 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

$$\min cx \tag{10}$$

$$\sum_{i \notin S,\ j \in S} x_{ij} \geq 1, \text{ for all } S \subset V,\ r \notin S,\ S \cap T \neq \emptyset \tag{11}$$

$$0 \leq x \leq 1. \tag{12}$$

We deal with this relaxation in a cutting plane fashion. This formulation was proposed in [1] and has been used in [11, 25]. An equivalent formulation has been used in [29]. Although the separation problem for inequalities (11) 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' [16] 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 [32] and is described in Algorithm 1.

Algorithm 1 Takahashi-Matsuyama Steiner tree heuristics

```
Chose an initial terminal vertex v.
Let T be a tree containing only v.
while there exists a terminal node t ∉ T do
  for all terminal node t ∉ T do
    Compute the shortest path from t to T.
  end for
  Add the terminal node is closest to T and the corresponding path to T.
end while
```

We have extended this algorithm 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.

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

Our implementation is very similar to the one presented in [25], the main difference is that we use the VA to handle the linear programs. In Tables 1-3 we present a comparison between our simplex based code and our volume based code. We took instances from *Steinlib* [24], a library for Steiner tree problem in graphs. We chose those instances treated in [25] that took more than 1000 seconds to solve. We have used the same pre-processing code as in [25]. In the first column we have the name of the instance, then we present the number of nodes, number of edges, and number of terminals after pre-processing. Then we present the time, the lower bound, and the upper bound for the volume based code. Finally we have the time, the lower bound, and the upper bound for the simplex-based code. We mark with a bullet (•) all cases where we have a proof of optimality. We had set a limit of 10000 seconds. Since code checked the time only after solving each LP, the times reported are sometimes slightly larger than 10000 seconds. Observe that the number of instances solved to optimality with the volume based code is substantially larger than the number of instances solved to optimality by the simplex based code. Also for those cases not solved to optimality, the volume based code always produced a better lower bound. Several instances left unsolved in [25] have been solved here. We ran all experiments on an IBM RS/6000 44P Model 270 workstation with a 375 MHz processor and 2GB of core memory.

Figures 1 and 2 give an even more striking picture of the superiority of the volume based code. Here we chose several instances from Steinlib and run them for at most 1 hour. The case c20 was solved to optimality by the volume based code in 180 seconds., and by the simplex based code in 1200 seconds. 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 in Figure 1 depicts 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 Figure 2 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

Name	$\|V\|$	$\|E\|$	$\|T\|$	Volume			Simplex		
				Time	LB	UB	Time	LB	UB
e08	1344	2623	361	4438	2640.00•	2640	1171	2640.00•	2640
e09	1056	1849	431	3146	3604.00•	3604	426	3604.00•	3604
e13	1884	4300	395	10048	1280.00	1282	4272	1280.00•	1280
e14	1514	3058	484	10151	1732.00	1733	7136	1732.00•	1732
e15	536	831	361	2014	2784.00•	2784	148	2784.00•	2784
e18	2382	7368	408	10028	563.46	573	12399	554.48	627
e19	2127	5352	574	10285	758.00	766	12604	755.78	782
e20	904	1618	534	27	1342.00•	1342	7692	1342.00•	1342
diw0234	5258	9968	25	10278	1992.96	2012	12097	1742.37	2033
diw0445	1709	3186	33	2118	1363.00•	1363	2648	1363.00•	1363
diw0459	3491	6595	25	6493	1362.00•	1362	11122	1296.58	1362
diw0473	2097	3986	25	2470	1098.00•	1098	3629	1098.00•	1098
diw0559	3597	6837	18	10295	1538.37	1577	11455	1315.27	1595
diw0778	7134	13612	24	10045	2086.71	2193	10726	1669.49	2208
diw0779	11680	22346	50	10016	3464.52	4638	12346	2389.35	4686
diw0795	3076	5752	10	10554	1537.65	1568	10128	1268.50	1603
diw0801	2848	5350	10	10065	1561.26	1587	10234	1323.54	1615
diw0819	10427	19914	32	10092	2943.49	3442	11216	2100.60	3478
diw0820	11604	22202	37	10158	3039.42	4294	10434	2054.41	4344
taq0014	5886	10360	128	10222	5229.13	5432	11473	4149.39	5573
taq0365	3751	6569	22	10273	1905.89	1921	10475	1581.99	2002
taq0377	6316	11135	136	10260	6349.11	6571	11799	4879.19	6667
taq0903	5503	9690	130	10004	4954.37	5205	11187	4016.29	5270
dmxa0368	1926	3532	18	3305	1017.00•	1017	4044	1017.00•	1017
dmxa1010	3680	6733	23	5370	1488.00•	1488	10088	1463.95	1488
dmxa1801	2087	3840	17	1913	1365.00•	1365	10240	1339.13	1390

