Mathematical Programming 72 (1996) 125–145

Packing Steiner trees: a cutting plane algorithm and computational results

M. Grötschel *, A. Martin, R. Weismantel

Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, 14195 Berlin, Germany

Received 14 January 1993; revised manuscript received 28 August 1995

Abstract

In this paper we describe a cutting plane algorithm for the Steiner tree packing problem. We use our algorithm to solve some switchbox routing problems of VLSI-design and report on our computational experience. This includes a brief discussion of separation algorithms, a new LP-based primal heuristic and implementation details. The paper is based on the polyhedral theory for the Steiner tree packing polyhedron developed in our companion paper (this issue) and meant to turn this theory into an algorithmic tool for the solution of practical problems.

Keywords: Branch and cut; Packing; Routing; Separation; Steiner tree; VLSI-design

1. Introduction

Given a graph $G=(V,E)$ and a node set $T \subseteq V$, we call an edge set $S \subseteq E$ a *Steiner tree for T* if, for each pair of nodes $u, v \in T$, S contains a $[u,v]$-path. The *Steiner tree packing problem*, as introduced in [12], can be stated as follows. Given an undirected graph $G=(V,E)$ with edge capacities $c_e \in \mathbb{N}$ for all $e \in E$ and a list of node sets $\mathcal{N} = \{T_1, \ldots, T_N\}$, $N \in \mathbb{N}$, find Steiner trees S_k for T_k, $k = 1, \ldots, N$, such that each edge $e \in E$ is contained in at most c_e of the edge sets $S_1, \ldots, S_N$. Every collection of Steiner trees $S_1, \ldots, S_N$ with this property is called a Steiner tree packing. If a weighting of the edges is given in addition and a (with respect to this weighting) minimal Steiner tree packing must be found, we call this the *weighted Steiner tree packing problem*.

The motivation for studying this problem arises from the design of electronic circuits, i.e., the task of casting a given (complex) logic function in silicon. In a first phase

* Corresponding author.

SSDI 0025-5610(95)00038-0

(*logical design*) it is specified which of the given basic logical operations are combined to logical units (so-called *cells*) and which of these cells must be connected via wires. The points at which wires have to contact the cells are called *terminals*, and a set of terminals that must be connected is called a *net*. The list of cells and the list of nets are the input of the second phase, the *physical design*. The task here consists in assigning (placing) the cells to a given area and connecting (routing) the nets via wires. The problem, in fact, is more complicated than sketched above, since various company given design rules and technical constraints have to be taken into account and an objective function like the resulting area has to be minimized. Due to the inherent complexity, the problem is usually decomposed into the *placement problem* and the *routing problem*. We are interested in the routing problem. Roughly speaking, this problem can be stated as follows.

> Given an area (typically a rectangle with some "forbidden zones" occupied by the cells) and a list of nets. The routing problem is to connect (route) the terminals of each net by wires on the area such that certain technical side constraints are satisfied and some objective function is minimized.

The routing problem strongly depends on the used fabrication technology and the underlying design rules. The design rules specify, for instance, the routing area (i.e., the area that is available for connecting the nets) or the objective function (possible choices are, for example, the wiring length or the resulting area). The routing itself takes place on so-called *layers*. Each layer is divided into *tracks* on which the wires run. The tracks and the *vias*, the points where wires change the layers, must meet certain distance requirements.

The routing problem in its general form is still too complex to be solved in one step. In practice, the problem is generally decomposed into two subproblems. In a first step, one determines how wires "maneuver around the cells" (*global routing*). Here, the design rules are only partially considered. Thereafter, the wires are assigned to the layers and tracks according to the homotopy which was specified in the global routing phase (*detailed routing*). This decomposition scheme gives rise to many variants of the routing problem.

A number of the routing problems resulting from this approach can be modelled as a (weighted) Steiner tree packing problem. We will illustrate two examples in the following.

For modelling the global routing problem, the routing area is subdivided into subareas and these are represented by nodes in a graph. Of course, there are many ways to do this. One possible way of subdividing the routing area is illustrated in Fig. 1. The enclosing rectangle represents the given area. The rectangular units with a diagonal between their lower left and upper right corner denote the cells. The routing area is subdivided into rectangular subareas (by means of the additional dotted lines in Fig. 1). This subdivision of the routing area is represented by a graph as follows. We define a node for each subarea and introduce an edge between two nodes, if the corresponding subareas are adjacent. Let $G = (V, E)$ denote the resulting graph. Additionally, a

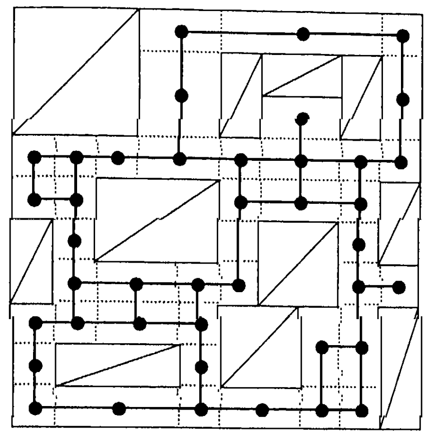


Fig. 1.

capacity $c_{uv} \in \mathbb{N}$ is assigned to an edge $uv \in E$ limiting the number of nets that may run between the subareas associated with the two nodes u and v. The weight of an edge w_{uv} corresponds to the distance between the two midpoints of the according subareas. The terminals of a net are assigned to those nodes, whose corresponding subareas contain the terminal or are as close as possible to the position of the terminal. The global routing

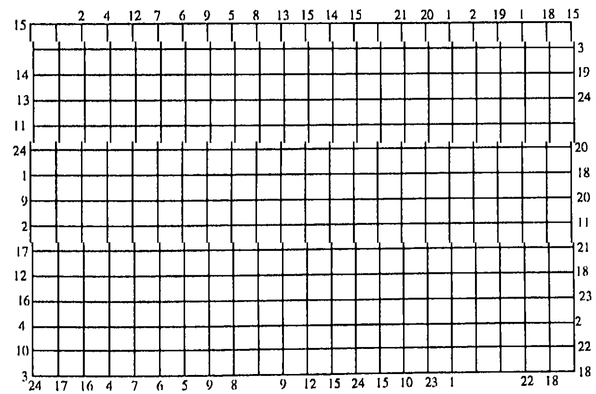


Fig. 2.

problem consists in routing all nets in G such that the capacity constraints are satisfied and the total wiring length is as small as possible. Obviously, this task defines an instance of the Steiner tree packing problem.

The second example we want to mention is a variant of the detailed routing problem called *switchbox routing problem* (see Fig. 2). Here, the underlying graph is a complete rectangular grid graph and the terminal sets are located on the four sides of the grid. Remember that the task of detailed routing is to assign the wires to layers and tracks. Detailed routing problems, and thus also switchbox routing problems, are classified by distinguishing whether or to which extent the layers are taken into account while the nets are assigned to tracks. Here, the following models are of special interest.

