Solving Steiner Tree Problems in Graphs to Optimality

T. Koch, A. Martin

Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustrasse 7, D-14195 Berlin–Dahlem, Germany

Received 11 February 1997; accepted 9 April 1998

Abstract: In this paper, we present the implementation of a branch-and-cut algorithm for solving Steiner tree problems in graphs. Our algorithm is based on an integer programming formulation for directed graphs and comprises preprocessing, separation algorithms, and primal heuristics. We are able to solve nearly all problem instances discussed in the literature to optimality, including one problem that—to our knowledge—has not yet been solved. We also report on our computational experiences with some very large Steiner tree problems arising from the design of electronic circuits. All test problems are gathered in a newly introduced library called *SteinLib* that is accessible via the World Wide Web. © 1998 John Wiley & Sons, Inc. Networks 32: 207–232, 1998

Keywords: branch-and-cut; cutting planes; reduction methods; Steiner tree; Steiner tree library

1. INTRODUCTION

Given an undirected graph $G = (V, E)$ and a node set $T \subseteq V$, a *Steiner tree for T in G* is a subset $S \subseteq E$ of the edges such that $(V(S), S)$ contains a path from s to t for all $s, t \in T$, where $V(S)$ denotes the set of nodes incident to an edge in S. In other words, a Steiner tree is an edge set S that spans T. The *Steiner tree problem* is to find a minimal Steiner tree with respect to some given edge costs c_e, $e \in E$. This problem is known to be $\mathcal{NP}$-hard (Karp [28]), even for grid graphs (Garey and Johnson [18]).

Nourished from the increasing demand in the design of electronic circuits, the solution of Steiner tree problems has received considerable and strongly growing attention in the last 20 years. Among the proposed solution methods are exact algorithms, heuristic procedures, approximation algorithms, polynomial algorithms for special instances, polyhedral approaches, preprocessing techniques, and more. Excellent surveys were given in Winter [40], Maculan [32], Hwang and Richards [25], and Hwang et al. [26]. To solve the Steiner tree problem to optimality, Aneja [1] proposed a row-generation algorithm based on an undirected formulation, Dreyfus and Wagner [11] and Lawler [29] used dynamic programming techniques, Beasley [4, 5] presented a Lagrangean relaxation approach, Wong [43] described a dual-ascent method, Lucena [31] combined Lagrangean and polyhedral methods, and Chopra et al. [8] developed a branch-and-cut algorithm. In particular, polyhedral methods have turned out to be quite powerful in finding optimal solutions for various Steiner tree problems. Reasons for that are the better understanding of the associated polyhedra, the availability of fast and robust LP solvers, and the experience gained to turn the theory into an algorithmic tool.

This paper moves within this framework and presents

Correspondence to: A. Martin

Mathematical Subject Classification (1995): 05C40, 90C06, 90C10, 90C35

 CCC 0028-3045/98/030207-26

a branch-and-cut algorithm. It is strongly related to the algorithm described in Chopra et al. [8]; we solve the same integer programming formulation, again by means of the separation of cutting planes. However, the new algorithm differs considerably, not only in several aspects of implementation but also due to some extensions. The main extensions are a more effective preprocessing phase by incorporating three preprocessing tests, an extension of the initial integer program with so-called flow-balance constraints, and a more careful and more efficient separation of active cut constraints resulting in leaner LPs. In Section 2, we review two different integer programming formulations. The second, on which the branch-and-cut algorithm is based, is a bidirected version of the first. In Section 3, we discuss preprocessing and exploit ideas known from the literature. In particular, our presolve algorithm includes three strong reduction techniques of Duin and Volgenant [13, 14]. Our computational results demonstrate how important preprocessing is: Without this tool, it would not have been possible to solve any of the large instances. Details of the cutting plane phase of our branch-and-cut algorithms are discussed in Section 4. It includes refined separation strategies (resulting in leaner LPs) and improved primal heuristics such that at an earlier stage the lower- and upper-bound values meet. Extensive tests are given in Section 5. We solve almost all test instances from the literature including one problem that—to our knowledge—has not yet been solved and find the optimal solution for many very large instances arising from real-world problems in the design of electronic circuits. We introduce a library for Steiner tree problems called *SteinLib* (including most of the models from the literature and all new VLSI-instances discussed in this paper). This library is available via anonymous ftp or from WWW at URL: ftp://ftp.zib.de/pub/Packages/mp-testdata/steinlib/.

2. INTEGER PROGRAMMING FORMULATION

In this section, we present the integer programming formulation that we are going to solve with our branch-and-cut algorithm. Let an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0, e \in E$, be given. We assume throughout the paper that the edge costs are nonnegative and integer. In addition, there is a node set $T \subseteq V$, called the *set of terminals.* We will denote an instance of the Steiner tree problem by the triple $\mathrm{ST}(G, T, c)$.

A canonical way to formulate the Steiner tree problem as an integer program is to introduce, for each edge $e \in E$, a variable x_e, indicating whether e is in the Steiner tree ($x_e = 1$) or not ($x_e = 0$). Consider the integer program

$$
\text{(uSP)}\qquad
\begin{array}{lll}
\min & c^T x & \\
(i) & x(\delta(W)) \geq 1, & \text{for all } W \subset V, \\
 & & W \cap T \neq \emptyset, \\
 & & (V \setminus W) \cap T \neq \emptyset, \\
(ii) & 0 \leq x_e \leq 1, & \text{for all } e \in E, \\
(iii) & x \text{ integer}, &
\end{array}
$$

where $\delta(X)$ denotes the cut induced by $X \subseteq V$, that is, the set of edges with one end node in X and one in its complement, and $x(F) := \sum_{e \in F} x_e$, for $F \subseteq E$. It is easy to see that there is a one-to-one correspondence between Steiner trees in G and $0/1$ vectors satisfying (uSP) (i). Hence, the Steiner tree problem can be solved via (uSP).

Another way to model the Steiner tree problem is to consider the problem in a directed graph. We replace each edge $[u, v] \in E$ by two antiparallel arcs (u, v) and (v, u). Let A denote this set of arcs and $D = (V, A)$, the resulting digraph. We choose some terminal $r \in T$, which will be called the *root.* A *Steiner arborescence* (*rooted at* r) is a set of arcs $S \subseteq A$ such that $(V(S), S)$ contains a directed path from r to t for all $t \in T \setminus \{r\}$. Obviously, there is a one-to-one correspondence between (undirected) Steiner trees in G and Steiner arborescences in D which contain at most one of two antiparallel arcs. Thus, if we choose arc costs $\vec{c}_{(u,v)} := \vec{c}_{(v,u)} := c_{[u,v]}$, for $[u, v] \in E$, the Steiner tree problem can be solved by finding a minimal Steiner arborescence with respect to $\vec{c}$. Note that there is always an optimal Steiner arborescence which does not contain an arc and its antiparallel counterpart, since $\vec{c} \geq 0$. Introducing variables y_a for $a \in A$ with the interpretation that $y_a := 1$, if arc a is in the Steiner arborescence, and $y_a := 0$, otherwise, we obtain the integer program

$$
\text{(dSP)}\qquad
\begin{array}{lll}
\min & \vec{c}^{\,T} y & \\
(i) & y(\delta^+(W)) \geq 1, & \text{for all } W \subset V, \\
 & & r \in W, \\
 & & (V \setminus W) \cap T \neq \emptyset, \\
(ii) & 0 \leq y_a \leq 1, & \text{for all } a \in A, \\
(iii) & y \text{ integer}, &
\end{array}
$$

where $\delta^+(X) := \{(u, v) \in A | u \in X, v \in V \setminus X\}$ for $X \subset V$, that is, the set of arcs with tail in X and head in its complement. Again, it is easy to see that each $0/1$ vector satisfying (dSP) (i) corresponds to a Steiner arborescence, and, conversely, the incidence vector of each Steiner arborescence satisfies (dSP) (i)–(iii). How are the models (uSP) and (dSP) related?

Polyhedral aspects of both models are intensively dis-

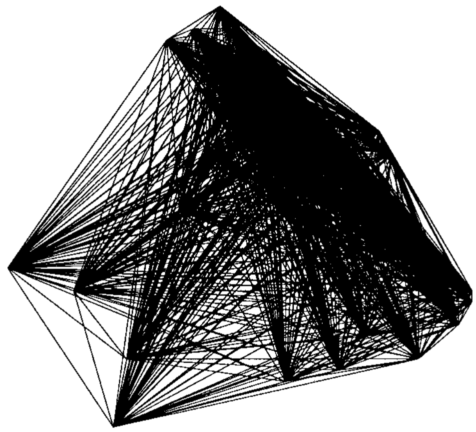

Fig. 1. Original problem.

cussed in the literature. The undirected model was studied in Grötschel and Monma [23], Goemans [19], Goemans and Myung [20], and Chopra and Rao [9, 10], whereas the directed version, in Ball et al. [3], Fischetti [17], Goemans and Myung [20], and Chopra and Rao [9, 10]. Chopra and Rao [9] and Goemans and Myung [20] related both formulations. Chopra and Rao [9] showed that the optimal value of the LP relaxation of the directed model $z_d := \min\{\tilde{c}^T y \mid y$ satisfies (dSP) (i) and (ii)$\}$ is greater or equal to the corresponding value of the undirected formulation $z_u := \min\{c^T x \mid x$ satisfies (uSP) (i) and (ii)$\}$. Even, if the undirected formulation is tightened by the so-called Steiner partition inequalities (see Grötschel and Monma [23]; Chopra and Rao [9]) and odd hole inequalities (see Chopra and Rao [9]), this relation holds. In addition, Goemans and Myung [20] showed that z_d is independent of the choice of the root r. These results suggest the directed model and we followed this suggestion. Nevertheless, one disadvantage of the directed model is that the number of variables is doubled. But it will turn out that this is not really a bottleneck, since we are minimizing a nonnegative objective function, and thus the variable of one of two antiparallel arcs will usually be at its lower bound.

It should be mentioned that further models to solve the Steiner tree problem can be found in the literature; for example, models based on flow formulations (Wong [43]; Maculan [32]) or models extending the undirected formulation by introducing node variables (Lucena [31]; Goemans and Myung [20]). Relations between relaxations of these and the above-discussed models can be found in Wong [43], Maculan [32], Duin [12], and Goemans and Myung [20].

3. PREPROCESSING

Preprocessing is a very important algorithmic tool in solving combinatorial and integer programming problems of large scale. The idea, in general, is to detect unnecessary information in the problem description and to reduce the size of the problem by logical implications. For the Steiner tree problem, many reduction methods are discussed in the literature and have been shown to be very effective for solving large instances; see, for example, Balakrishnan and Patel [2], Beasley [4], Chopra et al. [8], Duin [12], Duin and Volgenant [14], Lucena [31], Winter [41], and Winter and Smith [42]. These methods focus on detecting special configurations that allow one to neglect certain edges and/or nodes for the optimization or they show that some edges and/or nodes are contained in some optimal solution. In this section, we sketch the main concepts from the literature and show how they are incorporated in our code.

How successful preprocessing methods might be in reducing the size of some problem is demonstrated in Figures 1 and 2. Figure 1 shows the original graph of problem *br* (complete graph on 58 nodes; for a description of the problem, see Section 5), and Figure 2, the graph that we obtain after applying our preprocessing algorithm.

3.1. Degree-Test I

The following tests summarized under the name *degree-test I* (see [4]) are easy to check:

(i) A nonterminal node of degree one can be removed.
(ii) If a nonterminal node v is of degree two, node v and the two incident edges $[u, v]$ and $[v, w]$, $u \neq w$, can be replaced by an edge connecting u and w of cost $c_{[u,w]} = c_{[u,v]} + c_{[v,w]}$.
(iii) An edge incident to a terminal node of degree one is always in an optimal solution.
(iv) If an edge e is of minimal cost among the edges incident to a terminal node, and the other end node is also a terminal, then e is choosable in any optimal solution.

3.2. Special-Distance-Test

This test (introduced in Duin and Volgenant [13]) computes for each pair of nodes a number (called the special distance) which can be exploited to remove some edges.

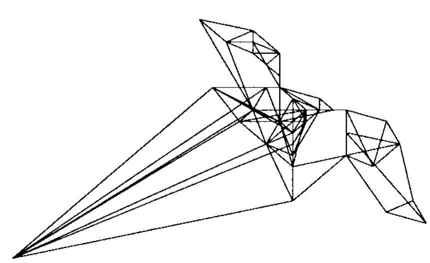

Fig. 2. Reduced problem.

Definition 3.1 (Special Distance). Let two nodes $u, v \in V$ with $u \neq v$ be given, and consider some path $P \subseteq E$ connecting u and v. Set $T_P = V(P) \cap T \cup \{u, v\}$ and let

$$b(P) = \max\{c(F) | F \subseteq P \text{ is a path connecting two nodes from } T_P \text{ such that } |T_P \cap V(F)| = 2\}.$$

The number

$$s(u, v) = \min\{b(P) | P \text{ is a path connecting } u \text{ and } v\}$$

is called the *special distance* (*between u and v*).

To give an idea what $s(u, v)$ means, consider each terminal as a petrol station and suppose you want to drive from location u to v. Then, $s(u, v)$ denotes the distance you must be able to drive without refilling if you choose among all possible routes. Note that the following relations

$$s(u, v) \leq d(u, v) \leq c_{[u,v]}$$

hold, where $d(u, v)$ denotes the length of a shortest path between u and v. The special distance can be computed by a modified shortest path algorithm (cf. Hwang et al. [26]).

Given the values $s(u, v)$ for all $u, v \in V$, there is an easy and very effective test for deleting edges. An optimal solution S^* of a Steiner tree problem $\mathrm{ST}(G, T, c)$ cannot contain any edge $[u, v] \in E$ with $s(u, v) < c_{[u,v]}$.

The *special-distance-test* is a generalization of many other tests known in the literature; this was comprehensively treated in Duin and Volgenant [13]. Concerning implementation, it should be noted that certain special cases of this test can be implemented more efficiently. However, one can also resort to a well-performing approximation of the special-distance-test that runs in $O(|V| \log |V| + |E| + |T|^2)$ (cf. Duin [12]).

3.3. Bottleneck Degree *m* Test

The *bottleneck degree m test* introduced in Duin and Volgenant [14] is the following: Consider some node $v \in N$ with $|\delta(v)| = m$. Let (W, F) be the complete graph on node set $W := V(\delta(v)) \setminus \{v\}$ with edge costs $\bar{s}_{[u,v]} = s(u, v)$ for $[u, v] \in F$. If, for all subsets $U \subseteq W$ with $|U| \geq 3$,

$$\bar{s}(B^*) \leq \sum_{u \in U} c_{[v,u]},$$

where B^* is the edge set of a minimal spanning tree in (W, F), holds, node v can be deleted, and for all $u, w \in W$, $u \neq w$, edge $[u, w]$ with cost $c_{[u,w]} = c_{[u,v]} + c_{[v,w]}$ has to be introduced. (In case of parallel edges, only one edge will be retained.) Of course, this might create many new edges, but, in general, most of these can be eliminated by the *special-distance-test.*

The running time for this test is $O(2^m \cdot \gamma)$, where γ denotes the time for computing a minimal spanning tree. Due to the exponential behavior, we perform this test only for $m \leq 3$. In fact, the bottleneck degree m test generalizes the ones in Section 3.1 (i), where $m = 1$, and Section 3.1 (ii), where $m = 2$.

3.4. Terminal-Distance-Test

In this test, we consider a connected subgraph $H = (W, F)$ of G with $T \cap W \neq \emptyset$ and $T \setminus W \neq \emptyset$. Let $e = \mathrm{argmin}_{e' \in \delta(W)} c_{e'}$ and $f = \mathrm{argmin}_{f' \in \delta(W) \setminus \{e\}} c_{f'}$ be a shortest and a second shortest edge of the cut induced by W.

