A branch-and-price algorithm for the Steiner tree packing problem

Gue-woong Jeong\textsuperscript{a}, Kyungsik Lee\textsuperscript{b}, Sungsoo Park\textsuperscript{a,}\textsuperscript{*}, Kyungchul Park\textsuperscript{c}

\textsuperscript{a}Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, 373-1 Gusong-dong, Yusong-gu, Taejon 305-701, South Korea
\textsuperscript{b}Electronics and Telecommunications Research Institute, 161 Kajong-dong, Yusong-gu, Taejon 303-350, South Korea
\textsuperscript{c}Telecommunication Network Lab., Korea Telecom, 463-1 Junmin-dong, Yusong-gu, Taejon 305-390, South Korea

Received 1 February 1999; received in revised form 1 January 2000

Abstract

This paper deals with the Steiner tree packing problem. For a given undirected graph \( G = (V, E) \) with positive integer capacities and non-negative weights on its edges, and a list of node sets (nets), the problem is to find a connection of nets which satisfies the edge capacity limits and minimizes the total weights. We focus on the switchbox routing problem in knock-knee model and formulate this problem as an integer programming using Steiner tree variables. We develop a branch-and-price algorithm. The algorithm is applied on some standard test instances and we compare the performances with the results using cutting-plane approach. Computational results show that our algorithm is competitive to the cutting plane algorithm presented by Grötschel et al. and can be used to solve practically sized problems.

Scope and purpose

VLSI circuits becomes more complex with the need of miniaturization and sophisticated functions of electronic device. That makes designing a VLSI more difficult. Due to these difficulties, it is common way to decompose it into several sub-problems. One of the sub-problems is the detailed routing problem, where exact path of wires is determined while minimizing some objective (for examples, total wire length) under some design rules. Among many kinds of detailed routing problem, we focus on the switchbox routing problem in knock-knee routing model. For this problem, we developed an integer programming model and solved it successfully for some test problems. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Steiner tree packing problem (STP); VLSI design; Branch-and-price; Column generation

*Corresponding author. Tel.: + 82-42-869-3121; fax: + 82-42-869-3110.
E-mail address: sspark@cais.kaist.ac.kr (S. Park).

0305-0548/01/$ - see front matter © 2001 Elsevier Science Ltd. All rights reserved.
PII: S 0 3 0 5 - 0 5 4 8 ( 0 0 ) 0 0 0 6 6 - 6
1. Introduction

Very large-scale integration (VLSI) refers to a technology through which it is possible to implement large circuits in silicon. The VLSI technology has been successfully used to build microprocessors, signal processors, large capacity memories, memory controllers and interconnection networks [1]. Since a VLSI circuit is usually composed of numerous transistors, designing a VLSI circuit is a very complex and difficult task. To overcome the difficulties, a VLSI circuit is usually designed through two phases, i.e. logical design and physical design. In the logical design phase, it is specified which logical units (called cells) will be used and it is determined which of the specified logical units should be connected by wires [2]. A cell has contact points, called terminals, through which cells are connected by wire. A set of terminals, which must be connected, is called a net. The list of cells and the list of nets are determined in the logical design phase and provided as an input to the physical design phase. In the physical design phase, it is determined where the cells are located in silicon and how the nets are connected while minimizing an objective function under some design rules. Most physical design problems are NP-hard and moreover, in the real world application, there are so many cells and nets that we cannot solve it to optimality. Therefore, in order to reduce this involved complexity, the physical design problem is usually decomposed into several subproblems. First, we find the appropriate locations of cells (placement problem). Subsequently, we determine approximate course of wires to connect nets (global routing problem). After global routing, exact path of the wires is determined without changing the layout area (detailed routing). Finally, if needed, compaction is performed to reduce the layout area. The above process is repeated if the resulting solution is not acceptable.