Multiple layer model. Given a k layered grid graph (that is a graph obtained by stacking k copies of a grid graph on top of each other and connecting related nodes by perpendicular lines), where k denotes the number of layers. The nets have to be routed in a node disjoint fashion. The multiple layer model is well suited to reflect reality. The disadvantage is that in general the resulting graphs are very large.

Manhattan model. Given a (subgraph of a) complete rectangular grid graph. The nets must be routed in an edge disjoint fashion with the additional restriction that nets that meet at some node are not allowed to bend at this node, i.e., so-called *knock-knees* (cf. Fig. 3) are not allowed. This restriction guarantees that the resulting routing can be realized on two layers at the possible expense of causing long detours.

Knock-knee model. Again, a (subgraph of a) complete rectangular grid graph is given and the task is to find an edge disjoint routing of the nets. In this model knock-knees are possible. Very frequently, the wiring length of a solution in this case is smaller than in the Manhattan model. The main drawback is that the assignment to layers is neglected. Brady and Brown [2] have designed an algorithm that guarantees that any solution in

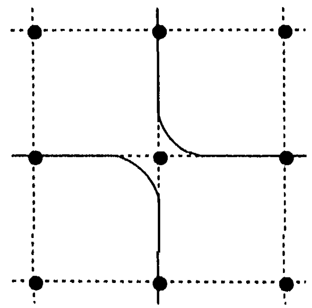


Fig. 3.

this model can be routed on four layers. It was shown in [18] that it is $\mathcal{NP}$-complete to decide whether a realization on three layers is possible.

As in the case of the global routing problem the weighted Steiner tree packing problem is a natural mathematical model of the switchbox routing problem in the knock-knee mode. All examples that this computational study reports on are instances of this type of switchbox routing problems. Of course, we only sketched some of the routing problems arising in the design of electronic circuits and that can be modelled as Steiner tree packing problems. For more details on this subject we refer to the excellent book of Lengauer [17].

The paper is organized as follows. In Section 2 we briefly discuss the separation problem for several classes of inequalities introduced in [12]. Implementation issues of our branch and cut algorithm will be mentioned in Section 3. Finally, we report on computational results for several switchbox routing instances in Section 4.

Notation

We use the same notation as in [12]. Thus, we restrict ourselves in this subsection to briefly summarize the main notation concerning the (weighted) Steiner tree packing problem.

Let $G = (V, E)$ be a graph and $T \subseteq V$ a node set of G. An edge set S is called a *Steiner tree for T*, if the subgraph $(V(S), S)$ contains a path from s to t for all pairs of nodes $s, t \in T$, $s \neq t$.

Problem 1.1 (*The weighted Steiner tree packing problem*).

Instance: A graph $G = (V, E)$ with positive, integer capacities $c_e \in \mathbb{N}$ and non-negative weights $w_e \in \mathbb{R}_+$, $e \in E$. A list of node sets $\mathcal{N} = \{T_1, \ldots, T_N\}$, $N \geq 1$, with $T_k \subseteq V$ for all $k = 1, \ldots, N$.

Problem: Find edge sets $S_1, \ldots, S_N \subseteq E$ such that

(i) S_k is a Steiner tree in G for T_k for all $k = 1, \ldots, N$,

(ii) $\sum_{k=1}^{N} |S_k \cap \{e\}| \leq c_e$ for all $e \in E$,

(iii) $\sum_{k=1}^{N} \sum_{e \in S_k} w_e$ is minimal.

If requirement (iii) in Problem 1.1 is omitted we call the corresponding problem the *Steiner tree packing problem* without the prefix "weighted". The list of node sets $\mathcal{N}$ is called a *net list*. Any element $T_k \in \mathcal{N}$ is called a *set of terminals* and the nodes $t \in T_k$ are called *terminals*. Instead of terminal set T_k we will often simple say *net k*. We call an N-tuple $(S_1, \ldots, S_N)$ of edge sets a *Steiner tree packing* or *packing of Steiner trees* if the sets $S_1, \ldots, S_N$ statisfy (i) and (ii) of Problem 1.1. We will refer to an instance of the weighted Steiner tree packing problem by $(G, \mathcal{N}, c, w)$ and to an instance of the Steiner tree packing problem by $(G, \mathcal{N}, c)$. $\mathbb{R}^{\mathcal{N} \times E}$ denotes the $N \cdot |E|$-dimensional vector space $\mathbb{R}^E \times \ldots \times \mathbb{R}^E$, where the components of each vector $x \in \mathbb{R}^{\mathcal{N} \times E}$ are indexed by x_e^k for $k \in \{1, \ldots, N\}$, $e \in E$. The Steiner tree packing polyhedron $\mathrm{STP}(G, \mathcal{N}, c) \subset \mathbb{R}^{\mathcal{N} \times E}$ is the convex hull of all incidence vectors of Steiner tree packings.

We assume throughout the paper that every terminal set of the net list $\mathcal{N}$ has at least cardinality two and that $N \geq 1$.

Note that the Steiner tree packing problem as well as its weighted variant are $\mathcal{NP}$-complete or $\mathcal{NP}$-hard, respectively (see, for example, [8,15,16]). The problem remains hard in the case of switchbox routing problems in the knock-knee model (see [22]).

2. The separation problem for several classes of inequalities

In this section we briefly sketch some of the ideas for separating the classes of inequalities presented in [12]. The separation algorithms and the associated correctness proofs are quite complicated. For details of this issue we refer the reader to [20] and [11]. Formally, the *separation problem* for a given class of inequalities can be stated as follows.

Given an instance $(G, \mathcal{N}, c)$ of the Steiner tree packing problem, a vector $y \in \mathbb{R}^{\mathcal{N} \times E}$, $y \geq 0$, and a class of valid inequalities for STP $(G, \mathcal{N}, c)$. Decide, whether y satisfies all inequalities of the given class and, if not, find an inequality of this class violated by y.

2.1. *Separation of the Steiner partition inequalities*

Due to [12, Theorem 5.3] every facet-defining inequality for the Steiner tree polyhedron yields a valid and, in case G is complete and the net list is disjoint, a facet-defining inequality for STP $(G, \mathcal{N}, c)$. We focus here on one class of facet-defining inequalities that was characterized in [13]. Let G be a graph and $T \subseteq V$ be a terminal set. We call a partition $V_1, \ldots, V_p$, $p \geq 2$, of V a *Steiner partition with respect to T*, if $V_i \cap T \neq \emptyset$ for $i = 1, \ldots, p$. The inequality

$$x(\delta(V_1, \ldots, V_p)) \geq p - 1$$

is called *Steiner partition inequality*. It is valid for STP $(G, \{T\}, \mathbb{1})$ and Grötschel and Monma have characterized conditions under which it defines a facet. Though the corresponding separation problem is $\mathcal{NP}$-complete in general [14], there exist special cases for which it can be solved in polynomial time. One of these special cases is obtained if we restrict the graph G to be planar and the set of terminal nodes T to lie on the outer face of G. This special case is of particular practical interest, because it includes the switchbox routing problem. The main idea of the algorithm for solving the separation problem in this case is as follows.