Edge $e = [u, v]$ with $u \in W$ and $v \in V \setminus W$ is part of some optimal solution of $\mathrm{ST}(G, T, c)$ and can thus be contracted, if

$$c_f \geq d_u + c_e + d_v,$$

with $d_u = \min\{d(t, u) | t \in T \cap W\}$ and $d_v = \min\{d(t, v) | t \in T \setminus W\}$.

Duin and Volgenant [14] introduced this test and gave an implementation in $O(|V|^3)$ steps. In Duin [12], it is shown that the detection of all edges satisfying the condition of the terminal-distance-test needs only $O(|V|^2)$ steps. Note that the last two tests in Section 3.1 (iii) and (iv) are special cases of the terminal-distance-test. Two other special cases are the *R-R aggregation* method of Balakrishnan and Patel [2] and the *nearest vertex test* of Beasley [4].

3.5. Results

When it comes to implement these reduction methods, several questions arise: Which of these tests should be implemented? For each single test, should all cases be checked (complete test) which might result in high running times or should one restrict the search to certain promising special cases which might result in an incomplete test? In which order should the methods be called? How often should they be called? Some reduction of one test might give rise to further reductions by some other (already performed) test. These questions were already addressed in Duin and Volgenant [14]. With respect to our algorithm, we should also answer the questions: How much effort and computation time should one spent in the preprocessing phase? At what point is it usually better to switch over to the branch-and-cut phase? We tried to find answers in the following way: First, we implemented all the tests and each test in the complete version. We called all these tests consecutively and iterated this process until no more reductions could be found. Of course, this might be very time consuming but for large difficult problems it might be worth to reduce as much as possible (see Section 5). For small and medium-sized problems, the situation is different. Often it did not pay to perform a complete test, but rather to switch to the branch-and-cut phase which usually solved the (reduced) problem very fast. We performed many test runs to find a balance between the total running times and the success of the reduction methods. Algorithm 3.2 shows our default selection:

Algorithm 3.2 (Default Presolve)

(1) Degree-Test I
(2) Special-Distance-Test
(3) Degree-Test I
(4) Terminal-Distance-Test
(5) Special-Distance-Test
(6) Degree-Test I
(7) Special-Distance-Test
(8) Degree-Test I
(9) Return

Note that the bottleneck degree *m* test is not included in our default strategy. For some difficult instances, however, it pays to use the bottleneck degree *m* test and iterate all four tests as along as there is some reduction possible. The success of our presolve strategy is illustrated in Section 5.

4. IMPLEMENTATION DETAILS

In this section, we describe the implementation of our branch-and-cut algorithm for solving the Steiner tree problem. We assume that the reader is familiar with the general outline of a branch-and-cut algorithm (see Caprara and Fischetti [7] for a survey). Algorithm 4.1 presents the main steps of such an algorithm:

Algorithm 4.1 (Branch-and-Cut Algorithm)

(1) Initialization
(2) **repeat**
(3) select a leaf from the tree and consider the associated LP
(4) **repeat** (*iterate*)
(5) solve the LP
(6) call primal heuristics
(7) separate violated inequalities and add them to the LP
(8) **until** there are no violated inequalities
(9) branch if necessary, otherwise remove the leaf from the tree
(10) **until** branch-and-bound tree is empty
(11) print the optimal solution
(12) **STOP.**

In the *Initialization* phase, we set up the first LP and initialize the branch-and-bound tree with the root node representing the whole problem. In our case, the starting LP is essentially empty, consisting only of the trivial inequalities (dSP) (ii). We experimented with initial cuts for the first LP by doing a breadth-first search from the root to every other terminal and adding the cuts between nodes of different depth. Although these cuts have disjoint support for each root-terminal pair, only the smaller instances profited from this idea. While the number of cutting plane iterations [i.e., the number of runs through Steps (4)–(8)] needed to solve the problems was always smaller, the effect from initially having a lot of dense inequalities (i.e., inequalities with many nonzero entries) in the LP considerably slows down the whole process.

For solving the linear programs, we used CPLEX*, Versions 4.0.9 and 5.0, a very fast and robust linear programming solver, which features both a primal and dual simplex solver and a primal-dual barrier solver. We used the dual simplex algorithm, since the LPs from one iteration to the next stay dual feasible, when cutting planes are added or variables are fixed to one of their bounds. It turned out that the best pricing strategy was steepest-edge pricing, that is, to select a variable entering the basis that has largest (obtuse) angle with the gradient of the objective function. However, for some instances (in particular, for large grid problems), the arising LPs are highly primal and dual-degenerated.

We tried to avoid degeneracy by perturbing the objective function. We used $\tilde{c} = \vec{c} - b\varepsilon_a$, where $b = \min\left(10^{-1}, \frac{1}{2(|A| + 1)}\right)$, and $\varepsilon_a \in [0, 1)$ is some uniformly distributed random number for each $a \in A$. Since we assumed that c is an integer, our choice of $\tilde{c}$ ensures that an optimal solution with $\tilde{c}$ is also optimal for $\vec{c}$. The running times for solving the LPs were always better with the perturbed objective function than with the original. Nevertheless, some of the larger problems continued to show signs of degeneracy. We tried two further ways to remedy degeneracy: First, we perturbed the objective function with $b = 0.1$. This, however, requires reoptimization with the original objective function after the problem has been solved for the perturbed objective function. Sometimes, this reoptimization step needed several thousand simplex iterations and we obtained a significant speed up only in very few cases. Second, we tried the

* CPLEX is a registered trademark of ILOG.

primal-dual barrier solver of the CPLEX package. The barrier code does not suffer from degeneracy, but has to solve each LP from scratch so that, on average, it could not outperform the dual simplex method with our initial perturbation. Thus, our default choice to solve the LPs is to use the perturbed objective function $\tilde{c}$ and apply the dual simplex algorithm with steepest-edge pricing.

4.1. Branching and Selecting Leafs

In Step (9), if it is necessary to branch, we use strong branching (Bixby [6]), that is, we determine a set of variables whose LP value is close to 0.5, perform for each variable of this set a certain number of simplex iterations for the linear program where the variable is set to one or zero, and, finally, select the variable in the set as branching variable that obtained the best increase in the LP value.

We run through the branching tree in a depth-first-search fashion. The reasons are that the memory requirements for the whole tree stay small and that we try to find a good primal solution as soon as possible. It almost never happened that our branching tree grew to much. Branching was a rather rare event in our computations anyway (within the time limit and with the default parameter setting branching was necessary only in 37 of 414 cases; see Section 5).

4.2. Primal Heuristic

The primal heuristic that we use is basically the one introduced by Takahashi and Matsuyama [38]. The idea of this heuristic is to start from one terminal and connect a terminal by a shortest path that is closest to the starting terminal. The next terminal is chosen among the remaining terminals in such a way that it is closest to the already existing path or subtree in general. This process is continued until all terminals are connected. As edge costs for this heuristic we use $(1 - x_e) \cdot c_e$ for $e \in E$, if x is the optimal solution of the current LP, that is, we try to prefer those edges that are already chosen in the LP solution. (A slightly different objective function was suggested in Lucena [31] who used as edge costs 0, if $x_e = 1$, and c_e, otherwise; Chopra et al. [8] used the original edge costs, but considered only edges e with positive LP value $x_e > 0$.) As suggested in Rayward-Smith and Clare [36], we also try to improve the heuristic solution by computing a minimal spanning tree among the chosen nodes and prune nonterminal leaves as long as possible.

A parameter to be specified for this heuristic is the starting terminal. Since running the algorithm for all terminals is usually too time-consuming, we made the following selection: We always try the terminal which gave the best solution so far and try in addition up to 10 randomly selected terminals. The frequency in which the heuristic is called in our code is specified by some parameter (default is every five cutting plane iterations).

In 138 out of 414 test examples, the first call to the heuristic found the optimal solution, and in 90% of the cases, the gap [(heuristic solution − lower bound)/lower bound] was below 5%.

We also experimented with the Rayward-Smith [35] heuristic. The results are quite promising; however, a main bottleneck is the running time, especially for big problems. The reason is that the heuristic requires all-to-all node distances, and due to memory limitations, we must compute these on the fly, so most of the time is spent for calculating shortest paths.

4.3. Separating Inequalities

In this section, we start with the description of our separation routines for the cut inequalities (dSP) (i). We first discuss how the generic cut separation works and give an efficient implementation. In the following three subsections, we discuss three improvements on the generic cuts: *back* cuts, *nested* cuts, and *creep-flow* cuts. All these cuts aim at selecting violated cuts that give the most progress in terms of an increase in the lower bound with respect to the running time. We finally present a further class of inequalities, the so-called *flow-balance* inequalities, that may strengthen the LP relaxation further. All four suggestions can be combined with each other, resulting in 16 possible ways to separate inequalities. Based on some computational tests in the last subsection, we present our final separation strategy.

4.3.1. Generic Cuts

It is well known that the separation problem for the cut inequalities (dSP) (i) can be solved by any max-flow algorithm and can thus be solved in polynomial time. We regard the LP solution as capacities in the graph and check, for each $t \in T \setminus \{r\}$, whether the minimal (r, t)-cut is less than one. If so, a violated cut inequality is found; otherwise, there is none. We add inequalities only if they are violated by at least some epsilon (default is 10^{-4}). To determine a minimal (r, t)-cut, for all $t \in T \setminus \{r\}$, we must call, in principle, $(|T| - 1)$ times a max-flow algorithm. However, Hao and Orlin [24] showed that by a careful implementation of a preflow-push algorithm the time to determine minimal cuts from the root node r to all other nodes is comparable with the time to find a single (r, t)-cut. If we use the highest label preflow-push algorithm of Goldberg and Tarjan [21] the overall running time of the Hao–Orlin algorithm to determine a minimal (r, t)-cut, for all $t \in T \setminus \{r\}$, is $O(|V|^2|E|^{1/2})$.

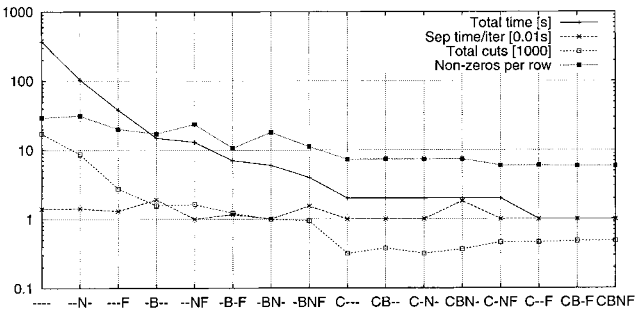

Fig. 3. *alue7229.*

4.3.2. Back Cuts

Chopra et al. [8] described a method to increase the number of separated inequalities by swapping the flow on each arc and checking in addition all (t, r)-cuts, for $t \in T\setminus\{r\}$. A drawback here is that we cannot use the speed-up feature mentioned above, since for each (t, r)-cut computation, the source node changes and thus the algorithm has to start from scratch again. However, as we will see in Section 4.3.6, this idea significantly improves the overall running time compared with the generic cut generation.

4.3.3. Nested Cuts

Another way to increase the number of violated inequalities is to *nest* the cuts. After finding a minimal cut between r and some terminal t, we temporary fix all corresponding variables in the actual LP solution to one and try again to find a cut between r and t. We repeat this procedure until the flow between r and t is at least one. This idea can be combined with *back* cuts so that we are trying to find *nested* inequalities in both directions. The results of this procedure are usually an increase in the time spent for separation and reoptimization the linear programs per iteration, while the total number of cutting plane iterations drastically decreases, resulting in a total running time of about one magnitude faster than without *nested* and *back* cuts.

4.3.4. Creep-Flow

We obtained another major speedup in the performance of our algorithm when we implemented the following idea: Instead of trying to increase the number of separated inequalities, we tried to raise the ''quality'' of the inequalities. Since most of the variables in our LP solution are zero, the optimal solution of the min-cut algorithm is not necessarily arc minimal. So, we add a tiny capacity of some ϵ (in the code we use $\epsilon = 10^{-6}$) to all arcs to get among all weight minimal cuts one that is also arc minimal. While this increased the running time for computing a minimal cut, since much more arcs have to be considered, the time needed for reoptimization the linear programs decreased by a factor of 10. Moreover, the reduction in the number of cutting plane iterations by using these ideas over just adding pure (r, t)-cuts is between two and three orders of magnitude.

4.3.5. Flow-Balance Inequalities

In our cutting plane phase we take another class of inequalities into consideration. An (optimal) Steiner arborescence can be viewed as a set of flows sending one unit from the root r to each terminal in $T\setminus\{r\}$. This means that for all nonterminal nodes that are not branching nodes in the Steiner arborescence the flow-balance equality $y(\delta^-(v)) = y(\delta^+(v))$ must hold, and for the other nonterminal nodes, $y(\delta^-(v)) \le y(\delta^+(v))$. This is expressed in the following set of inequalities:

$$
\begin{aligned}
& y(\delta^-(v)) \begin{cases} = 0, & \text{if } v = r; \\ = 1, & \text{if } v \in T\setminus\{r\}; \\ \le 1, & \text{if } v \in N; \end{cases} \\
& y(\delta^-(v)) \le y(\delta^+(v)), \quad \text{for } v \in N; \\
& y(\delta^-(v)) \ge y_e, \quad \text{for all } e \in \delta^+(v), \ v \in N.
\end{aligned}
\tag{1}
$$

Note that this system of inequalities is not valid for all Steiner arborescences (e.g., cycles are cut off), but there is always an optimal solution that satisfies these conditions, since the objective function is nonnegative. Note that the addition of inequalities (1) to (dSP) has already been considered in Duin [12]. He gives an example where these inequalities strengthen the LP relaxation.

4.3.6. A Comparison

We performed several tests to evaluate the performance of these four separation routines and all its combinations. Figures 3 and 4 show the results of all 16 possible separation strategies for examples *alue7229* and *taq0631* (for

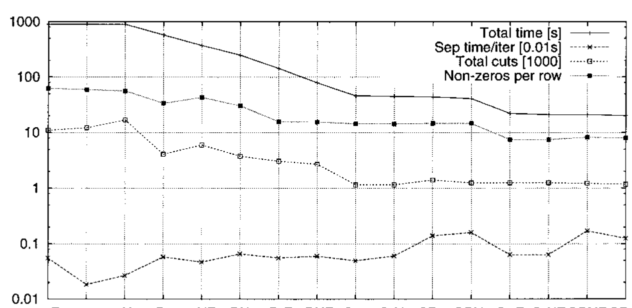

Fig. 4. *taq0631.*

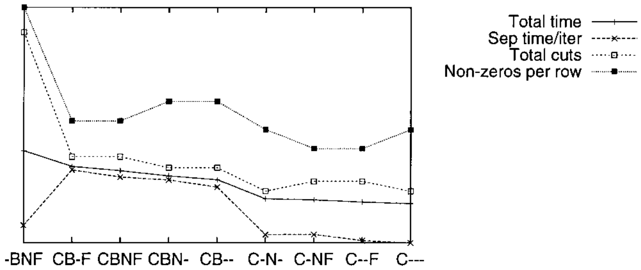

Fig. 5. *gr.*

a description of these problems, see Section 5). F means that flow-balance inequalities are applied; C, B, and N, indicates that creep-flow cuts, back cuts, and nested cuts are added, respectively; and ''----'' indicates that just the generic cut separation is applied. The x-axis shows the 16 possibilities sorted according to their total running time in decreasing order from left to right. The curves depict the total running time (in seconds), the separation time per iteration (in hundredth of seconds), the total number of added cuts (in thousands), and the average number of nonzeros per row. We observe that the differences in the running times are up to two orders of magnitude (note that the y axis is logarithmically scaled). We also see that the strategy ''-B--'' that was used in Chopra et al. [8] gives a significant speed up compared with just adding generic cuts, although the separation time per iteration increases. This was already observed by Chopra et al. However, their strategy is not the overall best. For almost all combinations, it is better to apply flow-balance inequalities. The same holds for creep-flows; the strategy with creep-flows is always better than the one without. The reason is mainly a significant reduction in the number of generated cuts. In both figures, the eight combinations using creep-flows together with ''-BNF'' are always the best. We evaluated these nine strategies on some larger instances. Figures 5 and 6 show the results for problem *gr* and *msm1234* (note that the curves are not uniformly and the y-axis is not logarithmically scaled any more to better illustrate the differences of the strategies).