TABLE 1. Steiner tree series E, DIW, TAQ, and DMXA

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 (6). 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 [30] 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.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Volume			Simplex		
				Time	LB	UB	Time	LB	UB
msm2152	1938	3453	37	675	1590.00•	1590	5699	1590.00•	1590
msm2492	3761	6733	12	6536	1459.00•	1459	11451	1393.75	1460
msm2525	2726	4868	12	2168	1290.00•	1290	9706	1290.00•	1290
msm2601	2668	4759	16	5103	1440.00•	1440	10411	1318.17	1488
msm2846	3091	5564	89	10009	3134.00	3136	11663	2996.95	3218
msm3727	4354	7942	21	1308	1376.00•	1376	10219	1341.36	1406
msm3829	3849	6811	12	10034	1528.20	1619	10911	1322.13	1627
msm4312	4733	8374	10	10018	1926.98	2059	11410	1472.13	2114
gap2007	1860	3325	17	847	1104.00•	1104	1882	1104.00•	1104
gap3128	9612	17171	104	10081	4184.87	4378	11487	3462.83	4440
alue3146	2951	5088	64	3157	2240.00•	2240	10239	2234.37	2261
alue5067	2766	4693	68	1360	2586.00•	2586	8128	2586.00•	2586
alue5345	4168	7060	68	10019	3506.00	3507	11880	3104.97	3609
alue5623	3499	5869	68	10011	3410.33	3413	10013	3070.80	3516
alue5901	9650	16415	68	10086	3760.76	4076	10717	3069.98	4081
alue6179	2630	4400	66	2278	2452.00•	2452	4095	2452.00•	2452
alue6457	3167	5307	68	4331	3057.00•	3057	10102	2858.23	3132
alue6735	3422	5910	68	2217	2696.00•	2696	10090	2651.29	2774
alue6951	2188	3723	67	575	2386.00•	2386	3452	2386.00•	2386
alue7065	28711	49226	543	3452	5879.88	46169	3452	5677.27	45914
alue7066	5444	9400	14	10049	2012.32	2288	12372	1534.39	2288
alue7080	29164	49573	2326	12372	18575.34	106041	12372	15800.50	108106
alut1181	2894	5486	64	9404	2353.00•	2353	10830	2119.36	2449
alut2010	5710	10535	68	10033	3307.00	3365	10132	2801.13	3400
alut2288	8758	16227	68	10021	3679.67	3954	10157	2731.91	3973
alut2566	4697	8656	67	10073	3072.50	3111	10931	2597.79	3154

TABLE 2. Steiner tree series MSM, GAP, ALUE and ALUT

Name	$\|V\|$	$\|E\|$	$\|T\|$	Volume			Simplex		
				Time	LB	UB	Time	LB	UB
es30c	649	1246	30	772	43120447.00•	43120447	2043	43120447.00•	43120447
es30d	669	1283	30	1943	42150952.00•	42150952	1610	42150952.00•	42150952
es30e	652	1249	30	756	41739738.00•	41739738	1122	41739738.00•	41739738
es30g	659	1267	29	1161	43761406.00•	43761406	931	43761406.00•	43761406
es30h	577	1101	29	871	41691214.00•	41691214	1160	41691214.00•	41691214
es30k	665	1275	30	2263	41648009.00•	41648009	1732	41648009.00•	41648009
es30o	616	1180	29	790	43035569.00•	43035569	1303	43035569.00•	43035569
es40a	1169	2268	39	1287	44841520.00•	44841520	9230	44841520.00•	44841520
es40b	1123	2173	39	1836	46811310.00•	46811310	10012	46759050.42	47010664
es40c	1147	2225	40	4572	49974160.00•	49974160	10011	49220977.02	50926159
es40d	1120	2166	40	3395	45289858.00•	45289858	9332	45289858.00•	45289858
es40e	1280	2488	40	10012	51811830.00	51963807	10045	50819122.92	52494871
es40f	1098	2129	40	7335	49753380.00•	49753380	10042	49486480.44	50184378
es40g	1162	2251	39	1430	45638998.00•	45638998	5318	45638998.00•	45638998
es40h	1248	2424	40	1593	48745996.00•	48745996	10129	48428944.91	49316377
es40i	1220	2369	40	4685	51761786.00•	51761786	10084	50165455.56	53025182
es40j	1246	2419	40	5718	57136850.00•	57136850	10144	56304351.90	57809710
es40k	1184	2293	40	9851	46734214.00•	46734214	10070	46309602.30	47098451
es40l	1252	2430	40	1837	43843376.00•	43843376	8388	43843376.00•	43843376
es40m	1373	2671	40	3940	51884545.00•	51884545	10125	51161520.74	52567117
es40n	1300	2528	40	2575	49166944.00•	49166944	10170	48453539.70	50733640
es40o	1303	2530	40	10022	50828064.00	51608580	10001	50133955.58	52261031