Without loss of generality we can assume that G is 2-node connected (otherwise the graph can be decomposed). Thus, the edge set that encloses the outer face of G is a cycle. Suppose the terminal set $T = \{t_1, \ldots, t_z\}$ is numbered in a clockwise fashion along this cycle. Now, consider the dual graph $G^* = (V^*, E)$ of G and subdivide the node representing the outer face in z nodes $d_1, \ldots, d_z$ such that every edge belonging to a

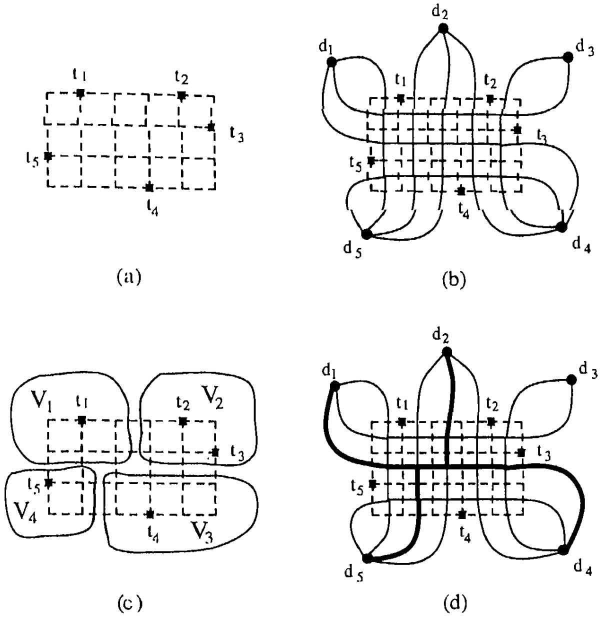


Fig. 4.

path in G from t_i to t_{i+1} on the outer face is now incident to d_{i+1} for $i = 1, \ldots, z$. Let $G_D = (V_D, E)$ denote the resulting graph and set $D = \{d_1, \ldots, d_z\}$ (cf. Figs. 4(a) and 4(b)).

It turns out that under mild assumptions every edge set $S = \delta(V_1, \ldots, V_l)$ induced by Steiner partition $V_1, \ldots, V_l$ is in one-to-one correspondence with an edge-minimal Steiner tree in G_D with respect to some subset $J \subseteq D$ (cf. Figs. 4(c) and 4(d)).

This equivalence yields that the problem of separating the class of Steiner partition inequalities reduces to the problem of finding a subset J of D and an edge-minimal Steiner tree in G_D with respect to J. Given J, we can determine an optimal Steiner tree in G_D with respect to J by applying the dynamic programming approach proposed in [6,7]. Thus, the crucial point is to find the subset $J \subseteq D$. In [11] we show that we can locally decide which terminal belongs to an optimal solution. This observation can be taken into account by modifying the recursion formula of the dynamic program appropriately.

The algorithm for separating the Steiner partition inequalities gives rise to several heuristic procedures. Instead of calculating the optimal Steiner tree in G_D we heuristically determine Steiner trees. For more details, we again refer to our paper [11].

2.2. Separation of the alternating cycle inequalities

Given an instance $(G, \mathcal{N}, c)$ of a Steiner tree packing problem with $\mathcal{N} = \{T_1, T_2\}$ and a vector $y \in \mathbb{R}^{\mathcal{N} \times E}$, $y \geq 0$, decide, whether y satisfies all alternating cycle inequalities (see [12, Theorem 6.2]). If this is not the case, find an alternating cycle inequality that is violated by y.

Yet not proved, we strongly conjecture that, in general, this problem is not solvable in polynomial time. Instead, we restrict our attention to the case where G is planar and all terminals lie on the outer face of G. Here, our idea to separate alternating cycle inequalities is to apply dynamic programming techniques in a similar way as was done for finding Steiner partition inequalities.

Again, we show that alternating cycle inequalities are in one-to-one mapping with Steiner trees in an appropriate dual graph. In this case, however, these Steiner trees have to satisfy many technical conditions.

In particular, these technical conditions cause that some edges are evaluated differently for different nets. This is due to the fact that for the alternating cycle inequality, edge sets F (edges which have a zero coefficient for both nets), F_1 (edges which have zero coefficient just for net 1) and F_2 (edges which have a zero coefficient just for net 2) are involved (cf. [12, Theorem 6.2]). Unfortunately, taking all these constraints into account we obtain a dynamic program, whose optimum solution does not necessarily correspond to the most violated alternating cycle inequality. Rather, the optimum value found by the dynamic program provides just a lower bound for the most violated alternating cycle inequality. If this value is nonnegative, we can guarantee that there does not exist a violated inequality of this type. Otherwise, there may exist a violated alternating cycle inequality, but the algorithm terminates with an edge set that does not correspond to an alternating cycle inequality (see [11]).

Beyond that the relationship between alternating cycle inequalities and Steiner trees satisfying certain technical conditions in the appropriate dual graph gives rise to many heuristics. Again, we have implemented an algorithm that determines heuristically such Steiner trees and checks whether the corresponding alternating cycle inequalities are violated.

2.3. Finding critical cuts

Remember that a cut induced by a set of nodes W is critical, if $s(W) := c(\delta(W)) - |S(W)| \leq 1$, where $S(W) := \{k \in \{1, \ldots, N\} \mid T_k \cap W \neq \emptyset,\ T_k \cap (V \setminus W) \neq \emptyset\}$. In the following we briefly explain why we concentrate on the problem of finding critical cuts rather than on the separation problem for the critical cut inequalities itself.

First, let us point out that, from a practical point of view, we are interested in Steiner tree packings where each of the single Steiner trees is edge-minimal. Since a positive objective function is minimized, we know in advance that the weight-minimal Steiner trees are also edge-minimal, and we exploit this property to reduce the problem size.

Suppose $W \subsetneq V$ is a node set and T_k is a set of terminals with $T_k \subseteq W$ or $T_k \subseteq V \setminus W$. Then any edge-minimal Steiner tree for T_k that uses one edge of $\delta(W)$ has to contain at least two of these edges. But, if $\delta(W)$ is a critical cut then at most one edge of $\delta(W)$ can be used by the Steiner tree for T_k. Hence, the following variables can be fixed accordingly, i.e.,

$$x_e^k = 0, \quad \text{for all } k \in \{1, \ldots, N\} \setminus S(W),\ T_k \subseteq W,\ e \in E(V \setminus W) \cup \delta(W),$$

$$x_e^k = 0, \quad \text{for all } k \in \{1, \ldots, N\} \setminus S(W),\ T_k \subseteq V \setminus W,\ e \in E(W) \cup \delta(W).$$

Let us now point out the relationship to the critical cut inequality. Consider the situation in [12, Definition 6.9(b)], where V_1, V_2, V_3 is a partition of V such that $\delta(V_1)$ is a critical cut and $T_1 \cap V_1 = \emptyset$ and $T_1 \cap V_i \neq \emptyset$, $i = 2, 3$. Since $\delta(V_1)$ is critical, we can fix all variables x_e^1 to zero for $e \in \delta(V_1)$. Thus, by fixing these variables we can separate the critical cut inequalities via separating the Steiner cut inequalities. For example, a Steiner cut inequality for T_1 of the instance described in [12, Definition 6.9(b)] is $x^1(\delta(V_2)) = x^1([V_2 : V_1]) + x^1([V_2 : V_3]) \geq 1$. By taking the fixed variables into account we obtain the critical cut inequality $x^1([V_2 : V_3]) \geq 1$.