The ''-BNF-'' strategy shows a big increase in the number of cuts and nonzeros resulting in high LP times (LP times are not shown in the diagram). The increase in the LP time per iteration is not completely compensated by the decrease in the total number of iterations, resulting in running times that are not among the best. For larger instances, this effect becomes even clearer. Again, we recognize the positive impact of the flow inequalities. For example, in problem *gr*, the ''C--F'' strategy has the best nonzero per row index. In fact, this strategy is very robust: It is always among the four best, while the performance of the other strategies does not seem to be predictable. It is remarkable that the connection of C with B and N (with or without F) does not outperform ''C--F.'' Therefore, we have chosen the ''C--F'' option as the final separation strategy in our branch-and-cut algorithm.

4.4. Removing Inequalities

Sometimes in the iteration process inequalities become nonbinding, that is, the slack of the inequalities are positive. In these cases the inequality can be removed from the LP without changing its optimal value. Although the inequality can be violated again, it is, in general, a good idea to remove these inequalities in order to keep the LP small. To minimize the occurrences of these reviolated inequalities, we added a ''life'' counter to each inequality currently in the LP. If the slack of an inequality is nonzero, the counter is decreased; if the slack is zero, it is reset to an initial value (in our implementation 5). If the counter reaches zero, the inequality is removed. This way we are delaying the removal of inequalities to a point where it is more likely that it will never be used again.

4.5. Reduced Costs and Reduced Set of Variables

Every time the primal heuristic finds a better solution, we try to fix variables by the reduced-cost criterium. For a discussion on reduced-cost fixing, see for instance Padberg and Rinaldi [34]. With the exception of the class of so-called *incidence* problems (see Section 5), this idea has little effect on the performance of our algorithm. Due to the high degeneracy of the LPs, the reduced costs tend to be very small and, thus, the reduced-cost criterium (and possibly also other reduction methods that are based on reduced costs, see Duin [12]) are likely to fail.

Another commonly known idea is to work only on a reduced set of variables by fixing variables temporally to one of its bounds. After the problem has been solved on the reduced set, we check the reduced costs of the temporally fixed variables, add them if necessary to the current set of variables, and reoptimize. Instead of really removing the variables that are fixed from the problem as it is usually done in such a type of column-generation algorithm, we only fix these variables to their bounds and keep them in the LP. CPLEX (the LP solver that we use)

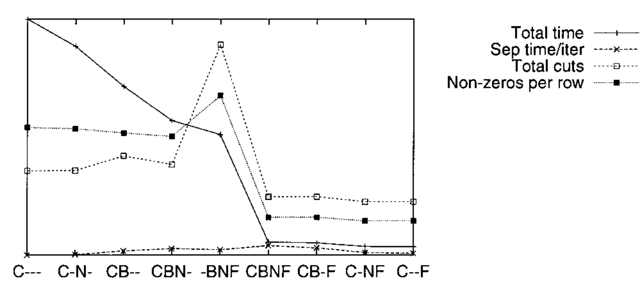

Fig. 6. *msm1234.*

manages fixed variables very efficiently so that we could not detect a major loss of performance (under the assumption that limits of memory are not reached). The advantage is that we do not have to take care of the management of inequalities for which some of the variables are in the current set of variables and some are not. For the limited test runs that we performed for this column-generation idea, we could not obtain a speedup on average.

5. COMPUTATIONAL RESULTS

In this section, we report on computational experiences with our branch-and-cut algorithm. Our code is implemented in C and all runs (with the exception of the *incidence* problems, see the relevant page in the sequel) are performed on a Sun SPARC 20 Model 71. The test examples include public available benchmarks discussed in the literature, some instances that authors of other Steiner tree codes made us available, and some realistic problems arising in the design of electronic circuits. All instances are gathered in the library *SteinLib* that is available via anonymous ftp or the World Wide Web.†

The format of our tables is as follows: The first column gives the problem name and columns 2–4 and 5–7 give the number of nodes, edges, and terminals of the original problem and the reduced problem, respectively. Comparing these two sets of columns reflects the success of our preprocessing algorithm. The next three columns give statistics about the branch-and-cut algorithm. *Nod* contains the number of branch-and-bound nodes (1 means that no branching was necessary), *Iter* gives the number of cutting plane iterations, and *Cuts* gives the number of violated cuts added to the LP. The following three columns provide information of the root LP, which is the final linear program if no branching was necessary and, otherwise, the last linear program before branching. *Frac* denotes the number of fractional variables in the root LP, and *Rows* and *NZ*, the number of rows and nonzeros. Then, time statistics follow: *Pre* stands for presolve time; *Heu*, for the heuristic time; and *LP*, for the time spent to solve the LPs; the separation time is shown in column *Sep*, and, finally, *Tot* gives the whole running time to solve the problem. The times are in CPU seconds. The time limit for all runs (with the exceptions of *e18*, *diw0234*, and some *incidence* problems) was 10,000 seconds. The last three columns show the solution values. *Heu(1)* is the value of the solution found by the first call to the primal heuristic, that is, when no linear programming solution is at hand ($x = 0$). Comparing this value to the lower bound depicted in column *LB* provides information about the quality of the primal heuristic. If the difference between the lower and upper bound is less than 1, the upper bound in the last column is shown in bold face to indicate that the optimal solution was found. If there is still a gap greater than 1 between *LB* and *UB*, we have not found the optimal solution within the time limit.

Tables I and II show our results for the test series introduced by Beasley [5]. Test set C is easy: We solve all instances with one exception within a minute. Interesting to note is that already the first call to the heuristic (without any dual information) gives in 11 out of 20 examples the optimal solution. Series D with 1000 nodes is a bit more difficult: The running times increase up to 6 minutes. However, the optimal solution is obtained in the root node in all but one case (*d19*), that is, branching was not necessary. To solve test series E (with the exception of *e18*), we need up to 2 hours per instance, although still no branching is necessary. The number of cuts needed to solve these examples increases to about 66,000. We could not detect a correlation between the number of violated inequalities and the number of variables or terminals. The number of inequalities in the final LP is rather high compared with the number of cuts separated. This means that the inequalities mostly stay in the LP whenever they are added and elimination does not happen too often. The exception of test series E is *e18*. To the best of our knowledge, nobody solved this problem up to now to optimality. We are able to solve it within half a day of CPU time, where Algorithm 3.2 was replaced by a complete reduction test. *e18* and *d19* are the only examples of Beasley's test set where branching was necessary with the default parameter setting.‡

Figure 7 depicts a diagram of the run for *e16*. The *x*-axis shows the number of cutting plane iterations (i.e., the number of LPs that have been solved). There were 267 iterations for example *e16* (see Table II). The curves in the diagram illustrate trends of certain numbers in the course of the algorithm. The top curve *Integer* gives the number of variables that are integer in the actual LP solution. In the first LP, all variables are integer, since we start just with the trivial inequalities (see above). Thus, the straight line indicates that almost all variables are integer during the whole run of the algorithm (we will see different patterns for other problems in a moment). The next two curves show the upper and lower bounds. The horizontal line for the upper bound means that the first call to the heuristic already found an optimal solution. The lowest curve gives the number of rows in the LP. Its steady increase demonstrates the effect just mentioned that elimination of inequalities from the LP does not frequently happen. This property is common to many test

† URL: ftp://ftp.zib.de/pub/Packages/mp-testdata/steinlib.

‡ In fact, branching was only necessary to obtain an optimal solution; the objective function value of the root LP rounded up already yields the optimal solution value.

TABLE I. Beasley's test sets C and D

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
c01	500	625	5	138	247	5	1	23	110	25	106	872	0.1	0.0	0.1	0.2	0.5	85	84.7	**85**
c02	500	625	10	120	218	8	1	30	212	0	196	1404	0.1	0.1	0.3	0.3	0.9	144	144.0	**144**
c03	500	625	83	100	157	47	1	8	373	0	350	2381	0.1	0.2	0.2	0.3	1.0	755	754.0	**754**
c04	500	625	125	96	145	52	1	8	322	0	298	1686	0.1	0.3	0.2	0.2	0.9	1080	1079.0	**1079**
c05	500	625	250	45	61	35	1	13	204	14	175	989	0.1	0.1	0.1	0.1	0.5	1579	1578.5	**1579**
c06	500	1000	5	368	839	5	1	48	269	34	265	2829	0.2	0.2	0.9	1.3	2.9	55	54.2	**55**
c07	500	1000	10	380	856	9	1	35	399	70	384	4298	0.2	0.3	1.0	1.6	3.3	102	101.5	**102**
c08	500	1000	83	337	669	70	1	23	1486	0	1368	19,680	0.7	2.7	3.1	5.8	13.0	510	509.0	**509**
c09	500	1000	125	288	517	95	1	17	1397	26	1290	18,211	0.7	5.3	2.2	3.8	12.7	715	706.2	**707**
c10	500	1000	250	112	165	74	1	7	410	13	368	2185	0.7	0.8	0.2	0.3	2.3	1093	1092.5	**1093**
c11	500	2500	5	498	2045	5	1	118	418	89	406	8480	2.7	1.6	5.3	6.8	16.7	32	31.2	**32**
c12	500	2500	10	493	1786	10	1	75	550	14	543	10,008	2.6	1.2	3.7	4.9	12.8	46	45.5	**46**
c13	500	2500	83	420	969	79	1	25	1916	8	1651	25,911	2.4	3.9	5.4	8.3	20.9	262	257.2	**258**
c14	500	2500	125	333	643	99	1	15	1409	0	1341	21,762	2.0	5.3	1.6	5.0	14.5	324	323.0	**323**
c15	500	2500	250	180	284	102	1	9	750	0	702	6952	1.8	2.6	0.5	1.2	6.6	557	556.0	**556**
c16	500	12,500	5	500	3504	5	1	22	144	14	144	3499	14.8	0.8	1.0	2.6	19.6	11	10.5	**11**
c17	500	12,500	10	500	3002	10	1	75	583	14	573	15,112	12.6	1.8	5.2	8.7	28.9	19	17.5	**18**
c18	500	12,500	83	471	1384	80	1	40	3146	149	1930	51,146	9.9	5.9	56.1	31.6	104.8	120	112.2	**113**
c19	500	12,500	125	446	1094	117	1	24	2390	0	2195	39,274	9.0	9.1	9.4	13.3	41.8	150	146.0	**146**
c20	500	12,500	250	201	351	114	1	7	607	0	606	4485	7.6	3.8	0.4	0.9	13.2	268	267.0	**267**
d01	1000	1250	5	273	507	5	1	39	231	45	203	1872	0.2	0.2	0.6	0.7	1.8	106	105.2	**106**
d02	1000	1250	10	284	521	10	1	28	304	74	288	2309	0.1	0.1	0.4	0.8	1.7	220	219.6	**220**
d03	1000	1250	167	186	288	84	1	7	625	0	585	4208	0.3	1.6	0.4	0.6	3.4	1570	1565.0	**1565**
d04	1000	1250	250	126	183	73	1	13	533	0	464	3427	0.4	1.1	0.3	0.6	2.8	1936	1935.0	**1935**
d05	1000	1250	500	66	93	51	1	8	243	24	227	1244	0.3	0.3	0.1	0.1	1.0	3252	3250.0	**3250**
d06	1000	2000	5	761	1738	5	1	129	639	196	515	8056	0.6	1.6	7.0	7.3	17.0	70	66.1	**67**
d07	1000	2000	10	747	1708	10	1	100	564	52	539	7486	0.8	1.8	4.2	7.0	14.1	103	102.3	**103**
d08	1000	2000	167	661	1307	151	1	22	3302	9	3042	53,930	2.8	37.1	11.6	26.6	79.8	1092	1071.5	**1072**
d09	1000	2000	250	531	946	199	1	12	2340	121	2214	27,952	3.1	58.1	3.1	11.3	77.5	1462	1447.3	**1448**
d10	1000	2000	500	230	348	156	1	11	1212	0	968	9307	2.8	13.9	1.0	2.6	21.3	2113	2110.0	**2110**
d11	1000	5000	5	991	4390	5	1	157	599	73	556	14,399	11.1	4.5	12.9	23.5	52.7	29	28.1	**29**
d12	1000	5000	10	996	3824	10	1	107	784	95	758	16,354	12.7	3.8	10.7	19.3	47.1	42	41.1	**42**
d13	1000	5000	167	833	1890	156	1	20	2799	0	2683	47,796	11.3	42.4	7.6	23.9	87.1	510	500.0	**500**
d14	1000	5000	250	707	1430	213	1	21	3861	133	3513	77,303	10.6	89.4	10.5	49.6	162.4	675	666.8	**667**
d15	1000	5000	500	321	514	192	1	10	1549	26	1398	17,553	8.8	32.6	1.8	4.8	49.2	1120	1115.5	**1116**
d16	1000	25,000	5	1000	8621	5	1	70	394	25	392	13,565	101.5	4.9	8.6	25.1	140.8	13	12.3	**13**
d17	1000	25,000	10	1000	8035	10	1	115	1104	50	1091	46,932	98.4	9.1	31.3	57.4	197.1	23	22.1	**23**
d18	1000	25,000	167	948	2922	160	1	35	5359	86	4117	93,786	59.7	48.6	76.8	120.4	308.0	238	223.0	**223**
d19	1000	25,000	250	922	2514	235	3	47	8577	182	5078	243,180	55.8	419.5	105.2	281.7	868.8	325	310.0	**310**
d20	1000	25,000	500	523	975	295	1	12	2332	0	2148	26,168	46.7	121.5	3.3	34.7	208.6	539	537.0	**537**

TABLE II. Beasley's test set E

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
e01	2500	3125	5	678	1282	5	1	43	251	63	244	2253	0.9	0.4	0.8	1.9	4.3	111	110.2	**111**
e02	2500	3125	10	707	1315	10	1	71	590	70	562	6077	0.9	1.1	3.1	4.2	9.7	214	213.1	**214**
e03	2500	3125	417	494	776	244	1	13	2539	44	2195	28,965	2.7	93.6	4.6	11.1	114.0	4052	4012.3	**4013**
e04	2500	3125	625	342	517	210	1	9	1260	0	1061	7521	3.0	43.6	1.0	2.4	51.2	5114	5101.0	**5101**
e05	2500	3125	1250	109	153	90	1	13	635	0	457	3599	2.6	1.8	0.4	0.7	5.9	8130	8128.0	**8128**
e06	2500	5000	5	1845	4315	5	1	79	541	53	482	6031	3.9	3.2	5.7	12.4	25.5	73	72.3	**73**
e07	2500	5000	10	1889	4364	10	1	136	1341	257	991	17,361	4.1	7.7	33.7	37.0	83.4	149	144.1	**145**
e08	2500	5000	417	1651	3269	379	1	25	9512	0	8336	171,688	28.1	695.6	51.2	288.8	1070.2	2686	2640.0	**2640**
e09	2500	5000	625	1360	2454	472	1	23	7802	0	6996	125,746	31.8	992.3	33.6	181.6	1245.5	3656	3604.0	**3604**
e10	2500	5000	1250	627	957	402	1	13	3879	0	2941	64,688	27.5	372.9	7.9	37.1	448.8	5614	5600.0	**5600**
e11	2500	12,500	5	2498	11,868	5	1	130	691	52	612	16,101	81.4	14.8	24.0	78.3	199.5	34	33.1	**34**
e12	2500	12,500	10	2498	11,393	10	1	132	1721	254	1319	50,051	110.0	22.4	79.3	180.3	393.2	68	66.1	**67**
e13	2500	12,500	417	2113	4831	396	1	41	16,314	0	12,577	454,207	124.2	1016.9	398.9	1267.0	2816.3	1312	1280.0	**1280**
e14	2500	12,500	625	1803	3696	511	1	20	9293	0	8634	208,710	118.0	1120.4	64.4	300.6	1610.9	1752	1732.0	**1732**
e15	2500	12,500	1250	811	1290	503	1	24	7389	100	4069	122,736	95.2	885.8	80.1	226.7	1294.0	2792	2783.7	**2784**
e16	2500	62,500	5	2500	25,184	5	1	267	951	14	944	41,724	1034.5	66.6	99.9	388.6	1591.7	15	14.2	**15**
e17	2500	62,500	10	2500	21,508	10	1	176	1889	70	1864	83,628	879.5	57.9	110.4	459.1	1508.7	26	24.3	**25**
e18[a]	2500	62,500	417	2224	5996	394	19	306	66,658	760	10,650	375,453	3508.5	12,443.7	19,116.3	33,840.8	68,949.1	608	563.9	**564**
e19	2500	62,500	625	2207	5584	580	1	30	13,026	110	11,175	284,804	499.4	1660.5	318.3	2092.1	4581.5	788	758.0	**758**
e20	2500	62,500	1250	1257	2330	671	1	16	8667	0	7907	262,341	424.4	1595.3	27.7	260.5	2316.2	1349	1342.0	**1342**

[a] This run was performed with the default parameter setting except that a complete reduction test was used instead of Algorithm 3.2 and no time limit was given.