Here, we consider a problem arising in the detailed routing. The detailed routing problem is classified according to two criteria (routing graph, routing model). In the routing graph, there are channel routing graph, switchbox routing graph, and general routing graph. In the routing model, there are planar routing model, knock-knee routing model, Manhattan routing model, multilayer model, etc. [3]. The detailed routing problem considered in this paper is the switchbox routing problem in the knock-knee routing model (SBKP). We give a brief explanation of the switchbox routing graph and knock-knee routing model. The switchbox routing graph is a complete rectangular grid graph and the terminal sets are located on the four sides of the grid [2]. An example of the switchbox routing model is shown in Fig. 1. The numbers on the four sides denote net number and terminals with the same net number must be connected. Since the size of the routing graph (layout area) is fixed, we have to optimize the total wire length. We may be able to reduce the layout area in the compaction process with the detailed routing solution. In the knock-knee routing model, we find an edge disjoint connection of nets. Note that since the routings of nets need not be node disjoint, knock-knees are allowed (see Fig. 2). Since wires can cross (knock-knee is possible), this model has high routability relative to the other routing models. But, the assignment of the routing to layers is not considered in this model. If knock-knee is allowed, then at least two layers are needed. Brady and Brown [4] have designed an algorithm that routes any solution in this model on four layers. Lipski [5] proved that the problem of deciding whether a knock-knee routing needs only three layers is NP-complete. For more detailed explanations of routing graphs and models, refer to Burstein [6], Lengauer [3] and Sait and Youssef [1].

Now, let us illustrate an example (see Fig. 3). The numbered nodes denote terminals and the nodes with the same number (called a net) should be connected. We must find the edge-disjoint
Before defining our problem, we introduce a well-known terminology, *Steiner tree* in graph theory. Given an undirected graph \( G = (V, E) \) and a node set \( N \subseteq V \), if an edge set \( S \subseteq E \) connects all nodes in \( N \), we call this edge set \( S \) a *Steiner tree* for \( N \). The *Steiner tree packing* (STP) was described by Grötschel et al. [2] as follows. Given an undirected graph \( G = (V, E) \) with edge capacities \( c_e \in \mathbb{Z}_+ \) for all \( e \in E \) and a list of node set \( N = \{N_1, \ldots, N_n\} \) with \( N_k \subseteq V, k = 1, \ldots, n \), find Steiner trees \( S_k \) for \( N_k, k = 1, \ldots, n \) such that each edge \( e \in E \) is contained in atmost \( c_e \) of the edge set \( S_1, \ldots, S_n \). Every collection of Steiner trees \( S_1, \ldots, S_n \) with this property is called a Steiner tree packing. If weights of the edges are given in addition and (with respect to these weights) minimal Steiner tree packing must be found, we call this the weighted Steiner tree packing problem. Note that this description is more general than our problem (SBKP). Many detailed routing problems can be modeled as the Steiner tree packing problem. Especially, the switchbox routing connection of nets. In Fig. 3, an example of the connection is shown as solid lines. Dotted lines denote the edges that are not used. Note that each edge can be used at the most once.
The problem in the knock-knee routing model can be modeled as the Steiner tree packing problem with the following additional restrictions.

(i) The underlying graph $G = (V, E)$ is a complete rectangular grid graph,
(ii) The edge capacities $c_e = 1$ for all $e \in E$,
(iii) The edge weights $w_e = 1$ for all $e \in E$.

Considering that finding a Steiner tree in a graph is NP-hard, it is not strange that STP is NP-hard (see, for example, Garey and Johnson [7], Karp [8] and Kramer and Leeuwen [9]). And even the switchbox routing in knock-knee model (SBKP) is also NP-hard [10]. Due to the inherent difficulties to solve the detailed routing problem, many researchers have developed various heuristics, but the performance of these heuristics could not be evaluated exactly because the optimal solutions could not be found. However, Grötschel et al. [2] developed a cutting plane algorithm based on polyhedral theory, which solved SBKP to near optimal. Their work gives not only a good algorithm, but also the criteria for evaluating quality of some heuristics. But, there is a disadvantage that it takes a lot of time to solve the problem. This fact motivates us to find a different approach. So, we developed an integer programming model, which is appropriate for column generation.