In the remainder of this subsection we briefly sketch the ideas, how to find critical cuts. We restrict ourselves to instances $(G, \mathcal{N}, \mathbb{1})$, where G is a complete rectangular grid graph and all terminal sets of the net list $\mathcal{N}$ lie on the outer face of G. Here, we can show [20] that, if there exists a node set $W \subset V$, $W \neq \emptyset$ that induces a critical cut, there exists

(i) a node $w \in V$ such that $\delta(w)$ is a critical cut with respect to $(G, \mathcal{N}, \mathbb{1})$ or

(ii) a horizontal or vertical critical cut with respect to $(G, \mathcal{N}, \mathbb{1})$. (A cut F is called *horizontal* if there exists some $i \in \{1, \ldots, h-1\}$ such that $F = \{uv \in E \mid u = (i, j)$ and $v = (i+1, j)$ for some $j \in \{1, \ldots, b\}\}$; a vertical cut is defined accordingly).

Based on this observation we can now develop an algorithm for finding critical cuts. We check, for all nodes $v \in V$, whether $\delta(v)$ is critical. In addition, we also check whether there exists critical vertical or horizontal cuts. If we do not succeed in finding a critical cut, we can conclude that none exists. Otherwise, we fix the corresponding variables. In order to find further critical cuts, we inductively enlarge the node set $W = \{v\}$ in all four possible directions of the grid in a greedy like fashion. The variables of the critical cuts found this way are fixed accordingly.

Finally, we have developed a heuristic for separating grid inequalities that proceeds in a greedy-like fashion. Details can be found in [10].

3. Implementation of the cutting plane algorithm

In this section we discuss further features of our cutting plane algorithm for the (weighted) Steiner tree packing problem. Since we assume the reader being familiar with this method, we can avoid outlining how cutting plane algorithms work in general.

We next introduce a primal heuristic for the switchbox routing problem. Thereafter, some implementation details are discussed that are indispensable is solving switchbox routing problem instances.

3.1. A primal heuristic

This section is devoted to describing our primal heuristic. The idea of our heuristic is to make use of the information given be the actual solution of the cutting plane phase.

We have developed a sequential algorithm. We consider each terminal of a net to be an (isolated) component. We iteratively connect two components of a net according to an a priori determined sequence. However, we do not apply this scheme by routing one net completely after another, but we connect only two components in each iteration. The success of such a procedure strongly depends on the predefined sequence. In our algorithm this sequence is mainly determined by the solution y of the actual linear program. More precisely, we define a function f depending on y according to which the subsequent two components are selected. (A detailed explanation of the function f is given after the algorithmic description of the heuristic.) We try to connect the two selected components via a shortest path. Since in a complete rectangular grid graph a shortest path is not unique in general, we have implemented further criteria according to which the choice is made. Besides others, these criteria depend on the location of the terminals of the other nets, the position of the not yet connected terminals of the same net and, again, on the solution y. For a detailed description of these criteria we refer the reader to [20]. If it is possible to connect the two components on a shortest path by taking the mentioned criterion into account, we connect these two components and choose the next pair of components. Otherwise, we recompute the function f and the sequence by taking the already connected components into account. This iterative procedure is continued until all nets are connected or no further components can be connected. In detail, the algorithm can be described as follows.

Algorithm 3.1 (A primal heuristic).

Input: A complete rectangular $h \times b$ grid graph $G=(V, E)$ with edge capacities $c_e = 1$ and edge weights $w_e \in \mathbb{R}_+$, $e \in E$. Furthermore, a net list $\mathcal{N} = \{T_1, \ldots, T_N\}$ and a vector $y \in \mathbb{R}^{\mathcal{N} \times E}$, $y \geq 0$.

Output: A feasible solution of the weighted Steiner tree packing problem $(G, \mathcal{N}, \mathbb{1}, w)$ or the message "No feasible solution found".

(1) Set $S_k := \emptyset$ for $k = 1, \ldots, N$.

(2) Determine the graph $\hat{G} = (V, \hat{E})$ with $\hat{E} := \{e \in E \mid c_e > 0\}$.

(3) Compute shortest paths for all pairs of nodes in $\hat{G}$.

(4) For $k := 1, \ldots, N$ perform the following steps:

(5) If $S_k = \emptyset$, then
    determine $s_k, t_k \in T_k$ such that
    $f_{y^k}(s_k, t_k) = \min_{\substack{s,t \in T_k \\ s \neq t}} f_{y^k}(s, t)$;
    set $T'_k := T_k \setminus \{t_k\}$.

(6) Else
    determine $s_k \in T'_k$, $t_k \in V(S_k)$ such that
    $f_{y^k}(s_k, t_k) := \min_{s \in T'_k, t \in V(S_k)} f_{y^k}(s, t)$.

(7) As long as further connections are possible perform the following steps:

(8) Determine $k_0 \in \{1, \ldots, N\}$ with

$$f_{y^{k_0}}(s_{k_0}, t_{k_0}) = \min\{f_{y^k}(s_k, t_k) \mid k = 1, \ldots, N\}.$$

(9) Try to connect s_{k_0} with t_{k_0} via a shortest path by taking the above criteria into account.

(10) If the connection via a shortest path is possible, then
    let W be the chosen path;
    set $S_{k_0} := S_{k_0} \cup W$, $T'_{k_0} := T'_{k_0} \setminus \{s_{k_0}\}$ and $c_e := 0$ for all $e \in W$;
    if $T'_{k_0} = \emptyset$, set $f_{y^{k_0}}(s_{k_0}, t_{k_0}) := \infty$;
    else determine another pair (s_{k_0}, t_{k_0}) similar to (6).

(11) Else goto (2);

(12) If all terminal sets are connected, return the feasible solution $(S_1, \ldots, S_N)$.

(13) Otherwise, print the message "No feasible solution found".

(14) STOP.