TABLE 3. Steiner tree instances series ES30 and ES40

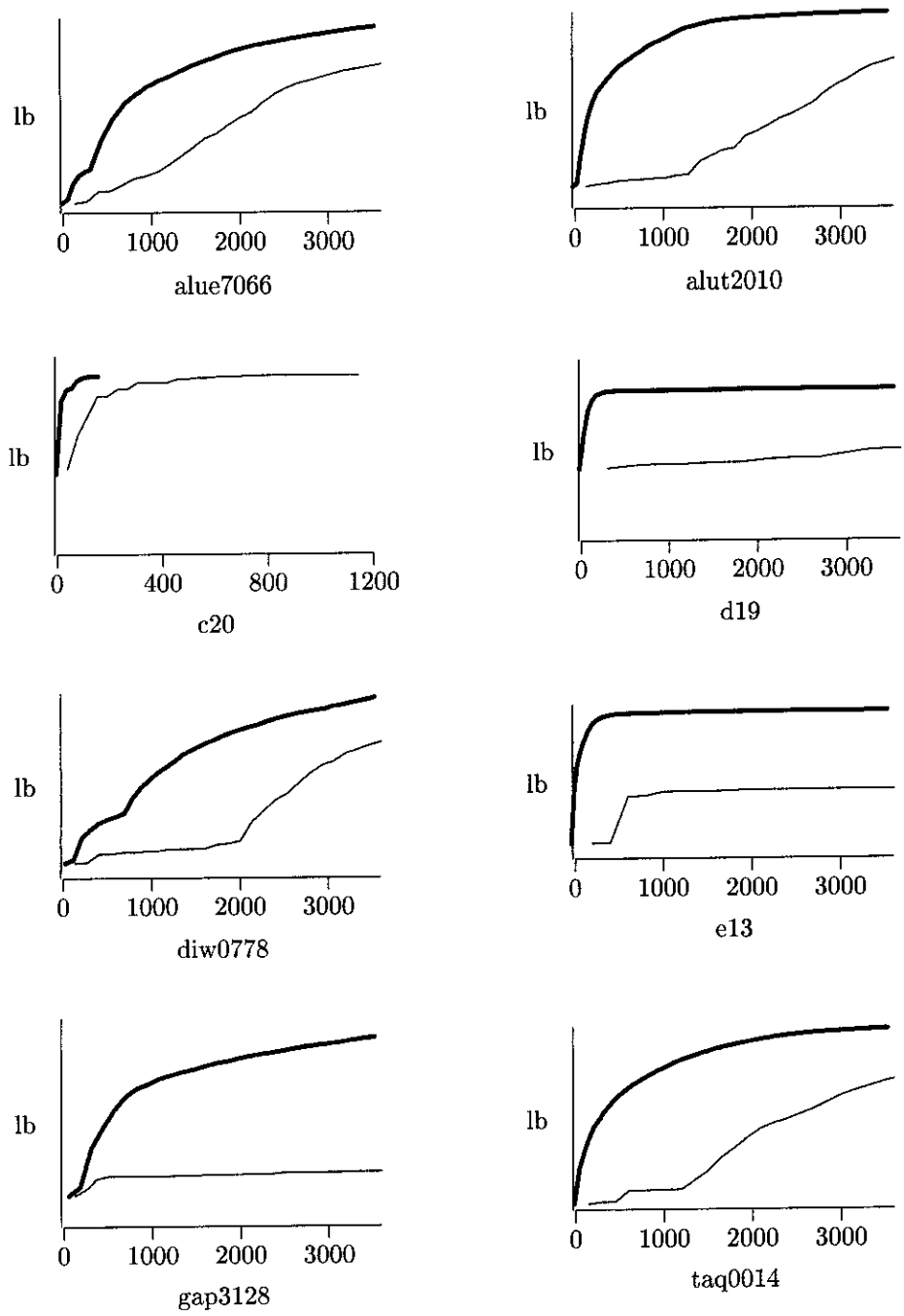

FIGURE 1. Lower bound vs. time elapsed (sec). Bold lines: VA. Thin lines: simplex.

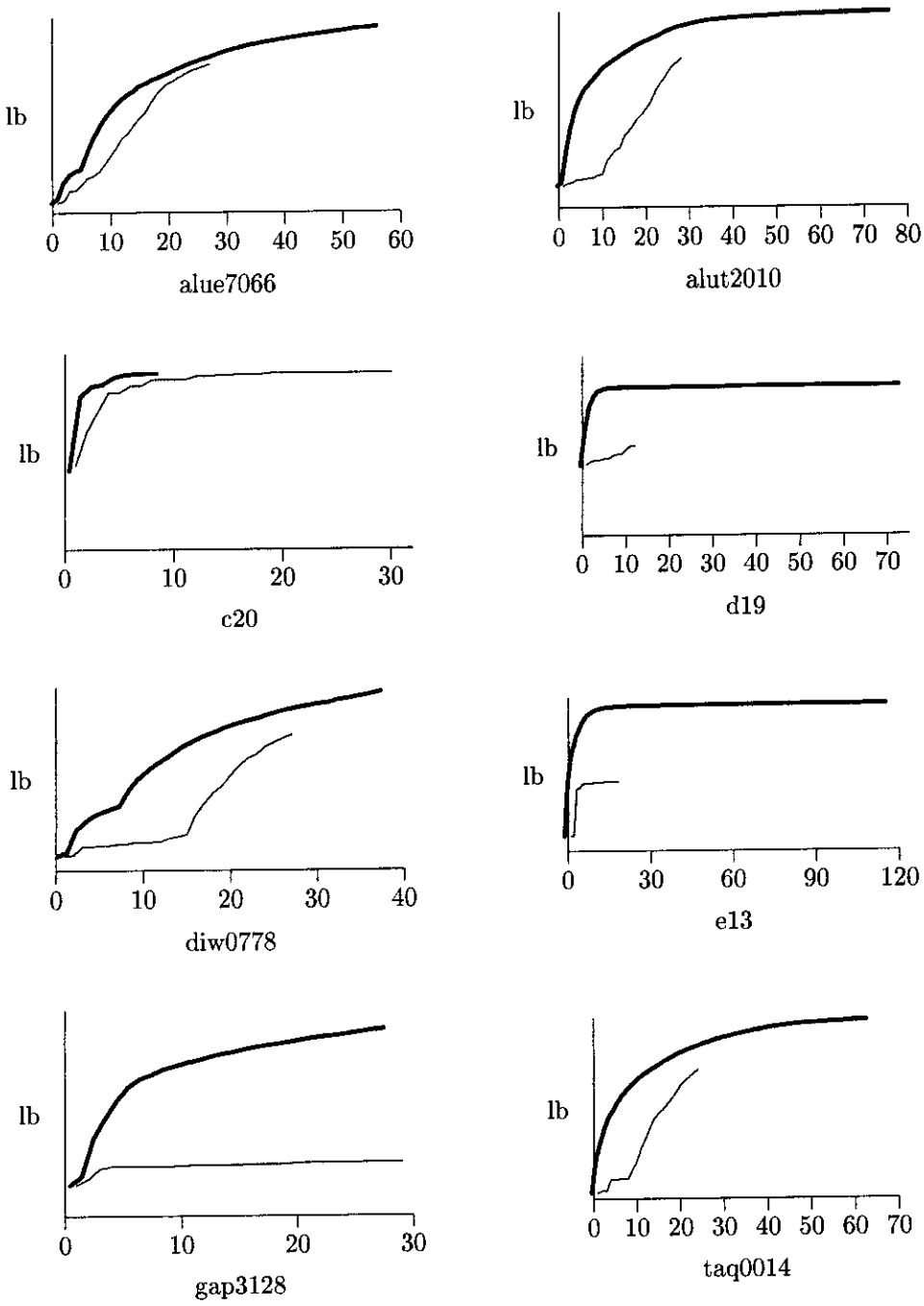

FIGURE 2. Lower bound vs. iteration count. Bold lines: VA. Thin lines: simplex.