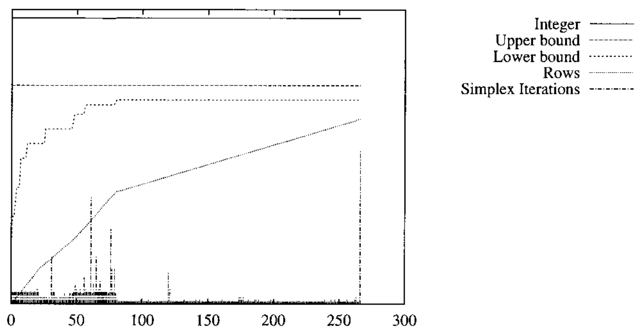

Fig. 7. *e16.*

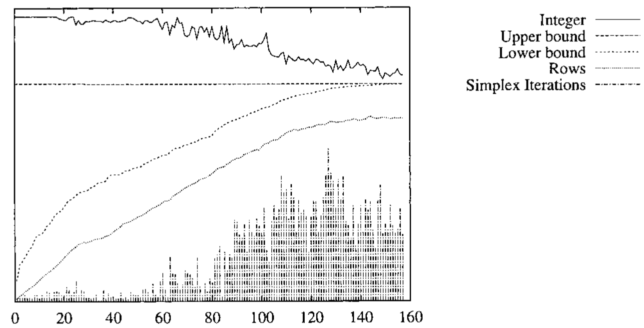

Fig. 9. *gap3100.*

examples (see also Figs. 8–10). The bottom bars depict the number of simplex iterations to solve the LPs; the higher the bars, the more difficult were the LPs to solve. We see that the number of simplex iterations is high if there is an increase in the lower bound, and the numbers are low when there is no progress in the lower bound. We have observed this behavior on many Beasley instances.

Table III contains some instances made us available by Margot [33] and some problems on complete graphs. *br* was introduced in Ferreira [16], whereas *berlin* and *gr* are taken from the TSP library, where some nodes are defined as terminals. It turns out that the winning procedure for complete instances is presolve. Algorithm 3.2 reduces up to 98% of the edges (variables) and provides the bases for solving even the big *gr* example with over 200,000 variables within 6 minutes.

The diagram of example *gr* in Figure 8 shows some different sign patterns from the one of *e16*. We observe that the number of fractionals (see the curve reflecting the number of *Integers*) is low at the beginning, increases continuously until the middle of the run, and decreases again toward the end. This u-shape behavior is typical for complete instances. We also see that the difficulty of the LPs is correlated to the number of fractional variables, which is also true for the examples depicted in Figures 9 and 10.

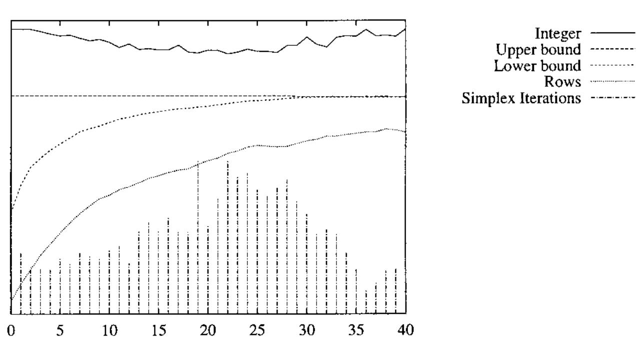

Fig. 8. *gr.*

Table IV contains a collection of examples described in Chopra et al. [8]. E. Gorres made these instances available to us. We solved all these instances within seconds.[§] Interesting to note is that almost always the root LP is integer (see Column *Frac*).

The next series, denoted by R, is taken from Soukup and Chow [37] (see Table V). We solve all of them in about 1 minute. Worthwhile to note are that in 24 of 38 examples the first call to the heuristic already found the optimal solution and that the LP time dominates all other times. The latter fact seems to be typical for grid examples, which the test set R consists of entirely. This phenomenon will become clearer in some of the next tests.

Tables VI and VII show results for the so-called *incidence* problems obtained from C. Duin. These problems, described in Duin [12] and Duin and Voß [15], are randomly generated and have the following sizes: There are four sizes of the node set $v := |V| = 80$, 160, 320, and 640; for each of them, 20 variants are generated combining four sizes of the terminal set $|T| = \log v$, $\sqrt{v}$, $2\sqrt{v}$, and $v/4$ with five different densities $|E| = (3v)/2$, $v \ln v$, $[v(v-1)]/2$, $2v$, and $[v(v-1)]/10$, all values are rounded down to the next integer. Every variant was drawn five times. The problem names have the pattern *v.tei*, where $v = 80$, 160, 320, and 640 gives the number of nodes of the problem, $t = 0, 1, 2, 3$ indicates which of the four alternatives (in the above sequence) of the sizes for the terminal sets have been chosen, $e = 0, 1, 2, 3, 4$ stands for the five densities, and $i = 1, 2, 3, 4, 5$ distinguishes the five instances drawn for each variant. To give an example, problem *160.141* is the first out of five instances with 160 nodes, $[v(v-1)]/10 = (160 \cdot 159)/10 = 2544$ edges, and $\lfloor\sqrt{v}\rfloor = \lfloor\sqrt{160}\rfloor = 12$ terminals. For each variant, our algorithm behaves very similarly for the five instances; thus, we show only the first in Tables VI and VII. The computations for these prob-

[§] The optimal values sometimes differ from the one described in Chopra et al. [8], because they did not add the values of variables fixed by presolve.

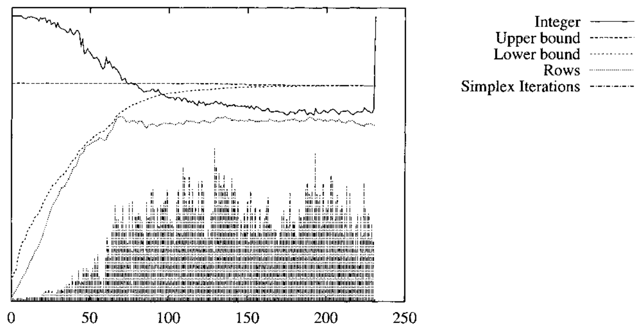

Fig. 10. *es40o.*

lems were performed on a Sun Enterprise 3000 at a later date, using CPLEX Version 5.0 (instead of Version 4.0.9 as for the other computations).

The *incidence* problems show completely different solution characteristics. One main difference is that our presolve algorithm (neither the default nor complete test) could not find any significant reductions (note that the problems were generated with the intention to have difficult problems for presolve, see Duin [12] and Duin and Voß [15]). On the other hand, the examples do not show the same sign of degeneracy and we could fix many variables in the course of the algorithm by the reduced cost criterium (see Section 4.5). To tackle the incident problems, it turned out to be a good idea to restart our branch-and-cut algorithm from scratch after a certain percentage of the variables could be fixed. Column *R* in Tables VI and VII shows the number of restarts performed and Columns 5–7 give, instead of the sizes of the presolved problems (which are almost always identical to the original sizes), the sizes of the problems after the last restart. We see that with this idea of iterative restarts sometimes a significant amount of variables can be fixed, especially if the number of terminals is small (for instance, the number of edges of *640.021* can be reduced by 97%). We are able to solve all problems on 80 and 160 nodes. However, we have difficulties to solve some of the larger instances. There are problems like *320.311* or *640.141* that we even cannot solve within 1 day. Table VII presents the results, where we used a unique time limit of 10,000 seconds for the difficult instances. Within this time limit, we can give a solution guarantee of at least 4.2% for all *incidence* problems.

What one would like to have at this point is a comparison with other codes. However, this is very difficult. People have different machines with different storage spaces, use different packages for the solution of subproblems like linear programs, and so on. We refrain from giving a comparison here. The interested reader may refer to Beasley [5], Chopra et al. [8], Duin [12], or Lucena [31], who developed comparable codes for the Steiner tree problem in graphs.

TABLE III. Examples of Francois Margot and complete instances

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
mc11	400	760	213	193	280	101	1	38	832	0	588	2957	0.2	6.7	1.2	2.0	10.9	11,722	11,689.0	**11,689**
mc13	150	11,175	80	149	669	80	57	232	5501	361	989	16,538	1.4	513.3	143.8	77.4	739.5	95	91.1	**92**
mc2	120	7140	60	120	489	60	11	60	1806	283	792	14,482	0.6	97.5	27.7	10.5	137.2	76	71.0	**71**
mc3	97	4656	45	97	1204	45	11	72	2518	575	2193	84,180	0.4	450.0	209.0	13.2	674.1	48	46.1	**47**
mc7	400	760	170	277	455	123	1	21	1065	0	883	6054	0.2	8.7	1.8	3.2	14.6	3486	3417.0	**3417**
mc8	400	760	188	248	364	129	1	33	1262	79	780	4190	0.2	14.8	2.7	4.0	22.6	1570	1565.5	**1566**
berlin	52	1326	16	48	147	15	1	13	159	0	152	1481	0.1	0.0	0.1	0.1	0.4	1048	1044.0	**1044**
br	58	1653	25	39	113	10	1	15	97	0	88	902	0.1	0.0	0.1	0.1	0.4	13,666	13,655.0	**13,655**
gr	666	221,445	174	599	3114	137	1	41	3143	0	1930	58,769	161.9	32.1	86.3	46.4	329.9	123,076	122,467.0	**122,467**

TABLE IV. Test set of E. Gorres

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
p401	100	4950	5	83	183	5	1	26	113	0	106	886	0.3	0.2	0.1	0.2	0.9	155	155.0	**155**
p402	100	4950	5	68	142	5	1	8	32	0	30	159	0.2	0.1	0.0	0.0	0.6	116	116.0	**116**
p403	100	4950	5	87	198	5	1	32	125	0	121	1333	0.3	0.2	0.2	0.2	1.0	181	179.0	**179**
p404	100	4950	10	64	123	9	1	15	71	0	68	437	0.2	0.1	0.1	0.0	0.6	270	270.0	**270**
p405	100	4950	10	65	124	9	1	9	77	0	76	510	0.2	0.1	0.0	0.1	0.6	270	270.0	**270**
p406	100	4950	10	83	172	10	1	13	111	0	105	914	0.3	0.1	0.1	0.1	0.7	290	290.0	**290**
p407	100	4950	20	81	159	18	1	13	201	0	198	1695	0.3	0.1	0.1	0.2	0.9	590	590.0	**590**
p408	100	4950	20	64	121	17	1	13	183	0	174	1326	0.2	0.1	0.1	0.1	0.8	543	542.0	**542**
p409	100	4950	50	20	26	15	1	7	68	0	67	272	0.2	0.1	0.0	0.0	0.5	964	963.0	**963**
p455	100	4950	5	100	1057	5	1	56	337	0	181	9437	0.4	0.4	2.4	2.5	6.0	1166	1138.0	**1138**
p456	100	4950	5	100	880	5	1	81	358	0	257	11,680	0.4	0.4	3.2	2.9	7.2	1228	1228.0	**1228**
p457	100	4950	10	99	654	10	1	50	450	0	313	11,135	0.3	0.3	2.7	1.7	5.3	1639	1609.0	**1609**
p458	100	4950	10	100	594	10	1	31	422	0	264	7937	0.3	0.2	3.2	1.0	4.9	1868	1868.0	**1868**
p459	100	4950	20	98	415	20	1	23	261	0	239	3851	0.3	0.2	0.4	0.6	1.7	2345	2345.0	**2345**
p460	100	4950	20	97	448	20	1	26	468	0	352	8568	0.3	0.2	2.7	0.8	4.3	2976	2959.0	**2959**
p461	100	4950	50	68	136	32	1	16	205	0	187	1098	0.2	0.1	0.2	0.2	0.9	4482	4474.0	**4474**
p463	200	19,900	10	200	2213	10	1	72	977	0	444	30,574	4.6	1.6	26.5	10.9	44.1	1519	1510.0	**1510**
p464	200	19,900	20	195	1760	18	1	58	1228	0	493	30,584	4.1	1.3	46.6	9.1	61.5	2553	2545.0	**2545**
p465	200	19,900	40	191	811	39	1	37	1002	0	815	16,770	3.1	1.2	6.1	4.6	15.6	3862	3853.0	**3853**
p466	200	19,900	100	143	302	68	1	20	536	0	416	3117	2.8	1.5	0.8	0.8	6.4	6252	6234.0	**6234**
p601	100	180	5	77	134	5	1	22	128	0	121	705	0.0	0.0	0.1	0.2	0.4	10,230	10,230.0	**10,230**
p602	100	180	5	77	133	5	1	21	121	0	117	673	0.0	0.0	0.1	0.1	0.3	8083	8083.0	**8083**
p603	100	180	5	78	136	5	1	16	75	0	71	394	0.0	0.0	0.1	0.1	0.2	5022	5022.0	**5022**
p604	100	180	10	72	124	8	1	20	152	0	145	905	0.0	0.0	0.1	0.2	0.4	11,397	11,397.0	**11,397**
p605	100	180	10	75	128	9	1	16	99	0	82	407	0.0	0.0	0.1	0.1	0.3	10,355	10,355.0	**10,355**
p606	100	180	10	80	135	9	1	18	139	0	125	710	0.0	0.0	0.1	0.1	0.3	13,048	13,048.0	**13,048**
p607	100	180	20	56	91	16	1	18	178	0	151	838	0.0	0.1	0.1	0.1	0.4	15,358	15,358.0	**15,358**
p608	100	180	20	54	87	15	1	13	131	0	124	647	0.0	0.0	0.1	0.1	0.3	14,439	14,439.0	**14,439**
p609	100	180	20	69	114	16	1	21	216	0	184	1120	0.0	0.1	0.2	0.1	0.5	18,462	18,263.0	**18,263**
p610	100	180	50	38	58	18	1	12	110	0	89	376	0.0	0.0	0.0	0.0	0.2	30,161	30,161.9	**30,161**
p611	100	180	50	30	42	17	1	9	78	0	74	296	0.0	0.0	0.0	0.0	0.1	26,903	26,903.0	**26,903**
p612	100	180	50	37	54	18	1	13	135	0	117	512	0.0	0.0	0.1	0.0	0.3	30,258	30,258.0	**30,258**
p613	200	370	10	169	292	10	1	38	408	0	266	1823	0.0	0.1	1.2	0.7	2.2	18,429	18,429.0	**18,429**
p614	200	370	20	181	309	19	1	34	490	0	366	2338	0.0	0.2	0.9	0.9	2.2	27,527	27,276.0	**27,276**
p615	200	370	40	154	252	39	1	22	542	0	434	2761	0.0	0.3	0.8	0.8	2.2	42,879	42,474.0	**42,474**
p616	200	370	100	61	88	35	1	17	202	0	175	743	0.1	0.1	0.1	0.1	0.6	62,263	62,263.0	**62,263**
p619	100	180	5	86	160	5	1	21	122	0	109	672	0.0	0.0	0.1	0.1	0.4	7485	7485.0	**7485**
p620	100	180	5	86	160	5	1	41	179	0	167	1115	0.0	0.1	0.2	0.3	0.7	8746	8746.0	**8746**