Let us now define the function $f_{y^k}: V \times V \mapsto \mathbb{R}_+$ for some $k \in \{1, \ldots, N\}$. We give a formal definition first and explain the underlying heuristic idea afterwards. For the ease of exposition let the nodes be numbered such that $V = \{(i, j) \mid i = 1, \ldots, h,\ j = 1, \ldots, b\}$ and let $V_{l,r,t,d} := \{(i, j) \mid i = l, \ldots, r,\ j = t, \ldots, d\}$ for $l, r \in \{1, \ldots, b\}$, $l < r$ and $t, d \in \{1, \ldots, h\}$, $t < d$. Suppose we want to execute step (5) (resp. (6)) in Algorithm 3.1. Let S_k be the edge set that was already determined for connecting T_k, T'_k the set of not yet connected terminals and $\hat{G}$ the underlying graph.

We consider the case $S_k \neq \emptyset$ (in the case $S_k = \emptyset$ the function f_{y^k} is defined similarly) and let $s_k = (i_s, j_s) \in T'_k$ and $t_k = (i_t, j_t) \in V(S_k)$ be given. Determine $l, r \in \{1, \ldots, b\}$, $l < r$ and $t, d \in \{1, \ldots, h\}$, $t < d$ such that $s_k, V(S_k) \in V_{l,r,t,d}$ and $|V_{l,r,t,d}|$ is minimal. Set $E_{l,r,t,d} := \{e \in \hat{E}(V_{l,r,t,d}) \mid y_e^k > 0\}$ and suppose (V_s, E_s) is the component in $(V_{l,r,t,d}, E_{l,r,t,d})$ with $s_k \in V_s$. Set

$$f_{y^k}(s_k, t_k) := \Big| w(W(s_k, t_k)) - \sum_{e \in E_s} w_e y_e^k \Big|,$$

where $W(s_k, t_k)$ is a shortest path from s_k to t_k in $\hat{G}$ (with respect to w).

The heuristic idea of this function is the following. We determine a graph $(V_{l,r,t,d}, E_{l,r,t,d})$ which is the smallest rectangular grid graph containing both components (often designated as the "minimal enclosing rectangle"). Inside the minimal enclosing rectangle we compute the weighted sum $(= \omega)$ of those edges that are in the same component as s_k, where only edges with $y_e^k > 0$ are considered. The value ω is compared to the length $(= \lambda)$ of a shortest path between the two nodes. If ω is smaller than λ, we assume that the information from y^k is too poor to decide how to connect the two nodes. The smaller the difference, the less information and the greater the value of f. On the other hand, if ω is greater than λ the two nodes will be probably connected via a detour. The greater the difference, the greater the value of f. Thus, we choose the components with value ω close to λ first.

Table 1

Iterations	Perturbed objective function			Original objective function		
	LP value	Pivots	CPU-time	LP value	Pivots	CPU-time
1	0.000	0	0:53	0.000	0	0:52
2	456.562	1163	3:25	456.850	2646	5:35
3	457.571	420	5:07	457.100	4656	19:49
4	457.589	548	7:28	457.350	5995	40:39
5	457.746	800	10:25	457.417	7001	67:37
6	457.793	1224	14:36	457.417	7657	97:55
7	458.014	3175	24:03	457.515	11367	149:21
8	458.314	2007	31:07	457.748	23718	244:11
9	458.625	2554	40:19	458.149	49393	456:26

Obviously all ideas mentioned so far are of heuristic nature and there is no guarantee that we will obtain good results. However, due to many tests we have performed this strategy seems to be reasonable.

3.2. *Implementation details*

In this section we want to focus on some little "tricks" that enter into our cutting plane algorithm. The underlying ideas might appear easy and not very deep to the reader. However, it turns out that these ideas are very effective and indespensible for solving practical problems. We want to illustrate the effects of the ideas on an example called "difficult switchbox" (for the data of this problem see the next section).

Let us mention here that we use the code CPLEX [1] for solving the linear programs that come up. Without such a fast and robust code we would not have been able to solve the given problem instances. The linear programs we encountered appeared to be quite difficult. One of the reasons for this is probably that our linear programs have many alternative optimum solutions and are simultaneously primally and dually highly degenerate.

A frequently used method to overcome such difficulties is to perturb the right hand side of the linear program. Since we are solving the problems with the dual simplex method we must perturb the objective function of the weighted Steiner tree packing problem. After many experiments and discussions with R.E. Bixby (Rice University, Houston, TX) we decided to proceed as follows. Let $\omega \in \mathbb{R}^{\mathcal{N} \times E}$ with $\omega_e^k = w_e$ for all $e \in E$, $k = 1, \ldots, N$ be the original objective function. For each terminal set T_k, we compute a Steiner tree S_k by applying a heuristic procedure and determine random numbers $\varepsilon_e^k \in [0, 1]$. Then we use the objective function vector $\tilde{w} \in \mathbb{R}^{\mathcal{N} \times E}$ defined by

$$
\begin{aligned}
\tilde{w}_e^k &:= \omega_e^k - b\varepsilon_e^k - \eta, && \text{if } e \in S_k, \text{ for } k = 1, \ldots, N; \\
\tilde{w}_e^k &:= \omega_e^k - b\varepsilon_e^k, && \text{if } e \notin S_k, \text{ for } k = 1, \ldots, N,
\end{aligned}
\tag{3.1}
$$

[1] CPLEX is a registered trademark of CPLEX optimization, Inc.

where $\eta = 1/2(n+1)$ and $b = \min\{10^{-5}, 1/2(n+1)\}$ with $n = |E|$ in the actual implementation. It is easy to see that, if the given objective function is integer, an optimal Steiner tree packing with respect to $\tilde{w}$ is also optimal with respect to ω and vice versa.

Table 1 demonstrates the success of the perturbation trick for the "difficult switchbox" routing problem. Column 1 gives the number of cutting plane iterations, column 2, 3 and 4 (resp. 5, 6 and 7) contain the LP objective value, the number of pivots and the accumulated CPU-time by using the perturbed (resp. original) objective function. The numbers are very impressive, in particular if one considers the last rows. The running time is reduced to less than one tenth of the original time.

Another (polyhedral) preprocessing trick helped to increase the lower bounds and to decrease the running time considerably. After "solving" the trivial initial linear program by setting all variables to zero we do not call our general separation routines; rather, we generate a particular class of Steiner cut and Steiner partition inequalities for which we have heuristic reasons to believe that they form a sensible set of "good" initial cutting planes.