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. We first consider complete graphs. B&C algorithms for complete or highly dense graphs have been presented in [8, 15]. Max-cut in complete graphs has also been treated with Semidefinite Programming in [23]. For a complete graph with n nodes, a linear relaxation of the integer programming problem is givem by

$$\begin{aligned} & \min c^T x \\ & \text{subject to} \\ x_{ij} + x_{jk} + x_{ik} &\leq 2 \quad (13) \\ x_{ij} - x_{jk} - x_{ik} &\leq 0 \quad (14) \\ -x_{ij} + x_{jk} - x_{ik} &\leq 0 \quad (15) \\ -x_{ij} - x_{jk} + x_{ik} &\leq 0 \quad (16) \\ 0 \leq x &\leq 1 \end{aligned}$$

Here x_{ij} should take the value 1 if the edge ij appears in the cut, and 0 otherwise. Constraints (13) - (16) are called the *triangle inequalities* and they define facets of the cut polytope, see [9]. These inequalities ensure that in every integral solution exactly 0 or 2 sides of a triangle will have value 1. It was pointed out in [15] that this linear program can be very difficult to solve with the simplex method, even for small graphs. Our experience is similar.

Another set of inequalities, that is a superset of (13) - (16) is the following. Let C be a cycle and $F \subseteq C$ with $|F| = 2k + 1$. Then

$$\sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |F| - 1 \quad (17)$$

is a valid inequality. This follows from the fact that the intersection of a cycle and a cut has even cardinality. Note, that although this set of inequalities include those in (13) - (16), the polytope defined by these is the same; i.e., these inequalities can be derived from those in (13) - (16) (see [9]). A polynomial time separation algorithm for this class of inequalities has been given in [9], however we use a faster heuristic as follows. 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 we have 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 (17), 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. Although, as we noted above, inequalities (17) are implied by (13) - (16), we use (17) because our simple separation heuristic is faster than enumerating triangles.

We have run our experiments on an IBM RS/6000 44P Model 270 workstation with a 375MHz processor and 2GB of core memory. We have randomly generated Max-Cut instances on 60, 80, and 100 nodes. Although we have generated complete graphs, only a certain fraction of the edges (20-50%) had nonzero weights. To smooth out the randomness we have generated 10 instances of each kind and took averages when reporting the results. In Tables 4 and 5 we report for each instance type the minimum, average and maximum runtime, gap in the root node, tree size and

	Runtime(sec)			Gap(%)			Tree size			LP / node		
Node/density	min	avg	max	min	avg	max	min	avg	max	min	avg	max
60/20%	7.5	11.8	31.8	0.00	0.00	0.03	1	1.4	3	4.0	9.8	13.0
60/30%	9.6	35.3	201.8	0.00	0.47	3.29	1	3.2	19	9.0	11.8	16.0
60/40%	8.7	66.4	364.5	0.00	0.90	5.29	1	6.2	35	9.4	13.5	22.0
60/50%	10.2	198.6	1259.7	0.00	2.34	8.56	1	19.0	123	10.0	12.3	16.0
80/20%	42.2	162.8	439.4	0.00	0.94	2.91	1	7.4	25	10.8	18.0	27.0
80/30%	47.7	197.5	305.2	0.00	1.28	2.29	1	9.6	15	11.1	14.1	27.0
80/40%	92.2	406.3	1362.6	0.07	2.13	5.18	3	21.0	79	10.8	15.7	22.7
80/50%	123.3	2577.3	11107.6	0.56	5.65	10.09	5	131.0	589	11.0	13.1	15.2
100/20%	167.9	2259.9	12171.2	0.18	2.60	6.37	3	92.2	521	11.0	14.8	28.3
100/30% (6)	293.1	1773.3	3500.3	0.55	3.45	5.08	7	66.7	141	11.3	13.8	18.0