Table IV continues

TABLE IV. Continued

Name	Original			Presolved			B & C			Root LP			Time					Solutions		
	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
p621	100	180	5	86	160	5	1	35	188	0	151	1014	0.0	0.0	0.3	0.2	0.7	8741	8688.0	**8688**
p622	100	180	10	87	159	10	1	25	224	0	193	1314	0.0	0.0	0.3	0.2	0.7	16,546	15,972.0	**15,972**
p623	100	180	10	86	156	10	1	27	277	0	234	1523	0.0	0.0	0.5	0.2	0.9	19,496	19,496.0	**19,496**
p624	100	180	20	81	142	14	1	16	187	0	172	970	0.0	0.0	0.1	0.1	0.4	20,246	20,246.0	**20,246**
p625	100	180	20	84	151	20	1	21	317	0	282	1891	0.0	0.1	0.5	0.3	1.0	23,677	23,078.0	**23,078**
p626	100	180	20	81	143	20	1	17	252	0	224	1389	0.0	0.1	0.2	0.2	0.5	22,346	22,346.0	**22,346**
p627	100	180	50	47	73	24	1	12	136	0	102	447	0.0	0.0	0.1	0.1	0.3	40,647	40,647.0	**40,647**
p628	100	180	50	57	94	29	1	15	180	0	160	869	0.0	0.0	0.1	0.1	0.4	40,008	40,008.0	**40,008**
p629	100	180	50	53	85	25	1	13	141	29	137	647	0.0	0.0	0.1	0.1	0.3	43,287	43,286.5	**43,287**
p630	200	370	10	189	355	10	1	40	498	0	329	2581	0.0	0.1	2.4	0.9	3.6	26,125	26,125.0	**26,125**
p631	200	370	20	185	342	20	1	36	565	0	464	3253	0.0	0.2	1.7	0.9	3.1	39,193	39,067.0	**39,067**
p632	200	370	40	177	322	34	1	25	692	0	567	4083	0.0	0.2	2.3	1.1	3.9	56,562	56,217.0	**56,217**
p633	200	370	100	99	160	51	1	16	296	0	253	1355	0.0	0.4	0.3	0.3	1.2	86,573	86,268.0	**86,268**

Tables VIII–X give computational results on real-world VLSI instances. One of the challenging problems in the design of electronic circuits is the routing problem, which is, roughly speaking, the task to connect terminal sets via wires on a predefined area. Depending on the underlying technology and the design rules, subproblems arise that can be formulated as the problem of packing Steiner trees in certain graphs (see Lengauer [30] for an excellent treatment of this subject). The problems that we are going to consider result from seven different circuits described in Jünger et al. [27]. The underlying graphs are grid graphs that contain holes. The holes result from so-called cells that block certain areas of the grid. The sets of terminals are located on the border of these holes. For each of the seven circuits and for each terminal set T_i (where index i runs from 1 to the number of terminal sets of the circuit), we constructed an instance of the Steiner tree problem. For the graph G, we have chosen the underlying grid graph restricted to the minimal enclosing rectangle of the terminal set. The distance of two neighbored grid points in horizontal and vertical directions differ for these circuits. This results in different edge costs for horizontal and vertical edges in G.

In the library *SteinLib*, we put all instances with terminal sets whose cardinality is at least 10 (in total 475). The examples are distinguished by the name of the circuit followed by the index of the terminal set. For example, *msm1234* means that the instance is defined by terminal set 1234 of circuit *msm*. As test problems for our algorithm, we chose for each circuit all instances whose two leading nonzeros of the index of the terminal set differ from the two leading nonzeros of all other indices. If there are more than one index with the same two leading nonzeros, we chose the instance with the smallest index (for instance, among examples *msm3727*, *msm3731*, *msm3761*, and *msm3786*, we chose *msm3727*). In addition, we added an instance with the smallest and largest number of terminals for each circuit. This way we obtained 116 different VLSI test instances.

The success of our branch-and-cut algorithm is shown in Tables VIII–X. We solve 83 out of the 116 instances to optimality within 10,000 seconds and provide a solution guarantee [(upper bound − lower bound)/lower bound] of less than 10% for 85% of the examples. The biggest with respect to number of terminals that we solve within the time limit are *alue5067* and *alue6735* with 68 terminals each. The biggest in size of the number of edges is *msm3727* with over 8000 edges. However, there are also smaller instances, for example, *diw0795* with 10 terminals or *msm2601* with less than 5000 variables after presolve, that we do not solve within the time limit. All runs were performed with the default strategy (except for *diw0234* and *alut2625*); in particular, we applied Algorithm 3.2 to reduce the problem and did not perform a complete reduction test (see Section 3). If there is no time

TABLE V. Test set of Soukup and Chow

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
r01	15	22	5	11	17	5	1	7	20	0	19	71	0.0	0.0	0.0	0.0	0.0	187	187.0	**187**
r02	12	17	6	4	5	3	1	2	6	0	6	18	0.0	0.0	0.0	0.0	0.0	164	164.0	**164**
r03	28	45	7	19	32	7	1	12	60	0	50	240	0.0	0.0	0.0	0.0	0.1	237	236.0	**236**
r04	64	112	8	49	90	8	1	23	209	0	145	1012	0.0	0.0	0.3	0.1	0.6	258	254.0	**254**
r07	30	49	12	19	29	12	1	10	54	0	48	185	0.0	0.0	0.0	0.0	0.1	254	248.0	**248**
r08	24	37	12	16	22	12	1	5	37	0	35	106	0.0	0.0	0.0	0.0	0.1	236	236.0	**236**
r09	15	22	7	10	14	7	1	3	17	0	17	57	0.0	0.0	0.0	0.0	0.0	164	164.0	**164**
r10	36	60	6	24	43	6	1	12	58	0	56	296	0.0	0.0	0.1	0.0	0.1	177	177.0	**177**
r11	30	49	6	21	35	6	1	7	33	0	33	146	0.0	0.0	0.0	0.0	0.1	144	144.0	**144**
r12	27	42	9	22	36	9	1	13	73	0	55	273	0.0	0.0	0.1	0.0	0.1	180	180.0	**180**
r13	42	71	9	32	54	9	1	13	105	0	92	465	0.0	0.0	0.1	0.1	0.2	150	150.0	**150**
r14	36	60	12	10	14	6	1	2	14	0	14	42	0.0	0.0	0.0	0.0	0.0	260	260.0	**260**
r15	100	180	14	89	165	14	1	20	273	24	234	1567	0.0	0.1	0.4	0.2	0.8	150	147.3	**148**
r17	48	82	10	34	60	10	1	17	106	0	101	546	0.0	0.0	0.1	0.1	0.2	200	200.0	**200**
r18	182	337	62	156	264	52	1	21	477	0	424	2172	0.0	0.6	0.5	0.5	2.1	406	404.0	**404**
r19	168	310	14	162	303	14	1	30	491	152	331	2606	0.1	0.1	1.9	0.8	3.0	190	187.3	**188**
r21	15	22	5	11	16	5	1	4	20	0	20	74	0.0	0.0	0.0	0.0	0.0	192	192.0	**192**
r22	16	24	4	7	10	3	1	4	8	0	8	27	0.0	0.0	0.0	0.0	0.0	63	63.0	**63**
r23	16	24	4	7	10	3	1	3	7	0	7	23	0.0	0.0	0.0	0.0	0.1	65	65.0	**65**
r24	16	24	4	8	12	4	1	5	12	0	12	43	0.0	0.0	0.0	0.0	0.0	30	30.0	**30**
r27	16	24	4	6	9	3	1	4	8	0	8	26	0.0	0.0	0.0	0.0	0.1	133	133.0	**133**
r28	12	17	4	9	14	4	1	5	13	0	13	48	0.0	0.0	0.0	0.0	0.0	24	24.0	**24**
r29	9	12	3	6	8	3	1	4	10	0	10	37	0.0	0.0	0.0	0.0	0.1	200	200.0	**200**
r30	28	45	12	19	30	12	1	12	69	0	57	262	0.0	0.0	0.0	0.0	0.1	110	110.0	**110**
r31	130	237	14	116	216	14	1	25	415	111	323	2375	0.0	0.1	1.3	0.4	1.9	267	258.4	**259**
r32	210	391	19	192	364	19	1	41	1069	294	541	5303	0.0	0.3	15.3	1.8	17.5	315	312.1	**313**
r33	132	241	18	132	241	18	1	30	474	172	351	2414	0.0	0.1	2.0	0.6	2.9	268	267.1	**268**
r34	272	511	19	259	492	19	1	62	1914	0	699	8484	0.1	0.6	64.7	4.2	69.8	257	241.0	**241**
r35	240	449	18	228	432	18	1	50	1127	289	524	5047	0.0	0.4	16.8	2.2	19.7	159	150.9	**151**
r37	49	84	8	34	61	8	1	15	100	0	95	532	0.0	0.0	0.1	0.0	0.2	90	90.0	**90**
r38	100	180	14	75	140	12	1	18	250	36	220	1388	0.0	0.0	0.3	0.1	0.6	166	165.5	**166**
r39	100	180	14	70	130	12	1	18	214	0	189	1157	0.0	0.0	0.2	0.2	0.5	166	166.0	**166**
r40	64	112	10	54	97	10	1	22	218	39	166	1017	0.0	0.0	0.3	0.1	0.5	155	154.2	**155**
r41	144	263	20	135	249	20	1	24	474	103	372	2514	0.0	0.1	1.3	0.5	2.1	224	223.2	**224**
r42	81	144	15	69	127	15	1	17	221	42	188	1255	0.0	0.0	0.3	0.1	0.6	154	152.3	**153**
r43	195	362	16	179	335	16	1	46	1078	0	482	4622	0.1	0.2	12.4	1.5	14.4	258	255.0	**255**
r44	196	364	17	176	331	17	1	55	1164	274	467	4879	0.1	0.3	16.1	2.0	18.6	256	251.9	**252**
r45	270	507	19	256	485	19	1	70	1865	315	686	8216	0.1	0.6	58.0	4.3	63.4	223	219.7	**220**

TABLE VI. Test set 80 and 160 of C. Duin

	Original			Final size			B & C			Root LP			Time				Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	R	N	It	Frac	Rows	NZ	H	LP	Sep	Tot	Heu(1)	LB	UB
80.001	80	120	6	80	120	6	0	1	15	0	98	657	0.0	0.1	0.0	0.2	1787	1787.0	**1787**
80.011	80	350	6	77	195	6	1	1	56	26	197	3215	0.0	0.3	0.3	0.8	1482	1478.2	**1479**
80.021	80	3160	6	80	448	6	1	1	88	0	602	26,151	0.0	2.1	2.8	5.5	1175	1175.0	**1175**
80.031	80	160	6	80	160	6	0	1	18	0	100	822	0.0	0.0	0.1	0.2	1570	1570.0	**1570**
80.041	80	632	6	79	302	6	1	1	102	0	364	9827	0.0	1.3	1.0	2.5	1279	1276.0	**1276**
80.101	80	120	8	80	120	8	0	1	15	0	103	621	0.0	0.0	0.1	0.1	2608	2608.0	**2608**
80.111	80	350	8	80	350	8	0	3	63	0	399	9811	0.1	2.3	0.8	4.0	2051	2025.4	**2051**
80.121	80	3160	8	80	589	8	1	1	85	0	881	39,730	0.1	3.4	3.4	7.5	1561	1561.0	**1561**
80.131	80	160	8	80	160	8	0	1	36	0	202	1663	0.0	0.1	0.2	0.4	2284	2284.0	**2284**
80.141	80	632	8	75	216	8	2	1	107	0	242	4654	0.1	1.7	1.5	3.5	1847	1788.0	**1788**
80.201	80	120	16	80	120	16	0	1	22	0	178	1388	0.0	0.1	0.1	0.3	4862	4760.0	**4760**
80.211	80	350	16	80	350	16	0	1	33	0	498	13,993	0.0	1.5	0.6	2.3	3758	3631.0	**3631**
80.221	80	3160	16	80	1015	16	2	1	107	0	1489	70,586	0.2	8.8	12.1	22.1	3158	3158.0	**3158**
80.231	80	160	16	80	160	16	0	1	18	0	264	2636	0.0	0.1	0.2	0.4	4456	4354.0	**4354**
80.241	80	632	16	80	632	16	0	115	913	305	499	47,935	0.9	929.0	58.8	1097.1	3551	3487.1	**3538**
80.301	80	120	20	80	120	20	0	1	9	0	154	1058	0.0	0.0	0.1	0.2	5530	5519.0	**5519**
80.311	80	350	20	80	350	20	0	1	38	101	434	13,952	0.0	7.0	1.3	8.6	4708	4553.0	**4554**
80.321	80	3160	20	80	783	20	3	1	224	0	1037	37,245	0.5	10.5	17.3	29.8	3932	3932.0	**3932**
80.331	80	160	20	80	160	20	0	5	23	0	301	3601	0.0	0.8	0.2	1.7	5321	5204.5	**5226**
80.341	80	632	20	80	632	20	0	1	30	95	664	30,800	0.0	1.6	1.3	3.1	4316	4235.1	**4236**
160.001	160	240	7	91	113	7	1	1	38	0	124	726	0.0	0.2	0.2	0.6	2496	2490.0	**2490**
160.011	160	812	7	110	193	7	2	1	66	0	176	2552	0.0	0.5	0.8	1.6	1757	1677.0	**1677**
160.021	160	12,720	7	160	1049	7	1	1	164	0	1534	126,445	0.2	29.5	21.0	53.4	1352	1352.0	**1352**
160.031	160	320	7	97	122	7	1	1	34	0	92	545	0.0	0.2	0.2	0.5	2170	2170.0	**2170**
160.041	160	2544	7	160	1810	7	1	1	195	0	797	50,471	0.4	12.6	15.4	29.3	1542	1494.0	**1494**
160.101	160	240	12	105	135	12	1	1	32	0	156	955	0.0	0.2	0.4	0.8	3859	3859.0	**3859**
160.111	160	812	12	138	278	12	1	1	53	0	295	5996	0.0	0.9	1.2	2.5	3059	2869.0	**2869**
160.121	160	12,720	12	160	932	12	6	1	1217	0	1394	80,633	3.8	581.8	214.6	808.0	2363	2363.0	**2363**
160.131	160	320	12	115	174	12	1	1	43	0	160	1417	0.0	0.4	0.5	1.3	3356	3356.0	**3356**
160.141	160	2544	12	135	556	12	4	1	254	0	674	27,248	0.4	25.4	15.5	42.8	2549	2549.0	**2549**
160.201	160	240	24	160	240	24	0	5	24	0	352	3860	0.1	0.6	0.6	1.9	7117	6909.0	**6923**
160.211	160	812	24	160	812	24	0	15	470	344	786	45,208	1.2	1032.8	74.6	1143.5	5735	5540.3	**5583**
160.221	160	12,720	24	160	2380	24	5	1	798	118	4698	499,305	6.2	547.5	325.2	889.1	4729	4728.6	**4729**
160.231	160	320	24	160	320	24	0	1	24	0	497	6911	0.1	1.3	0.7	2.3	6810	6662.0	**6662**
160.241	160	2544	24	160	2544	24	0	121	1802	731	1469	332,829	8.7	19,156.6	1162.0	21,008.6	5186	5034.7	**5086**
160.301	160	240	40	160	240	40	0	1	11	0	425	3676	0.0	0.3	0.3	1.1	11,909	11,816.0	**11,816**
160.311	160	812	40	160	812	40	0	25	356	346	1055	46,698	1.6	560.0	82.4	687.9	9510	9083.9	**9135**
160.321	160	12,720	40	160	12,720	40	0	9	160	1125	3806	1,297,794	5.1	1613.9	1179.9	2959.4	7903	7871.4	**7876**
160.331	160	320	40	160	320	40	0	1	16	0	666	9274	0.1	0.4	0.8	1.8	10,820	10,414.0	**10,414**
160.341	160	2544	40	160	2544	40	0	131	2837	996	1485	346,171	23.2	50,712.0	2750.5	54,500.7	8558	8277.9	**8331**