Since the underlying graph is a complete rectangular grid graph, we add all Steiner cut inequalities that are induced by a horizontal or vertical cut. The advantage is that these inequalities have pairwise different support. In addition, for multiterminal nets we extend each Steiner cut inequality to a Steiner partition inequality with right hand side greater than two. For example, let $|T_k| = p \geq 3$, $F = \delta(W)$, $W \subset V$ be a vertical cut that induces a Steiner cut inequality. First, we determine a Steiner partition $W_1, \ldots, W_q$ of W such that $[W_i : W_{i+1}]$ is a horizontal cut in $(W, E(W))$ for $i = 1, \ldots, q-1$ and q is maximal. The only node sets of $W_1, \ldots, W_q$ that possibly contain more than one terminal are W_1 and W_q. For these two node sets we again determine a Steiner partition $W_r^1, \ldots, W_r^{l_r}$ for $r = 1$ and $r = q$ such that $[W_r^i : W_r^{i+1}]$ is a vertical cut in $(W_r, E(W_r))$ and l_r is maximal. The same procedure is applied to the node set $V \setminus W$. Taking both together we obtain (after renumbering) a Steiner partition $W_1, \ldots, W_s$ with $s = p$, and $x(\delta(W_1, \ldots, W_s)) \geq p - 1$ defines a Steiner partition inequality. We extend each horizontal and vertical cut that defines a Steiner cut inequality in this way. Obviously, the

Table 2

Iterations	With special Steiner partition inequalities		Without special Steiner partition inequalities	
	LP-value	CPU-time	LP-value	CPU-time
1	0.000	0:52	0.000	0:52
2	456.562	3:24	394.725	3:29
3	457.574	5:38	397.335	6:23
4	457.741	10:21	401.075	17:05
5	457.802	22:39	407.586	43:05
6	458.070	45:53	416.256	67:43
7	458.551	76:13	423.642	103:58
8	458.983	107:28	428.051	158:42
9	459.615	142:39	431.740	203:22

resulting inequalities do not necessarily have different support, but the right hand side is quite large. Let us denote all inequalities constructed this way and the Steiner cut inequalities induced by a horizontal or vertical cut by *special Steiner partition inequalities*.

Table 2 illustrates the progress we obtain by using the special Steiner partition inequalities after solving the initial linear program. Column 1 presents the number of cutting plane iterations. Columns 2 and 3 (resp. 4 and 5) give the LP objective value and the accumulated CPU-time by using (resp. not using) the special Steiner partition inequalities after the first iteration. The results are impressive. The lower bound we obtain within three minutes after the second iteration by adding the special Steiner partition inequalities is much better than after running the algorithm with the separation algorithms for the Steiner partition inequalities discussed in Section 2 for more than 3 h.

Next, we want to deal with the separation of the alternating cycle inequalities. The separation algorithms we have outlined in Section 2 (the dynamic program as well as the heuristics) need a pair of nets as input. The problem we are concerned with is to choose one (or several) "good" pairs of terminal sets for which we want to execute the separation algorithms. If we would call one of these algorithms for all net pairs, we would obtain a non-acceptable running time, because the number of calls is quadratic in the number of nets.

In order to overcome this problem, we try to exploit the information given by the primal heuristic 3.1. Remember that two components are gradually connected in this heuristic. More precisely, in step (9) it is tried to connect two components via a shortest path. If this is not possible, another net must block this path. Obviously the two nets concurrently prefer certain edges in this case. Moreover, this situation indicates that the information provided by the linear programming solution is too poor to decide which of the nets is forced to make a detour. Hence, we conclude that more inequalities combining these nets are necessary. Thus, we call the separation algorithms for the alternating cycle inequalities for nets that are in conflict due to the information of the primal heuristic. Practical experiments have shown that the number of such conflicts is sublinear in the number of nets and that strongly violated alternating cycle inequalities can be obtained for such conflicting net pairs.

We want to point out that not only the linear program solution supplies important information for the primal heuristic. But also conversely, the primal heuristic indicates which type of inequalities are promising for a further execution of the cutting plane algorithm. In our opinion this interplay of the methods for determining the lower and upper bound is essential in order to solve large scale problems.

Let us now summarize the overall algorithm.

Algorithm 3.2 (Branch and cut algorithm for the switchbox routing problem)
Input: A complete rectangular grid graph $G=(V,E)$ with edge capacities $c_e=1$ and edge weights $w_e \in \mathbb{N}_0$, $e \in E$; a net list $\mathcal{N}=\{T_1,\ldots,T_N\}$ where the terminal sets are on the outer face of G.

Output: An optimal solution of the weighted Steiner tree packing problem.
Initialization
(1) Determine the perturbed objective function vector $\tilde{w}$ according to (3.1).
(2) Determine critical cuts by applying the ideas presented in Section 2 and fix the corresponding variables.
(3) Initialize the branching tree with the whole problem.
(4) Solve the following (trivial) linear program

$$\min \tilde{w}^{\mathrm{T}} x$$
$$\sum_{k=1}^{N} x_e^k \le c_e, \quad \text{for all } e \in E,$$
$$x_e^k \ge 0, \quad \text{for all } e \in E,\ k = 1,\ldots,N.$$

(5) Try to determine a feasible solution by applying primal heuristic 3.1.
(6) If a feasible solution was found
set b to the objective function value of the solution.
Else
set $b := \infty$.
(7) Add the special Steiner partition inequalities to the linear program.

Solution and evaluation of the linear program
(8) Determine an optimal solution y of the actual linear program.
(9) If y is the incidence vector of a Steiner tree packing and $\tilde{w}^{\mathrm{T}} y < b$, then
set $b := \tilde{w}^{\mathrm{T}} y$.
(10) Else
try to improve the upper bound b by applying primal heuristic 3.1.
(11) If $\lceil \tilde{w}^{\mathrm{T}} y \rceil = \lceil b \rceil$, then perform the following step:
If there still exists an unsolved subproblem in the branching tree, choose one and goto (8).
Else print the optimal solution corresponding to b, STOP.
(12) Eliminate all inequalities $a^{\mathrm{T}} x \ge \alpha$ with $a^{\mathrm{T}} y < \alpha$ from the actual linear program.

Separation
(13) Determine violated constraints from the "pool' (for an explanation of the *pool* see below) as well as by applying the separation heuristics mentioned in Section 2.
(14) If violated constraints are found, add them to the linear program and goto (8).
(15) Try to find violated Steiner partition inequalities and alternating cycle inequalities by using the dynamic programs.
(16) If violated constraints are found, add them to the linear program and goto (8).

Branching
(17) Determine a variable index (e, k) with $y_e^k \notin \{0, 1\}$.
(18) Generate two subproblems, one by adding the constraint $x_e^k = 0$ and the other by adding the constraint $x_e^k = 1$.
(19) Add both subproblems to the branching tree.
(20) Choose a subproblem from the branching tree and goto (8).

The cutting plane algorithm itself encloses (up to the initialization) steps (8) to (16). We have embedded this method into a general branch and cut framework developed by M. Jünger (Universität Köln). The enumeration scheme is only sketched in steps (4) and (17)–(20). In fact, an efficient implementation of such a scheme is a very difficult and complex task. For more details concerning the branch and cut framework the interested reader is refered to the software package of Jünger.

In step (12) we delete all inequalities (up to the capacity and trivial inequalities) that are not satisfied with equality from the linear program, in order to keep the size of the linear program small. The eliminated constraints are stored in a so-called "pool", which is checked during the separation phase.