TABLE 4. Max-Cut results on dense graphs with Volume Algorithm

	Runtime(sec)			Gap(%)			Tree size			LP / node		
Node/density	min	avg	max	min	avg	max	min	avg	max	min	avg	max
60/20%	54.4	176.9	460.1	0	0.07	0.34	1	1.8	5	16.4	23.9	32.0
60/30%	226.2	1130.9	4399.3	0	0.69	3.17	1	6.6	27	12.8	21.3	48.0
60/40%	110.1	2459.3	10144.8	0	1.30	6.51	1	9.6	37	11.4	23.1	51.0
60/50% (6)	746.1	5582.2	9083.6	0.05	3.08	6.69	3	19.33	31	10.9	13.8	18.3
80/20% (6)	4344.9	10731.0	19313.4	4	4.89	6.59	71	205.33	393	10.7	12.2	27.0
80/30% (0)	*	*	*	*	*	*	*	*	*	*	*	*
80/40% (0)	*	*	*	*	*	*	*	*	*	*	*	*
80/50% (0)	*	*	*	*	*	*	*	*	*	*	*	*
100/20% (1)	*	>19h	*	*	16	*	*	1109.00	*	*	10.6	*
100/30% (0)	*	*	*	*	*	*	*	*	*	*	*	*

TABLE 5. Max-Cut results on dense graphs with Simplex

average number of LPs per search tree node. If not all 10 instances were solved to optimality then we have indicated in parentheses how many were solved. The "*"s in Table 5 indicates that we couldn't solve a single instance of the given instance type within one day.

These tables speak for themselves. For the max-cut problem a VA based B&C algorithm is much faster and much more robust than a dual simplex method based one.

Now we discuss with sparse graphs. In this case one has to use inequalities (17). An application of the max-cut problem is to find ground states of Ising spin glasses, a model in Statistical Physics, that has been extensively studied [7, 4, 13, 14, 30]. Toroidal grids with edge-weights 1 and -1 have been identified as a challenging case for simplex based cutting plane algorithms. In [14] the authors report that a 70x70 grid takes on the order of 16 hours on a Sun SPARC 10 machine. Toroidal grids have also be treated with a cutting plane algorithm based on an interior point algorithm in [30]. In this reference times of the order of 2000 seconds. are reported for 70x70 grids, and 11000 seconds. for 100x100 grids on a Sun SPARC 20. In table 6 we report the times taken to prove optimality by our volume based algorithm and our simplex based algorithm. These are averages over 100 samples for each size. We decided to run the simplex based code only up to sizes 30x30 because of the large computing times.

We also tried a set of 70x70 grids reported in [14] as requiring of the order of 16 hours. In table 7 we show the times taken by our algorithms. The symbol * means that it was not solved within 10000 seconds. Again these tables speak for themselves.

Size	Volume (secs)	Simplex (secs)
10x10	0.082	0.092
20x20	0.625	26.63
30x30	3.59	1367.25
40x40	10.60	
50x50	26.57	
60x60	68.25	
70x70	125.81	
80x80	219.08	
90x90	385.95	
100x100	727.98	
110x110	1395.57	

TABLE 6. Max-cut in toroidal grids with weights 1 and -1.

Name	Volume (secs)	Simplex (secs)
L_70_1	75	*
L_70_2	51	*
L_70_3	98	*
L_70_4	220	*
L_70_5	101	*
L_70_6	204	*
L_70_7	62	*
L_70_8	121	*

TABLE 7. 70x70 grids with weights 1 and -1, from [14]. The symbol * means that the instance was not solved within 10000 seconds.

Finally, we ran experiments for a slightly different class of Ising spin glass problems: the weights of the edges on the toroidal grid are drawn from a Gaussian distribution and the whole grid is placed in an external magnetic field. The external magnetic field is represented by a node external to the grid. This node is connected to every node on the grid and these edges have the same weight. Here we have experimented with grids of size 80x80 and 100x100, and with 5 levels of external magnetic field. From each problem type we considered 15 instances. The time limit for each run was three hours. Table 8 displays the number of instances solved within the time limit and the minimum, average and maximum running times for the solved problems. As expected, the problems become easier with the increase of the external field (the maximum cut gets closer and closer to just being the star around the external node). For the low external field problems the simplex based code is not able to solve a single instance within the time limit, and it is consistently slower for the higher external field problems as well.