TABLE VII. Test set 320 and 640 of C. Duin

	Original			Final size			B & C			Root LP			Time				Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	R	N	It	Frac	Rows	NZ	H	LP	Sep	Tot	Heu(1)	LB	UB
320.001	320	480	8	144	174	8	1	1	51	0	153	913	0.1	0.3	0.7	1.3	2672	2672.0	**2672**
320.011	320	1845	8	88	177	8	4	3	272	0	254	4184	0.3	13.9	10.0	25.5	2057	2049.5	**2053**
320.021	320	51,040	8	320	2227	8	3	1	453	0	3578	581,443	4.4	409.1	659.1	1102.0	1564	1553.0	**1553**
320.031	320	640	8	320	640	8	0	3	92	0	582	9176	0.1	5.8	3.1	10.5	2709	2660.0	**2673**
320.041	320	10,208	8	78	175	8	8	1	895	0	259	4610	4.4	182.5	239.2	434.6	1750	1707.0	**1707**
320.101	320	480	17	160	190	17	1	1	36	0	190	1037	0.0	0.3	0.9	1.7	5548	5548.0	**5548**
320.111	320	1845	17	320	1845	17	0	7	168	116	1175	41,141	0.7	149.8	42.1	206.5	4435	4253.4	**4273**
320.121	320	51,040	17	320	2046	17	8	1	1115	0	2619	218,395	17.7	621.6	1909.0	2595.3	3321	3321.0	**3321**
320.131	320	640	17	320	640	17	0	1	45	0	770	11,300	0.1	3.6	3.5	7.4	5353	5255.0	**5255**
320.141	320	10,208	17	320	10,208	17	0	45	834	183	2993	632,380	11.2	8188.6	1476.2	10,139.9	3701	3567.9	3606
320.201	320	480	34	320	480	34	0	1	33	0	708	7595	0.2	1.3	2.9	5.0	10,275	10,044.0	**10,044**
320.211	320	1845	34	320	1845	34	0	49	1254	517	1829	117,293	9.9	7121.4	922.4	8274.6	8476	7987.8	**8039**
320.221	320	51,040	34	320	15,726	34	2	1	624	267	10,214	3,936,831	69.0	3846.7	6040.0	10,019.4	6697	6664.0	6679
320.231	320	640	34	320	640	34	0	7	52	72	932	23,055	0.3	22.4	8.3	39.5	10,028	9855.8	**9862**
320.241	320	10,208	34	320	10,208	34	0	15	344	1003	4756	865,964	8.3	8389.4	1387.4	10,007.7	7214	6985.4	7027
320.301	320	480	80	320	480	80	0	1	19	0	1067	19,037	0.3	2.1	3.9	13.3	23,669	23,279.0	**23,279**
320.311	320	1845	80	320	1845	80	0	97	1101	503	1778	83,568	26.7	7754.0	1788.6	10,006.6	18,843	17,744.9	17,945
320.321	320	51,040	80	320	51,040	80	0	1	84	1182	7994	4,372,292	39.5	5323.6	4972.2	10,368.9	15,863	15,609.6	15,771
320.331	320	640	80	320	640	80	0	33	124	0	1206	24,025	2.0	159.1	45.9	266.7	22,351	21,460.7	**21,517**
320.341	320	10,208	80	320	10,208	80	0	3	106	1563	6219	1,620,175	7.3	8785.2	1096.9	10,021.5	16,463	16,150.3	16,374
640.001	640	960	9	33	54	9	1	1	51	0	101	571	0.0	0.5	1.0	5.2	4183	4033.0	**4033**
640.011	640	4135	9	45	91	9	3	1	121	0	119	873	0.2	2.3	6.3	18.8	2392	2392.0	**2392**
640.021	640	204,480	9	640	5215	9	3	1	818	0	8234	2,633,466	23.5	4358.5	6369.4	11,004.9	1749	1749.0	**1749**
640.031	640	1280	9	209	254	9	2	1	96	0	207	1330	0.2	1.8	3.0	6.5	3278	3278.0	**3278**
640.041	640	40,896	9	111	340	9	7	1	806	48	437	10,565	21.1	422.2	1732.3	2214.9	1945	1896.3	**1897**
640.101	640	960	25	384	515	25	1	1	67	0	640	8418	0.4	15.0	8.2	26.7	9054	8764.0	**8764**
640.111	640	4135	25	640	4135	25	0	83	1935	371	3204	133,572	24.0	11,537.8	1655.6	13,677.5	6415	6079.4	**6167**
640.121	640	204,480	25	640	204,480	25	0	1	59	36	2360	2,122,208	45.8	209.5	9710.8	10,158.6	4906	4741.2	4906
640.131	640	1280	25	387	517	25	1	1	76	0	576	7774	0.4	13.6	11.5	28.6	8319	8097.0	**8097**
640.141	640	40,896	25	640	40,896	25	0	3	230	209	6401	1,536,076	29.7	6378.3	3432.3	10,059.2	5307	5141.9	5247
640.201	640	960	50	640	960	50	0	3	39	0	1516	25,610	0.7	11.7	10.1	31.3	16,797	16,078.0	**16,079**
640.211	640	4135	50	640	4135	50	0	1	245	1149	3746	297,821	7.3	9297.2	690.2	10,002.7	12,714	11,793.8	12,291
640.221	640	204,480	50	640	204,480	50	0	1	50	54	3517	2,947,251	88.0	272.5	9656.3	10,219.6	9876	9542.4	9876
640.231	640	1280	50	640	1280	50	0	23	185	416	1707	43,770	2.7	627.7	103.6	804.1	15,201	14,974.8	**15,014**
640.241	640	40,896	50	640	40,896	50	0	1	139	1008	10,488	2,692,998	33.4	7544.2	2500.2	10,102.6	10,386	10,136.0	10,338
640.301	640	960	160	640	960	160	0	1	22	0	2061	54,098	4.7	15.7	20.3	152.3	46,104	45,005.0	**45,005**
640.311	640	4135	160	640	4135	160	0	3	184	1741	4644	193,418	40.5	8394.5	1417.8	10,039.2	37,706	35,256.9	36,562
640.321	640	204,480	160	640	204,480	160	0	1	30	0	6108	4,777,100	170.1	388.1	9174.5	10,058.1	31,757	30,594.0	31,757
640.331	640	1280	160	640	1280	160	0	55	507	678	2209	118,490	79.2	2227.3	1077.9	3817.9	44,711	42,754.7	**42,796**
640.341	640	40,896	160	640	40,896	160	0	1	63	2354	17,086	7,176,821	59.0	7469.7	2697.4	10,332.2	32,303	31,840.6	32,303

TABLE VIII. VLSI examples: *diw* and *taq*

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
diw0234[a]	5349	10,086	25	3856	7266	24	1	396	10,928	725	5113	54,492	12,657.3	66.9	10,554.2	717.3	24,002.9	2001	1995.1	**1996**
diw0250	353	608	11	308	545	11	1	57	584	0	505	3387	0.1	0.3	2.7	1.8	5.1	350	350.0	**350**
diw0260	539	985	12	518	954	11	1	67	628	0	606	3891	0.1	0.7	2.1	1.8	5.1	468	468.0	**468**
diw0313	468	822	14	421	752	12	1	50	619	71	500	3430	0.1	0.4	2.9	2.5	6.1	397	396.7	**397**
diw0393	212	381	11	194	353	11	1	40	460	0	385	2782	0.0	0.1	1.6	0.9	3.0	302	302.0	**302**
diw0445	1804	3311	33	1745	3240	33	1	239	6591	1060	2926	29,613	0.7	23.7	2664.8	136.7	2827.6	1387	1362.3	**1363**
diw0459	3636	6789	25	3516	6635	25	1	590	11,569	1318	5022	49,990	2.7	102.0	8014.3	506.1	8628.6	1367	1361.1	**1362**
diw0460	339	579	13	296	523	13	1	29	433	0	396	2440	0.1	0.1	0.8	0.9	2.2	358	345.0	**345**
diw0473	2213	4135	25	2140	4046	25	1	362	5882	701	3601	35,557	1.1	34.4	1658.3	172.9	1868.6	1107	1097.1	**1098**
diw0487	2414	4386	25	2294	4233	25	1	467	5295	0	3117	24,008	1.2	46.3	568.8	166.1	784.7	1451	1424.0	**1424**
diw0495	938	1655	10	894	1603	10	1	194	1534	109	1162	8067	0.2	3.3	22.7	13.7	40.7	626	615.5	**616**
diw0513	918	1684	10	867	1621	10	1	157	2147	467	1459	12,461	0.2	2.7	106.1	19.8	129.4	614	603.3	**604**
diw0523	1080	2015	10	1025	1943	10	1	267	2134	63	1663	14,383	0.3	5.3	156.5	24.9	188.1	561	560.7	**561**
diw0540	286	465	10	232	394	10	1	37	389	0	334	1953	0.0	0.1	0.9	1.0	2.2	374	374.0	**374**
diw0559	3738	7013	18	3627	6883	18	1	272	11,148	1501	4484	50,557	2.8	37.6	9589.4	437.6	10,070.0	1578	1373.9	1570
diw0778	7231	13,727	24	7145	13,629	24	1	200	12,428	1867	6585	69,632	10.1	79.5	9015.5	1026.9	10,135.7	2197	1800.3	2173
diw0779	11,821	22,516	50	11,715	22,399	50	1	141	15,341	2594	9499	82,366	28.3	255.0	7111.2	2767.7	10,167.2	4588	3158.3	4566
diw0795	3221	5938	10	3101	5792	10	1	298	9993	1762	4720	57,537	2.1	21.9	9715.5	333.0	10,074.8	1584	1455.7	1553
diw0801	3023	5575	10	2881	5400	10	1	269	9919	1815	4597	57,572	1.9	18.2	9689.6	324.8	10,036.6	1598	1528.3	1587
diw0819	10,553	20,066	32	10,447	19,942	32	1	140	12,691	2318	7923	76,641	22.5	133.0	8681.9	1426.5	10,267.9	3467	2458.8	3430
diw0820	11,749	22,384	37	11,634	22,253	37	1	156	15,756	2502	10,171	93,809	28.0	202.9	8169.4	1734.3	10,139.0	4271	2866.8	4259
taq0014	6466	11,046	128	6029	10,563	128	1	68	14,196	3620	8292	69,417	7.6	230.2	9578.2	714.4	10,536.2	5513	4688.3	5442
taq0023	572	963	11	501	873	11	1	94	1691	0	884	7946	0.1	0.9	60.2	7.5	69.2	623	621.0	**621**
taq0365	4186	7074	22	3830	6681	22	1	350	10,958	2036	4917	54,079	3.4	56.1	9296.8	663.6	10,023.3	1971	1819.2	1914
taq0377	6836	11,715	136	6433	11,301	136	1	56	13,936	4144	9095	74,502	8.7	256.1	9461.3	689.2	10,421.3	6659	5640.5	6565
taq0431	1128	1905	13	995	1745	13	1	139	3637	738	1564	15,975	0.3	3.0	516.1	37.9	558.1	937	896.1	**897**
taq0631	609	932	10	475	782	10	1	60	1114	311	624	4752	0.1	0.4	23.1	6.0	29.9	594	580.5	**581**
taq0739	837	1438	16	773	1362	16	3	91	2890	55	1318	13,277	0.2	30.9	308.4	20.6	360.8	859	847.1	**848**
taq0741	712	1217	16	636	1115	16	1	109	3186	733	1210	13,877	0.1	1.7	385.6	18.5	406.6	865	846.4	**847**
taq0751	1051	1791	16	945	1663	16	1	98	3467	793	1672	16,533	0.2	2.3	486.1	27.4	516.8	952	938.2	**939**
taq0891	331	560	10	269	476	10	1	82	624	180	473	3457	0.1	0.4	4.5	2.2	7.5	319	318.5	**319**
taq0903	6163	10,490	130	5652	9907	130	1	62	13,612	3593	7552	66,138	6.9	212.0	9655.8	604.7	10,484.6	5208	4510.3	5162
taq0910	310	514	17	254	437	15	1	29	429	0	376	2566	0.0	0.1	1.0	1.0	2.4	370	370.0	**370**
taq0920	122	194	17	64	105	13	1	12	99	0	92	451	0.0	0.0	0.0	0.1	0.2	210	210.0	**210**
taq0978	777	1239	10	670	1124	10	1	75	938	0	684	4709	0.1	1.0	9.3	8.6	19.4	566	566.0	**566**

[a] This run was performed with the default parameter setting except that the complete reduction test was used and no time limit was given.