If step (17) is executed, we are sure that there exists an index such that $0 < y_e^k < 1$. This is true, because y is not the incidence vector of a Steiner tree packing and in step (13) the Steiner cut inequalities are exactly separated by our separation algorithms. According to (3.2) in [12] the existence of such an index is guaranteed.

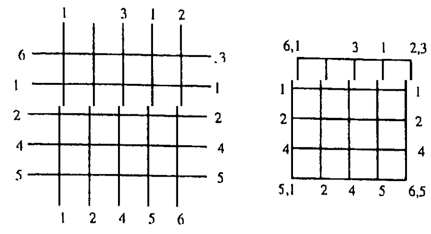
Algorithm 3.2 can be used, in principle, to determine an optimal solution of a given switchbox routing problem or to detect that no feasible solution exists. However, it may not be possible to guarantee this in acceptable time. For that reason we provide an option in our algorithm to limit the running time. If this limit is exceeded, the algorithm stops and prints the best lower and upper bound.

4. Computational results

In this section we report on our computational experiences with the algorithm introduced in Section 3. We have tested our algorithm on switchbox routing problems that are discussed in the literature. Table 3 summarizes the data. Column 1 presents the name used in the literature. In columns 2 and 3 the height and width of the underlying grid graph is given. Column 4 contains the number of nets. Columns 5 to 9 provide information about the distribution of the nets; more precisely, column 5 gives the number of 2-terminal nets, column 6 gives the number of 3-terminal nets and so on. Finally, the last column states the reference to the paper the example is taken from.

Table 3

Name	h	b	N	Distribution of the nets					Reference
				2	3	4	5	6	
Difficult switchbox	15	23	24	15	3	4	1	1	[3]
More difficult switchbox	15	22	24	15	3	5		1	[4]
Terminal intensive switchbox	16	23	24	8	7	5	4		[19]
Dense switchbox	17	15	19	3	11	5			[19]
Augmented dense switchbox	18	16	19	3	11	5			[19]
Modified dense switchbox	17	16	19	3	11	5			[4]
Pedagogical switchbox	16	15	22	14	4	4			[4]



(a) (b)

Fig. 5.

In all examples as they were originally introduced in the literature, the underlying graph is given as follows. The graph is obtained from a complete rectangular grid graph by removing the outer cycle, see Fig. 5(a). Hence, every terminal is incident to a unique edge, and obviously every Steiner tree must contain this edge. It is easy to see that by contracting all pending edges an equivalent problem is obtained, see Fig. 5(b). The graph resulting this way is a complete rectangular grid graph with terminals on the outer face. This instance is the input to our problem.

The first example "difficult switchbox" was introduced by Burstein and Pelavin. The second one "more difficult switchbox" is derived from the first one be deleting the last column. (More precisely, the edges $[(i, 23), (i, 24)]$ of the first grid graph are contracted for $i = 1, \dots, 15$ and parallel edges are deleted.) The net list is the same. The difference in the distribution occurs (see columns 7 and 8), because an edge whose endpoints belong to the same net is contracted. The third problem instance was introduced by Luk, here each outer face node is occupied by a terminal. The fourth switchbox routing problem is again due to Luk. Up to now it is not known whether a solution for this example exists, if the Manhattan or 2-layer model is used. Based on this example two variants can be obtained. One, called "augmented dense switchbox", has an additional column on the right, the other, called "modified dense switchbox", has an additional column near the middle and an additional row on the buttom. The last example was introduced by Cohoon and Heck [6]. They illustrated their algorithm on this problem.

In all examples the edge weights as well as the edge capacities are equal to one. Unfortunately, the problem instances do not fix the routing model (Manhattan, knock-knee or multiple layer model). To our knowledge all methods from the literature use the Manhattan model or 2-layer model. The choice of the underlying model strongly influences the solvability of the problems. For example, there may exist a solution in the 2-layer model, whereas it does not in the knock-knee model. Fig. 5 illustrates such an example (this example is taken from [6]). Moreover, there exist problem instances where shorter connections are possible in the 2-layer model than in the knock-knee model. The

Table 4

Example	Variables	Fixed variables	Remaining variables
Difficult switchbox	15648	2224	13424
More difficult switchbox	14952	2450	12502
Terminal intensive switchbox	16728	4913	11815
Dense switchbox	9082	4831	4251
Augmented dense switchbox	10298	2678	7620
Modified dense switchbox	9709	4057	5652
Pedagogical switchbox	9878	2039	7839

same is true for a comparison of the knock-knee model with the Manhattan model. Thus, a comparison of algorithms for the different models is not possible. So we confine ourselves to report on the results we have obtained by applying our algorithm.

Table 4 informs about the size of the problems and about the success of fixing variables with the algorithm discussed in the last subsection of Section 2. Column 2 states the total number of 0/1 variables, column 3 gives the number of fixed variables and the last column contains the number of remaining variables. Table 4 illustrates that many variables can be fixed, for example more than one half of the variables for problem "dense switchbox". Nevertheless, the number of remaining variables is still large (see the last column).

In Table 5 the results we have obtained with our branch and cut algorithm are summarized. Column 2 gives the best feasible solution. The values are not integer due to the perturbed objective function. To obtain the real value with respect to the original objective function the entries must be rounded up. The entries in column 3 are the objective function values of the linear program when no further violated constraints are found, i.e., when branching (steps (17)–(20) in Algorithm 3.2) is performed for the first time. This values are obviously lower bounds for the whole problem. In column 4 the

Table 5

Example	Best solution	LP value	Gap	Iterations	B&C	CPU-time
Difficult switchbox	463.711	463.709	0.0%	69	3	1564:15
More difficult switchbox	451.712	451.708	0.0%	53	1	983:23
Terminal intensive switchbox	536.694	535.196	0.2%	163	13	3755:44
Dense switchbox *	440.601	437.579	0.7%	119	4	1017:43
Augmented dense switchbox *	468.600	466.006	0.4%	105	1	4561:41
Modified dense switchbox	451.585	451.009	0.0%	51	1	387:03
Pedagogical switchbox	330.770	330.760	0.0%	77	5	251:58

percentage deviation of the best solution from the lower bound is given; more precisely, column 4 contains the value $(\lceil \tilde{w}_2 \rceil - \lceil \tilde{w}_3 \rceil)/\lceil \tilde{w}_3 \rceil$, where $\tilde{w}_2$ (resp. $\tilde{w}_3$) is the corresponding value of column 2 (resp. 3). Column 5 (resp. 6) gives the number of cutting plane iterations (resp. the number of nodes in the branching tree). Finally, the last column reports on the running times. The values are stated in minutes obtained on a SUN 4/50. The two examples "dense switchbox" and "augmented dense switchbox" marked with a star are stopped after the time given in the last column, because no further process could be achieved. We claim that the values given in column 2 are optimal, but we are yet not able to prove this with the cutting plane algorithm. All other problem instances are solved to optimality.