In Figure 3 we display the same data from a perspective similar to what we have already applied in presenting results for the Steiner tree problem. There we have displayed the lower bound as the function of if the number of iterations or time. Since the Max-Cut problems were randomly generated we have decided to present an average. For the 80-by-80 grids with 0.50119 external field we have computed the gap at every iteration then distributed them into buckets depending on at what percentage of the final iteration count time they were. Then we took the average gap for each bucket and rescaled the averages to the average iteration number across the problems. This way we got the picture of the average gap versus the iteration count. We created

		Volume				Simplex			
size	field	solved	min	avg	max	solved	min	avg	max
80	0.10000	15	1296	1827	2620	0	—	—	—
100	0.10000	13	2858	4337	7256	0	—	—	—
80	0.25119	15	651	887	2162	0	—	—	—
100	0.25119	15	1696	3473	10180	0	—	—	—
80	0.50119	15	441	597	1911	15	1270	1692	2115
100	0.50119	14	1204	1355	1523	11	4842	7828	10601
80	1.00000	15	170	281	351	15	320	362	395
100	1.00000	14	726	912	1042	15	784	969	1099
80	1.99526	15	67	97	133	15	95	119	138
100	1.99526	15	262	329	472	15	308	356	426

TABLE 8. Max-Cut: Ising spin glass problems with Gaussian distribution

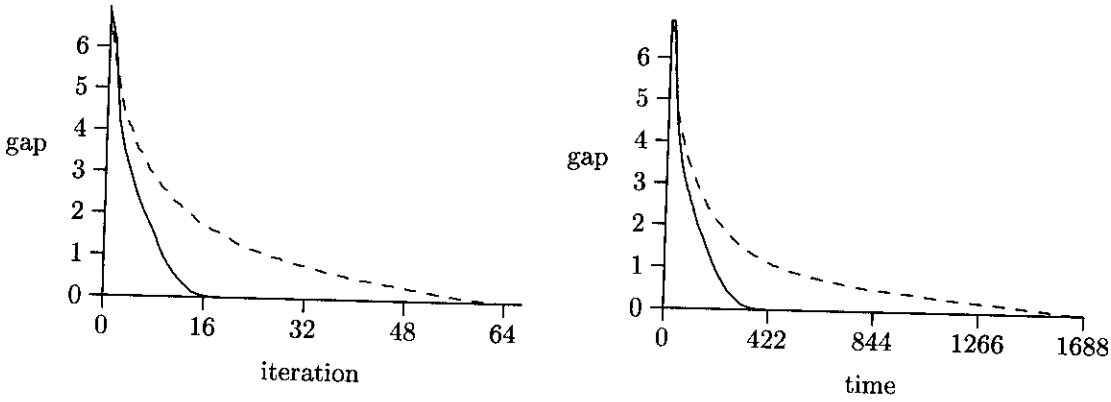

FIGURE 3. Max-Cut: Gap vs. Iteration count / Time

the average gap versus execution time plot similarly. On these plots the gap is displayed as a percentage, and the time is given is seconds.

Again, the volume algorithm based code performs much better than the simplex based code, both in terms of execution time and in terms of the gap at any given iteration.

7. 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, among them several previously unsolved problems. 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.

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.

All our code (including the Ising spin glass problem generator) are available as "open source" from COIN-OR [12].

8. Acknowledgements

We are grateful to Dr. T. Koch for providing us with his pre-processing code for Steiner tree problems and to Dr. Michael Jünger for sending us some max-cut instances from [14]. 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), pp. 167–178.

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

[3] L. Bahiense, N. Maculan, and C. Sagastizábal, *On the convergence of the volume algorithm*, tech. rep., 2000.

[4] F. Barahona, *Ground state magnetization of Ising spin glasses*, Physical Review B: Condensed Matter, 49(18) (1994), pp. 2864–2867.

[5] F. Barahona and R. Anbil, *On some difficult linear programs coming from set partitioning*, Research Report RC 21410, IBM Watson Research Center, to appear in Discrete Applied Mathematics, 1999.

[6] ———, *The volume algorithm: producing primal solutions with a subgradient method*, Mathematical Programming, 87 (2000), pp. 385–399.