TABLE IX. VLSI examples: *dmx* and *msm*

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
dmxa0296	233	386	12	192	332	12	1	40	425	55	349	2217	0.0	0.1	1.3	0.8	2.5	349	343.5	**344**
dmxa0368	2050	3676	18	1942	3558	18	1	274	4955	1016	2785	24,254	0.9	16.7	1120.5	134.3	1274.0	1021	1016.1	**1017**
dmxa0454	1848	3286	16	1747	3165	16	1	195	4168	780	2573	22,149	0.7	9.5	500.4	78.2	590.1	964	913.6	**914**
dmxa0628	169	280	10	144	249	10	1	37	400	132	319	2279	0.0	0.1	1.7	0.6	2.6	277	274.3	**275**
dmxa0734	663	1154	11	621	1103	11	1	111	1534	328	972	8073	0.1	1.4	80.8	11.0	93.8	519	505.4	**506**
dmxa0848	499	861	16	443	789	16	1	53	1341	0	872	7221	0.1	0.5	29.1	5.6	35.7	600	594.0	**594**
dmxa0903	632	1087	10	558	989	10	1	95	1917	411	975	9000	0.1	1.0	94.0	9.2	104.8	598	579.2	**580**
dmxa1010	3983	7108	23	3731	6817	23	1	812	10,320	870	4390	42,258	3.1	118.9	9107.0	785.5	10,019.1	1497	1407.2	1488
dmxa1109	343	559	17	296	502	17	1	37	728	0	493	3509	0.1	0.3	4.6	2.0	7.1	454	454.0	**454**
dmxa1200	770	1383	21	721	1322	21	1	69	2043	306	1331	11,829	0.2	1.5	117.4	11.6	131.4	762	749.6	**750**
dmxa1304	298	503	10	265	461	10	1	39	507	0	441	2912	0.0	0.2	1.8	1.5	3.8	312	311.0	**311**
dmxa1516	720	1269	11	667	1204	11	1	123	1376	291	894	6670	0.1	1.8	28.5	11.2	42.2	511	507.5	**508**
dmxa1721	1005	1731	18	923	1640	18	1	179	1699	138	1351	10,025	0.3	4.8	28.9	15.6	50.3	784	779.7	**780**
dmxa1801	2333	4137	17	2118	3890	17	1	191	8869	1548	3125	42,989	1.1	12.3	7063.0	180.5	7259.2	1423	1364.2	**1365**
msm0580	338	541	11	273	459	11	1	47	708	0	517	3636	0.0	0.2	5.6	1.6	7.8	480	467.0	**467**
msm0654	1290	2270	10	1163	2113	10	1	237	2931	591	1810	15,492	0.4	5.3	210.9	45.7	263.2	823	822.6	**823**
msm0709	1442	2403	16	1280	2211	16	1	97	2973	0	1608	13,760	0.5	3.3	206.2	35.6	246.4	884	884.0	**884**
msm0920	752	1264	26	643	1127	26	1	53	1739	0	1134	8535	0.1	1.5	34.1	10.0	46.3	821	806.0	**806**
msm1008	402	695	11	367	649	11	1	59	1093	304	655	5565	0.1	0.4	19.9	3.3	23.9	504	493.4	**494**
msm1234	933	1632	13	865	1548	11	1	176	1857	394	1081	8788	0.2	3.0	90.5	26.5	120.9	550	549.3	**550**
msm1477	1199	2078	31	1074	1915	31	1	86	2768	52	1631	14,427	0.3	5.0	234.9	20.9	262.0	1096	1067.5	**1068**
msm1707	278	478	11	238	420	11	1	47	339	0	323	1895	0.0	0.2	0.8	0.8	1.9	564	564.0	**564**
msm1844	90	135	10	60	95	10	1	15	143	0	133	623	0.0	0.0	0.1	0.1	0.2	191	188.0	**188**
msm1931	875	1522	10	795	1421	8	1	188	1720	367	1135	8707	0.2	2.5	56.8	15.7	75.7	604	603.8	**604**
msm2000	898	1562	10	814	1456	9	1	210	2024	444	1182	9980	0.2	2.9	90.3	19.0	113.2	594	593.5	**594**
msm2152	2132	3702	37	1996	3536	37	1	105	6697	1438	2871	31,772	1.0	14.1	2861.6	133.5	3011.7	1634	1589.1	**1590**

Table IX continues

TABLE IX. Continued

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
msm2326	418	723	14	383	682	14	1	35	677	0	529	3844	0.1	0.3	3.5	2.6	6.7	404	399.0	**399**
msm2492	4045	7094	12	3812	6815	12	1	681	11,530	1173	4836	45,321	3.2	63.9	9176.8	754.7	10,002.0	1460	1428.7	1459
msm2525	3031	5239	12	2780	4953	12	1	404	6285	1190	3759	32,556	2.0	28.3	1618.5	235.6	1886.5	1303	1289.1	**1290**
msm2601	2961	5100	16	2711	4829	16	1	284	9838	1590	3712	41,758	1.8	21.1	9704.3	311.3	10,043.5	1474	1403.6	1440
msm2705	1359	2458	13	1295	2381	13	1	114	2954	0	1682	15,480	0.4	3.5	328.8	37.3	370.8	730	714.0	**714**
msm2802	1709	2963	18	1567	2804	18	1	171	3662	0	2043	17,026	0.6	8.7	347.3	70.7	428.4	936	926.0	**926**
msm2846	3263	5783	89	3140	5637	89	1	111	12,358	2597	4935	55,724	2.2	91.9	9668.4	501.0	10,268.4	3208	3103.7	3141
msm3277	1704	2991	12	1564	2832	12	1	267	3781	350	2198	19,426	0.6	9.4	532.0	71.7	614.9	869	868.4	**869**
msm3676	957	1554	10	803	1367	10	1	125	1842	322	1039	8030	0.2	1.7	76.6	15.9	95.1	612	606.8	**607**
msm3727	4640	8255	21	4391	7988	21	1	458	10,565	0	4807	43,464	4.1	93.8	6646.7	917.7	7665.9	1406	1376.0	**1376**
msm3829	4221	7255	12	3895	6881	12	1	533	11,617	1444	4747	44,059	3.5	60.3	9374.0	629.2	10,070.4	1612	1384.6	1571
msm4038	237	390	11	181	313	11	1	59	515	112	366	2562	0.0	0.2	2.3	1.3	3.9	353	352.4	**353**
msm4114	402	690	16	352	625	16	1	43	684	29	510	3614	0.1	0.3	3.6	2.7	6.9	393	392.2	**393**
msm4190	391	666	16	340	602	16	1	39	763	209	586	4451	0.1	0.3	6.0	2.1	8.7	381	380.7	**381**
msm4224	191	302	11	156	258	11	1	38	417	134	313	2018	0.0	0.1	1.8	0.6	2.6	315	310.8	**311**
msm4312	5181	8893	10	4800	8464	10	1	371	10,370	1947	5678	58,738	5.0	45.4	9286.3	679.1	10,019.4	2055	1684.2	2049
msm4414	317	476	11	225	365	11	1	32	426	129	326	1909	0.0	0.1	1.3	0.7	2.2	408	407.6	**408**
msm4515	777	1358	13	734	1306	13	1	93	2175	453	1238	11,200	0.2	1.4	148.4	15.8	166.4	640	629.4	**630**

TABLE X. VLSI examples: *gap*, *alue*, and *alut*

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
gap1307	342	552	17	283	485	17	1	34	626	118	517	3401	0.0	0.2	2.5	1.7	4.8	554	548.1	**549**
gap1413	541	906	10	465	815	10	1	62	932	0	674	5006	0.1	0.4	11.3	4.4	16.6	457	457.0	**457**
gap1500	220	374	17	166	293	12	1	69	284	47	235	1546	0.0	0.2	0.7	0.7	1.9	254	253.2	**254**
gap1810	429	702	17	354	604	17	1	38	600	0	476	3003	0.1	0.3	2.2	2.1	4.9	490	482.0	**482**
gap1904	735	1256	21	673	1183	21	1	51	1385	131	1088	7755	0.1	1.2	15.4	8.1	25.4	778	762.6	**763**
gap2007	2039	3548	17	1894	3369	17	7	258	6645	60	2487	24,850	0.9	195.4	2048.2	163.6	2410.0	1120	1103.4	**1104**
gap2119	1724	2975	29	1557	2772	29	1	85	4427	0	2379	22,744	0.6	6.7	748.3	60.3	817.1	1269	1244.0	**1244**
gap2740	1196	2084	14	1080	1934	14	1	149	3547	496	1624	16,078	0.3	4.2	578.9	39.0	623.1	745	744.3	**745**
gap2800	386	653	12	328	577	12	1	65	824	0	603	4463	0.1	0.4	9.6	2.9	13.2	387	386.0	**386**
gap2975	179	293	10	156	267	10	1	40	316	0	277	1682	0.0	0.1	0.8	0.5	1.6	245	245.0	**245**
gap3036	346	583	13	308	535	13	1	43	912	288	567	4782	0.1	0.3	12.3	2.1	15.1	469	456.3	**457**
gap3100	921	1558	11	792	1393	11	1	158	2363	579	1301	12,083	0.2	2.3	156.3	19.3	178.9	642	639.2	**640**
gap3128	10,393	18,043	104	9711	17,308	104	1	120	11,689	2503	5900	54,435	20.3	377.6	6825.8	2790.2	10,021.1	4386	3616.3	4315
alue2087	1244	1971	34	962	1650	34	1	47	2047	0	1371	10,383	0.3	2.8	63.0	15.6	82.4	1065	1049.0	**1049**
alue2105	1220	1858	34	911	1510	34	1	118	1814	0	1290	10,091	0.3	5.7	38.4	23.5	68.7	1039	1032.0	**1032**
alue3146	3626	5869	64	3047	5223	64	1	231	8854	1131	4022	38,092	2.3	87.7	9403.7	587.3	10,085.0	2280	2215.0	2240
alue5067	3524	5560	68	2850	4819	68	1	101	7977	0	3553	34,789	2.1	45.4	3414.2	345.9	3809.9	2622	2586.0	**2586**
alue5345	5179	8165	68	4270	7202	68	1	126	10,940	2533	5199	49,767	4.4	89.0	9721.4	446.1	10,264.6	3603	3318.6	3560
alue5623	4472	6938	68	3589	6000	68	1	103	10,241	2405	4611	46,044	3.4	63.2	9811.4	335.2	10,216.0	3509	3258.1	3463
alue5901	11,543	18,429	68	9744	16,553	68	1	97	11,544	1836	6443	56,950	22.4	194.9	8376.2	1483.4	10,081.3	4042	3418.6	3994
alue6179	3372	5213	67	2701	4502	66	5	120	7620	61	3380	29,746	1.9	288.7	3989.4	289.8	4573.1	2483	2452.0	**2452**
alue6457	3932	6137	68	3233	5403	68	1	107	10,701	2161	4458	45,214	2.5	58.2	9643.1	390.6	10,097.3	3113	3005.3	3062
alue6735	4119	6696	68	3501	6021	68	1	73	7693	0	4028	37,632	3.0	42.4	3839.5	255.7	4142.9	2735	2696.0	**2696**
alue6951	2818	4419	67	2274	3843	67	1	95	5810	0	2780	26,221	1.4	31.3	1375.6	190.2	1600.4	2483	2386.0	**2386**
alue7065	34,046	54,841	544	29,243	49,948	544	1	22	12,456	1361	11,351	59,342	1026.3	6328.5	228.9	2422.4	10,042.9	24,827	12,589.4	24,827
alue7066	6405	10,454	16	5516	9506	15	1	390	10,907	1749	5621	52,684	6.9	74.2	8994.6	971.4	10,050.3	2285	1767.9	2275
alue7080[a]	34,479	55,494	2344	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
alue7229	940	1474	34	730	1234	34	1	104	1355	0	1115	8254	0.2	3.7	17.4	10.5	32.7	824	824.0	**824**
alut0787	1160	2089	34	1105	2023	34	1	107	2548	0	1759	14,332	0.3	6.5	148.9	27.7	184.4	987	982.0	**982**
alut0805	966	1666	34	852	1536	34	1	58	2634	512	1372	12,186	0.2	2.7	184.9	18.2	206.9	979	957.4	**958**
alut1181	3041	5693	64	2949	5571	64	1	99	10,540	2165	4722	53,720	1.9	43.9	9914.0	247.9	10,210.4	2462	2261.0	2390
alut2010	6104	11,011	68	5777	10,634	68	1	134	12,720	2717	6805	64,382	7.0	154.9	9208.0	695.5	10,069.7	3403	3218.0	3322
alut2288	9070	16,595	68	8824	16,329	68	1	98	13,097	2898	7958	73,484	15.7	186.2	8980.5	1013.4	10,200.3	3953	3291.7	3889
alut2566	5021	9055	68	4759	8746	67	1	104	11,235	2508	5543	54,087	4.7	83.5	9569.6	689.5	10,350.6	3129	2900.7	3127
alut2610	33,901	62,816	204	33,207	62,084	204	1	59	17,269	3115	14,250	106,987	1153.0	2260.8	2722.1	4316.0	10,470.0	12,797	6925.9	12,760
alut2625[b]	36,711	68,117	879	36,137	67,526	879	1	50	77,716	770	19,991	114,269	409.4	3347.7	291.8	6177.7	10,299.3	36,763	19,545.0	36,763
alut2764	387	626	34	320	539	32	1	78	590	15	540	2859	0.1	1.3	1.9	2.8	6.5	657	639.5	**640**

[a] We have not been able to solve this instance on any of our workstations; the memory requirements are more than 1 Gigabyte.

[b] This run was performed on a Sun Ultra 1 Model 170E with the following parameter changes: The primal heuristic was called every 50 iterations (default is 5) and the terminal-distance-test (Step (4) of Algorithm 3.2) was skipped.

TABLE XI. Rectilinear test sets *es10* and *es20*

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
es10a	57	94	10	53	90	10	1	31	243	0	182	1090	0.0	0.1	0.5	0.2	0.8	23,090,747	22,920,745.0	**22,920,745**
es10b	56	92	10	50	83	10	1	27	218	0	155	883	0.0	0.0	0.3	0.1	0.6	19,134,104	19,134,104.0	**19,134,104**
es10c	58	96	10	54	92	10	1	21	171	0	146	778	0.0	0.0	0.2	0.1	0.4	26,126,980	26,003,678.0	**26,003,678**
es10d	50	80	10	46	76	10	1	22	148	0	128	689	0.0	0.0	0.2	0.1	0.3	20,461,116	20,461,116.0	**20,461,116**
es10e	63	106	10	59	102	10	1	16	148	0	138	738	0.0	0.0	0.1	0.1	0.3	18,818,916	18,818,916.0	**18,818,916**
es10f	51	82	10	46	77	9	1	28	209	0	157	909	0.0	0.0	0.3	0.2	0.5	26,831,381	26,540,768.0	**26,540,768**
es10g	48	76	10	38	63	8	1	14	123	0	109	547	0.0	0.0	0.1	0.1	0.2	26,025,072	26,025,072.0	**26,025,072**
es10h	55	90	10	51	86	10	1	21	199	0	160	882	0.0	0.0	0.2	0.1	0.5	25,056,214	25,056,214.0	**25,056,214**
es10i	53	86	10	47	79	9	1	18	160	0	135	747	0.0	0.0	0.1	0.1	0.4	22,062,355	22,062,355.0	**22,062,355**
es10j	50	80	10	45	75	9	1	13	133	0	124	617	0.0	0.0	0.1	0.0	0.3	24,103,248	23,936,095.0	**23,936,095**
es10k	51	82	10	47	78	10	1	26	145	0	121	632	0.0	0.0	0.3	0.1	0.5	22,239,535	22,239,535.0	**22,239,535**
es10l	53	86	10	48	81	9	1	16	119	0	110	531	0.0	0.0	0.0	0.1	0.3	19,626,318	19,626,318.0	**19,626,318**
es10m	45	70	10	40	65	9	1	15	120	0	111	531	0.0	0.0	0.1	0.1	0.3	19,483,914	19,483,914.0	**19,483,914**
es10n	37	54	10	29	46	7	1	16	92	0	76	335	0.0	0.0	0.1	0.0	0.2	21,856,128	21,856,128.0	**21,856,128**
es10o	45	70	10	37	61	7	1	21	97	20	85	417	0.0	0.0	0.1	0.0	0.2	18,641,924	18,641,924.0	**18,641,924**
es20a	273	506	20	269	502	20	1	73	1656	0	737	6926	0.1	0.6	27.9	4.3	33.4	33,733,787	33,703,886.0	**33,703,886**
es20b	281	522	20	277	518	20	1	54	1551	0	716	6764	0.1	0.5	28.3	3.5	32.8	33,225,853	32,639,486.0	**32,639,486**
es20c	225	410	20	221	406	20	1	47	1106	0	584	4666	0.2	0.4	8.7	2.1	11.6	28,528,757	27,847,417.0	**27,847,417**
es20d	269	498	20	265	494	20	5	110	2448	72	668	6937	0.1	23.2	74.9	7.2	106.3	27,686,681	27,624,394.0	**27,624,394**
es20e	271	502	20	267	498	20	3	69	1708	46	724	6858	0.1	10.2	43.4	4.7	59.1	34,531,076	34,033,163.0	**34,033,163**
es20f	272	504	20	267	499	19	1	71	1305	0	671	5976	0.1	0.6	15.4	3.6	20.1	36,412,596	36,014,241.0	**36,014,241**
es20g	273	506	20	269	502	20	1	70	1801	0	694	7314	0.1	0.6	42.6	4.8	48.5	35,418,856	34,934,874.0	**34,934,874**
es20h	240	440	20	235	435	19	3	73	1845	24	609	6542	0.1	10.4	38.5	4.1	53.9	38,719,129	38,016,346.0	**38,016,346**
es20i	286	532	20	282	528	20	1	48	1442	0	695	6433	0.1	0.5	23.6	3.4	27.9	36,739,939	36,739,939.0	**36,739,939**
es20j	252	464	20	248	460	20	1	51	1338	0	647	5951	0.1	0.4	18.4	2.7	21.9	34,872,088	34,024,740.0	**34,024,740**
es20k	270	500	20	266	496	20	1	80	1762	0	647	6615	0.2	0.7	35.4	5.2	41.9	27,337,099	27,123,908.0	**27,123,908**
es20l	255	470	20	251	466	20	1	59	1666	0	652	6614	0.1	0.5	36.7	3.5	41.1	30,911,159	30,451,397.0	**30,451,397**
es20m	246	452	20	241	447	19	1	58	1142	0	663	5742	0.1	0.4	13.2	2.6	16.7	34,552,183	34,438,673.0	**34,438,673**
es20n	252	464	20	248	460	20	1	67	1769	0	671	7122	0.1	0.6	34.2	4.2	39.6	34,062,374	34,062,374.0	**34,062,374**
es20o	247	454	20	242	449	19	1	43	1256	0	620	5399	0.1	0.3	16.8	2.6	20.0	32,582,309	32,303,746.0	**32,303,746**