The numbers in Table 5 are quite encouraging. For all problem instances the lower bound in column 3 guarantees that the best feasible solution deviates at most 0.7% from the optimal solution. In our opinion the main advantage of our algorithm is that the quality of an heuristically determined solution can be evaluated with the lower bound. Especially, for problem instances arising in VLSI-Design, where in general only heuristics are at hand, a cutting plane algorithm helps in analyzing the heuristics and simultanously delivers a lot of knowledge about the problem itself.

Nevertheless, one major problem with our algorithm is its running time. The numbers in the last column of Table 5 are very high. One reason is that we are interested in an optimal solution or at least in the best lower and upper bound for each of the problems that we can achieve with our approach. This is time consuming. In practice, heuristics usually find feasible solutions for these instances in a few seconds. These running times are certainly not reachable with our algorithm. However, the main advantage of the cutting plane approach is to give a solution guarantee for the best known feasible solution. We are not aware of any method used in practice that is able to guarantee a certain quality of the feasible solutions found. From this point of view, we have analyzed our results also. Table 6 presents the time (measured in minutes), after which the lower bound deviates at most 5, 2, 1 and, if obtained, 0% from the best feasible solution.

It can be seen from column 2 that, for all problem instances, the lower bound deviates from the best feasible solution by at most 5% after no more than 6 min. Table 6

Table 6

Example	5%	2%	1%	0%
Difficult switchbox	3:24	3:24	90:12	688:49
More difficult switchbox	3:20	3:20	38:19	530:11
Terminal intensive switchbox	5:44	83:24	239:10	–
Dense switchbox *	2:00	2:00	103:07	–
Augmented dense switchbox *	2:04	2:04	269:20	–
Modified dense switchbox	2:04	2:04	2:04	387:03
Pedagogical switchbox	1:46	2:27	15:04	117:55

illustrates in addition that the amount of time increases strongly to obtain a quality below 1%.

In our opinion the times in column 1 of Table 6 are acceptable. However, we would like to point out that these examples are quite small in comparison to problem sizes arising in other practical applications for the design of electronic circuits. Our long-term goal is to apply the branch and cut algorithm to instances of larger scale, too. In order to achieve this, we surely must reduce the running times. We have analyzed our algorithm concerning the question where most of the time is spent. It turns out that about 90% of the time is used to solve the linear programs. To our present knowledge two possibilities arise to overcome this problem.

(1) *Reducing the number of variables*, we consider the problem only on a subset of the set of variables, solve the problem on this subset and check whether this solution is also optimal for the whole problem. If not, we add some variables and solve the extended problem again. This method is commonly used to solve large scale practical problems by a cutting plane algorithm (see, for instance, [9,21]).

(2) *Decompose the linear programs*, the constraint matrices of our problems are of very special structure. Due to this structure it seems to be promising to decompose the linear program. Methods for decomposing linear programs were suggested by Dantzig and Wolfe [5] or by Benders [1]. Up to now these methods are not used in practice, because the problems can be solved faster directly. However, with the help of parallel computers these methods may get competitive, especially for our problem instances.

5. Conclusion

In this paper we have developed a cutting plane algorithm for the Steiner tree packing problem. We have introduced some separation methods for special problem instances where the underlying graph is planar and all terminal sets lie on the outer face of the graph. This special instances include an important subproblem in VLSI-Design, the so-called switchbox routing problem. We have reported on computational results we have obtained with our branch and cut algorithm for this type of problems. The results are encouraging. Most of the problems discussed in the literature are solved to optimality. Thus, we have good hopes that this approach may also be applicable to large scale problem instances as they occur in practice. To achieve this long term goal there surely remain a lot of problems to be solved.

References

[1] J.F. Benders, "Partitioning procedures for solving mixed variables programming problems," *Numerische Mathematik* 4 (1962) 238–252.
[2] M.L. Brady and D.J. Brown, "VLSI routing: Four layers suffice," in: F.P. Preparata, ed., *Advances in Computing Research, Vol. 2: VLSI Theory*, (Jai Press, London, 1984) pp. 245–258.
[3] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Transactions on Computer Aided-Design* CAD-2 (1983) 223–234.

[4] J.P. Cohoon and P.L. Heck, "BEAVER: A computational-geometry-based tool for switchbox routing," *IEEE Transactions on Computer-Aided-Design* CAD-7 (1988) 684–697.
[5] G.B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research* 8 (1960) 101–111.
[6] S.E.Dreyfus and R.A. Wagner, "The Steiner problem in graphs," *Networks* 1 (1971) 195–207.
[7] R.E. Erickson, C.L. Monma and A.F. Veinott, "Send-and-split method for minimum concave-cost network flows," *Mathematics of Operations Research* 12 (1987) 634–664.
[8] M.R. Garey and D.S. Johnson, "The rectilinear Steiner tree problem is $\mathcal{NP}$-complete," *SIAM Journal on Applied Mathematics* 32 (1977) 826–834.
[9] M. Grötschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems," *Mathematical Programming* 51 (1991) 141–202.
[10] M. Grötschel, A. Martin and R. Weismantel, "Routing in Grid Graphs by Cutting Planes," *Zeitschrift für Operations Research* 41 (1995) 255–275.
[11] M. Grötschel, A. Martin and R. Weismantel, "Packing Steiner trees: separation algorithms," *SIAM Journal on Discrete Mathematics*, to appear.
[12] M. Grötschel, A. Martin and R. Weismantel, "Packing Steiner trees: polyhedral investigations," Mathematical Programming 72 (1996) (this issue).
[13] M. Grötschel and C.L. Monma, "Integer polyhedra associated with certain network design problems with connectivity constraints," *SIAM Journal on Discrete Mathematics* 3 (1990) 502–523.
[14] M. Grötschel, C.L. Monma and M. Stoer, "Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints," *Operations Research* 40 (1992) 309–330.
[15] R.M. Karp, "Reducibility among combinational problems," in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations*) (Plenum, New York, 1972) pp. 85–103.
[16] M.R. Kramer and J. van Leeuwen, "The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits," in: F.P. Preparata, ed., *Advances in Computing Research, Vol. 2: VSLI Theory* (Jai Press, London, 1984) pp. 129–146.
[17] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout* (Wiley, Chichester, 1990).
[18] W. Lipski, "On the structure of three-layer wireable layouts," in: F.P. Preparata, ed., *Advances in Computing Research, Vol. 2: VLSI Theory* (Jai Press, London, 1984) pp. 231–244.
[19] W.K. Luk, "A greedy switch-box router," *Integration* 3 (1985) 129–149.
[20] A. Martin, "Packen von Steinerbäumen: Polyedrische Studien und Anwendung," Ph.D. Thesis, Technische Universität Berlin,(1992).
[21] 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.
[22] M .Sarrafzadeh ,"Channel routing problem in the knock-knee mode is $\mathcal{NP}$-complete," *IEEE Transactions on Computer-Aided-Design* CAD-6 (1987) 503–506.