[7] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt, *An application of combinatorial optimization to statistical physics and circuit layout design*, Operations Research, 36(3) (1988), pp. 493–513.

[8] F. Barahona, M. Jünger, and G. Reinelt, *Experiments in quadratic 0-1 programming*, Mathematical Programming, 44(2) (1989), pp. 127–137.

[9] F. Barahona and A. Mahjoub, *On the cut polytope*, Mathematical Programming, 36 (1986), pp. 157–173.

[10] A. Caprara and M. Fischetti, *Branch-and-cut algorithms*, in Annotated Bibliographies in Combinatorial Optimization, M. D. Amico, F. Maffioli, and S. Martello, eds., Wiley, 1997, pp. 45–63.

[11] 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), pp. 320–335.

[12] COIN-OR, `http:www.coin-or.org` .

[13] 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*, Journal of Statistical Physics, 80 (1995), pp. 487–496.

[14] C. De Simone, M. Diehl, M. Junger, P. Mutzel, G. Reinelt, and G. Rinaldi, *Exact ground states of two-dimensional +-J Ising spin glasses*, Journal of Statistical Physics, (1996).

[15] C. De Simone and G. Rinaldi, *A cutting plane algorithm for the max-cut problem*, Optimization Methods and Software, 3 (1994), pp. 195–214.

[16] J. Edmonds, *Optimum branchings*, J. Res. Nat. Bur. Standards, 71B (1967), pp. 233–240.

[17] L. F. Escudero, M. Guignard, and K. Malik, *A Lagrangean relax-and-cut approach for the sequential ordering problem with precedence relations*, Annals of Operations Research, 50 (1994), pp. 219–237.

[18] M. X. Goemans and Y. S. Myung, *A catalog of Steiner tree formulations*, Networks, 23 (1993), pp. 19–28.

[19] M. Guignard, *Efficient cuts in lagrangean "Relax-and-Cut" schemes*, European Journal of Operations Research, 105 (1998), pp. 216–223.

[20] M. Held and R. M. Karp, *The travelling salesman problem and minimum spanning trees*, Operations Research, 18 (1970), pp. 1138–1162.

[21] ———, *The travelling salesman problem and minimum spanning trees: Part II*, Mathematical Programming, 1 (1971), pp. 6–25.

[22] M. Held, P. Wolfe, and H. P. Crowder, *Validation of subgradient optimization*, Mathematical Programming, 6 (1974), pp. 62–88.

[23] C. Helmberg and F. Rendl, *Solving quadratic (0,1)-problems by semidefinite programs and cutting planes*, Mathematical Programming, 82 (1998), pp. 291–315.

[24] T. Koch and A. Martin, *Steinlib*, `http://elib.zib.de/steinlib/steinlib.php` .

[25] ———, *Solving Steiner tree problems in graphs to optimality*, Networks, 32 (1998), pp. 207–232.
[26] C. LEMARÉCHAL, *Nondifferentiable optimization*, in Optimization, Hanbooks in Operations Research, G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, eds., North Holland, 1989, pp. 529–572.
[27] A. LUCENA, *Steiner problem in graphs: Lagrangian relaxation and cutting-planes*, COAL Bull, 21 (1992), pp. 2–7.
[28] ———, *Tight bounds for the Steiner problem in graphs*, in Proceedings of Netflow93, 1993, pp. 147–154.
[29] A. LUCENA. AND J. BEASLEY, *A branch-and-cut algorithm for the Steiner problem in graphs*, Networks, 31 (1998), pp. 39–59.
[30] J. E. MITCHELL, *Computational experience with an interior point cutting plane algorithm*, SIAM Journal on Optimization, 10(4) (2000), pp. 1212–1227.
[31] M. PADBERG AND G. RINALDI, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Rev., 33 (1991), pp. 60–100.
[32] H. TAKAHASHI AND A. MATSUYAMA, *An approximated solution for the Steiner tree problem in graphs*, Math. Japonica, 254 (1980), pp. 573–577.
[33] P. WOLFE, *A method of conjugate subgradients for minimizing nondifferentiable functions*, Mathematical Programming Study, 3 (1975), pp. 145–173.

(F. Barahona) IBM T. J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, NY 10589, USA

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

(L. Ladányi) IBM T. J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, NY 10589, USA

E-mail address, L. Ladányi: ladanyi@us.ibm.com