TABLE XII. Rectilinear test sets *es30* and *es40*

	Original			Presolved			B & C			Root LP			Time					Solutions		
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Nod	Iter	Cuts	Frac	Rows	NZ	Pre	Heu	LP	Sep	Tot	Heu(1)	LB	UB
es30a	664	1268	30	660	1264	30	1	120	6045	0	1483	20,589	0.2	4.4	790.1	37.9	833.7	41,445,994	40,692,993.0	**40,692,993**
es30b	646	1232	30	642	1228	30	3	171	7156	51	1485	21,423	0.2	35.6	1223.5	54.8	1315.6	41,799,775	40,900,061.0	**40,900,061**
es30c	659	1258	30	655	1254	30	1	289	12,690	0	1461	21,762	0.2	10.1	3877.9	115.4	4005.2	44,136,228	43,120,444.0	**43,120,444**
es30d	677	1294	30	673	1290	30	1	161	7134	0	1440	18,779	0.2	5.8	1506.0	57.0	1570.1	42,961,468	42,150,958.0	**42,150,958**
es30e	660	1260	30	656	1256	30	1	177	7380	0	1469	20,352	0.2	6.4	1473.5	56.3	1537.5	41,951,226	41,739,748.0	**41,739,748**
es30f	606	1152	30	602	1148	30	1	188	7456	0	1415	21,015	0.2	6.2	1203.4	59.3	1270.3	40,808,517	39,955,139.0	**39,955,139**
es30g	671	1282	30	665	1276	29	1	180	7031	0	1491	21,896	0.2	6.0	1401.8	51.1	1460.3	45,613,589	43,761,391.0	**43,761,391**
es30h	587	1114	30	582	1109	29	1	213	7390	0	1348	20,717	0.2	6.7	1295.2	56.3	1359.5	42,076,236	41,691,217.0	**41,691,217**
es30i	656	1252	30	650	1245	29	1	165	6992	0	1517	19,446	0.2	5.4	826.3	44.4	877.5	37,612,954	37,133,658.0	**37,133,658**
es30j	633	1206	30	628	1201	29	1	121	4975	0	1470	18,891	0.2	4.0	500.5	29.2	534.8	43,232,987	42,686,610.0	**42,686,610**
es30k	673	1286	30	669	1282	30	1	149	6919	0	1529	21,258	0.2	5.2	1203.8	48.6	1258.9	42,050,341	41,647,993.0	**41,647,993**
es30l	555	1050	30	548	1038	30	1	71	2750	0	1350	12,342	0.2	2.1	84.8	11.9	99.6	39,120,826	38,416,720.0	**38,416,720**
es30m	598	1136	30	592	1130	28	1	159	4763	0	1325	16,610	0.3	4.6	478.2	35.5	519.6	37,685,476	37,406,646.0	**37,406,646**
es30n	694	1328	30	688	1321	29	1	134	5871	20	1608	20,845	0.3	5.1	664.4	38.5	709.2	45,159,326	42,897,025.0	**42,897,025**
es30o	632	1204	30	627	1199	29	1	223	9704	0	1447	22,433	0.3	7.4	2632.5	74.6	2715.9	44,344,003	43,035,576.0	**43,035,576**
es40a	1181	2282	40	1175	2275	39	1	185	11,123	0	2387	33,544	13.2	16.3	5172.5	123.2	5327.5	45,629,452	44,841,522.0	**44,841,522**
es40b	1133	2186	40	1128	2181	39	1	170	10,087	0	2332	36,418	10.8	14.6	3628.8	135.6	3791.6	48,704,740	46,811,310.0	**46,811,310**
es40c	1162	2244	40	1158	2240	40	1	245	14,188	0	2254	34,859	12.0	22.5	9592.6	256.2	9886.3	51,414,386	49,974,157.0	**49,974,157**
es40d	1129	2178	40	1125	2174	40	1	175	11,042	0	2535	34,842	11.4	15.1	4899.7	145.0	5073.4	45,615,171	46,289,864.0	**45,289,864**
es40e	1296	2512	40	1292	2508	40	1	199	12,439	1850	2827	48,217	16.4	19.2	9864.3	177.2	10,079.9	52,406,272	51,392,344.3	52,016,120
es40f	1114	2148	40	1109	2143	40	1	379	18,120	1144	2472	42,023	10.8	33.0	9616.2	366.6	10,030.3	49,893,557	49,737,564.6	49,765,043
es40g	1172	2264	40	1164	2254	39	1	140	9181	0	2497	34,988	12.9	12.7	3037.9	110.4	3175.8	46,551,607	45,639,009.0	**45,639,009**
es40h	1262	2444	40	1254	2436	39	1	180	12,845	1663	2606	40,227	15.9	17.3	9825.5	168.1	10,029.3	49,953,763	48,739,666.4	48,745,996
es40i	1232	2384	40	1228	2380	40	1	228	14,657	1802	2597	42,316	15.1	22.4	9789.6	231.5	10,061.9	52,859,369	51,557,587.2	51,761,789
es40j	1255	2430	40	1251	2426	40	1	145	11,054	1788	2678	43,248	15.1	15.8	9894.1	139.6	10,066.7	58,390,862	56,761,892.0	57,414,203
es40k	1192	2304	40	1187	2299	40	1	201	12,952	0	2543	41,924	13.5	19.4	8631.1	182.8	8849.3	47,719,938	46,734,214.0	**46,734,214**
es40l	1261	2442	40	1256	2437	40	1	178	9872	0	2632	37,136	16.1	18.4	4280.2	160.7	4477.2	45,088,751	43,843,378.0	**43,843,378**
es40m	1381	2682	40	1377	2678	40	1	169	10,906	0	2881	41,255	18.1	18.7	5432.0	197.4	5668.3	52,374,560	51,884,545.0	**51,884,545**
es40n	1313	2546	40	1309	2542	40	1	195	12,127	1616	2777	45,105	16.6	21.6	9793.2	196.1	10,029.7	49,967,268	48,924,948.1	49,448,257
es40o	1307	2534	40	1300	2527	40	1	232	14,712	0	2575	40,762	17.1	24.8	8551.0	51.7	8847.2	51,340,298	50,828,067.0	**50,828,067**

limit given, it usually pays to call all reduction methods to reduce the problem as much as possible in size. For instance, we solve example *diw0234* with over 10,000 variables in about 24,000 seconds. The complete presolve reduces the problem from 10,086 edges to 7266, whereas Algorithm 3.2 reduces it just to 9991 edges. However, over 12,000 seconds are spent in presolve when the complete reduction test is performed and only 6 seconds when Algorithm 3.2 is applied. (With the default parameter setting, we obtain after 10,000 seconds an upper bound of 1997 and a lower bound of 1967 providing a solution guarantee of 1.5%.) The difficulty of the VLSI problems seem not only depend on the number of terminals, but also on the shape of the grid graphs, how many holes are there, and how big these holes are. Figure 9 shows a typical diagram for these problems. The numbers of fractional variables continuously increase (see the decrease of curve *Integer*), and the LPs get more and more difficult during the runs (see the number of simplex iterations).

Although our code was originally designed for solving Steiner tree problems in graphs, it is, of course, also possible to solve rectilinear instances by modeling them as graph problems. Tables XI and XII show results on rectilinear problems. Table XI contains the examples from Beasley with 10 and 20 terminals. They are not very difficult (up to 4 minutes), although branching is necessary in three cases. However, the situation changes for test sets *es30* and *es40*. The running times rapidly increase with the number of terminals and we are not able to solve all instances with 40 terminals within 10,000 seconds. Our diagram, for example, *es40o* in Figure 10, shows that the LPs become increasingly difficult during the run of the program, a behavior that we have already detected to some extent for the VLSI examples. In fact, the LPs are highly dual and primal degenerated, a phenomenon that seems to be inherent for grid problems (see also Grötschel et al. [22]). Another drawback is that our presolve procedures do not perform well. Reduction methods (as proposed for instance by Winter [41]) that exploit the structure of grid graphs would probably help to solve these instances faster. Recently, Warme [39] proposed an algorithm for rectilinear Steiner tree problems. By exploiting the typical structure of rectilinear problems, he was able to solve much bigger instances in less time.

6. CONCLUSIONS

We have presented an implementation of a branch-and-cut algorithm for the Steiner tree problem in graphs. We are able to solve almost all instances discussed in the literature. Our algorithm especially performs well on complete and sparse graphs. Here, a good presolve seems to pay. We have also introduced new real-world VLSI instances. We solve many of these instances and provide reasonable solution guarantees (in general, below 15%) for all examples except for the really big ones with several hundred terminals and tens of thousands of edges. On rectilinear Steiner tree problems, our code performs well only for examples with a small number of terminals. To be competitive with state-of-the-art software for rectilinear problems, our reduction methods have to be adapted to rectilinear instances and more investigations are necessary to avoid degenerated linear programs. All examples discussed in this paper are gathered in a newly introduced library called *SteinLib* that is accessible via anonymous ftp or the World Wide Web.

REFERENCES

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

[2] A. Balakrishnan and N. Patel, Problem reduction methods and a tree generation algorithm for the Steiner network problem. *Networks* **17** (1987) 65–85.

[3] M. Ball, W. Liu, and W. Pulleyblank, Two terminal Steiner tree polyhedra. *Contributions to Operations Research and Economics—The Twentieth Anniversary of CORE* (B. Cornet and H. Tulkens, eds.). MIT Press, Cambridge, MA (1989) 251–284.

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

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

[6] R. Bixby, Personal communication (1996).

[7] A. Caprara and M. Fischetti, Branch-and-cut algorithms. *Annotated Bibliographies in Combinatorial Optimization* (M. Dell'Amico, F. Maffioli, and S. Martello, Eds.). Wiley, Chichester (1997) 45–63.

[8] S. Chopra, E. Gorres, and M. R. Rao, Solving a Steiner tree problem on a graph using branch and cut. *ORSA J. Comput.* **4** (1992) 320–335.

[9] S. Chopra and M. R. Rao, The Steiner tree problem I: Formulations, compositions and extension of facets. *Math. Program.* **64** (1994) 209–229.

[10] S. Chopra and M. R. Rao, The Steiner tree problem II: Properties and classes of facets. *Math. Program.* **64** (1994) 231–246.

[11] S. E. Dreyfus and R. A. Wagner, The Steiner problem in graphs. *Networks* **1** (1971) 195–207.

[12] C. Duin, Steiner's problem in graphs. PhD thesis, University of Amsterdam (1993).

[13] C. Duin and A. Volgenant, An edge elimination test for the Steiner problem in graphs. *Oper. Res. Lett.* **8** (1989) 79–83.

[14] C. Duin and A. Volgenant, Reduction tests for the Steiner problem in graphs. *Networks* **19** (1989) 549–567.

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

[16] C. E. Ferreira, O problema de Steiner em grafos: Uma abordagem poliédrica. Master's thesis, Universidade de São Paulo (1989).

[17] M. Fischetti, Facets of two Steiner arborescence polyhedra. *Math. Program.* **51** (1991) 401–419.

[18] M. R. Garey and D. S. Johnson, The rectilinear Steiner tree problem is np-complete. *SIAM J. Appl. Math.* **32** (1977) 826–834.

[19] M. X. Goemans, The Steiner tree polytope and related polyhedra. *Math. Program.* **63** (1994) 157–182.

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

[21] A. Goldberg and R. Tarjan, A new approach to the maximum flow problem. *J. ACM* **35** (1988) 921–940.

[22] M. Grötschel, A. Martin, and R. Weismantel, Packing Steiner trees: A cutting plane algorithm and computational results. *Math. Program.* **72** (1996) 125–145.

[23] M. Grötschel and C. L. Monma, Integer polyhedra associated with certain network design problems with connectivity constraints. *SIAM J. Discr. Math.* **3** (1990) 502–523.

[24] J. Hao and J. B. Orlin, A faster algorithm for finding the minimum cut in a graph. *Proceedings of the Third Annual ACM-Siam Symposium on Discrete Algorithms,* Orlando, FL (1992) 165–174.

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

[26] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem.* Annals of Discrete Mathematics **53,** North-Holland, Amsterdam (1992).

[27] M. Jünger, A. Martin, G. Reinelt, and R. Weismantel, Quadratic 0/1 optimization and a decomposition approach for the placement of electronic circuits. *Math. Program.* **63** (1994) 257–279.

[28] R. M. Karp, Reducibility among combinatorial problems. *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, Eds.), Plenum Press, New York (1972) 85–103.

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

[30] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, Chichester (1990).

[31] A. Lucena, Tight bounds for the Steiner problem in graphs. Preprint, IRC for Process Systems Engineering, Imperial College, London (1993).

[32] N. Maculan, The Steiner problem in graphs. *Ann. Discr. Math.* **31** (1987) 185–212.

[33] F. Margot, Personal communication (1994).

[34] M. Padberg and G. Rinaldi, A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review* **33** (1991) 60–100.

[35] V. J. Rayward-Smith, The computation of nearly minimal Steiner trees in graphs. *Int. J. Math. Educ. Sci. Technol.* **14** (1983) 15–23.

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

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

[38] H. Takahashi and A. Matsuyama, An approximate solution for the Steiner problem in graphs. *Math. Jpn.* **24** (1980) 573–577.

[39] D. M. Warme, A new exact algorithm for rectilinear Steiner trees. Technical report, Telenex Corporation, Springfield, VA 22153 (1997).

[40] P. Winter, Steiner problems in networks: A survey. *Networks* **17** (1987) 129–167.

[41] P. Winter, Reductions for the rectilinear Steiner tree problem. Research Report 11-95, Rutcors University (1995).

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

[43] R. T. Wong, A dual ascent approach for Steiner tree problems on a directed graph. *Math. Program.* **28** (1984) 271–287.