Fig. 3. Example of SBKP.
The rest of this paper is organized as follows. In Section 2, we give two mathematical formulations of the problem and explain the motivation of our approach. In Section 3, overview of our algorithm is described briefly. In Section 4, we explain the column generation procedure in detail. In Section 5, two branching rules applied in the branch-and-bound procedure are presented. In Section 6, we explain the column generation procedure in detail. In Section 7, we address more technical details in implementing our algorithm. In Section 8, we present computational results of our algorithm on the standard test problems. Section 9 concludes this paper.

2. Problem formulations

In this section, we give two mathematical formulations and explain the motivation of our approach in view of the bounds provided by the LP relaxations.

SBKP can be stated as follows.

**Instance:** A graph $G = (V, E)$ and a list of node sets (called nets) $N = \{N_1, \ldots, N_n\}$

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

(i) A subgraph $G_k = (V(S_k), S_k)$ is a Steiner tree with set of terminals $N_k \subseteq V$, for all $k = 1, \ldots, n$,

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

(iii) $\sum_{k=1}^{n} |S_k|$ is minimal,

where $V(S_k)$ is the set of nodes that are incident to the edges in $S_k$. Based on the integer programming model suggested by Grötschel et al. [11], consider the following formulation.

**SBKP1:**

$$\min \sum_{k=1}^{n} \sum_{e \in E} x_e^k$$

s.t.

$$\sum_{e \in \delta(W)} x_e^k \geq 1 \quad \text{for all } W \subset V, W \cap N_k \neq \emptyset, (V \setminus W) \cap N_k \neq \emptyset, k = 1, \ldots, n,$$

(1)

$$\sum_{k=1}^{n} x_e^k \leq 1 \quad \text{for all } e \in E,$$

(2)

$$x_e^k \in \{0, 1\} \quad \text{for all } e \in E \text{ and } k = 1, \ldots, n,$$

(3)

where $\delta(W)$ denotes the set of edges with one end node in $W$ and the other in $V \setminus W$. The decision variable $x_e^k$ is a 0–1 variable where $x_e^k = 1$ if and only if edge $e$ is contained in the Steiner tree for net $k$. Constraints (1) are called the Steiner cut inequalities and originally proposed for the Steiner tree problem on graph. Constraints (2) mean that each edge $e \in E$ can be contained in at most one of the edge sets $S_1, \ldots, S_n$ Grötschel et al. [2] solved this problem using some cutting planes such as Steiner partition inequalities, alternating cycle inequalities, and critical cut inequalities. For more details, refer to Grötschel et al. [11,2].
Generally the Steiner tree problem is known to be \textit{NP-hard} \cite{7,8}. But, since in our problem a given graph $G$ is planar and all terminals are on the outer face of $G$, the Steiner tree problem can be solved in polynomial time \cite{23}. When all terminals are not on the outer face, the Steiner tree problem becomes \textit{NP-hard}. However, if the number of terminals in each net is small, the \textit{NP-hard} Steiner tree problem can be solved efficiently, for example, by a dynamic programming method with complexity $O(3^t v + 2^v \log v + v^2)$, where $t$ is the number of terminals in the net and $v$ is the number of nodes in the graph \cite{12}. This fact motivates us to use the column generation approach to solve SBKP with the following formulation.

**SBKP2:**

$$\min \sum_{k=1}^{n} \sum_{i \in T_k} |S_i| y_i$$

s.t.

$$\sum_{i \in T_k} y_i = 1 \quad \text{for all } k = 1, \ldots, n,$$  

(4)

$$\sum_{k=1}^{n} \sum_{i \in T_k(e)} y_i \leq 1 \quad \text{for all } e \in E,$$  

(5)

$$y_i \in \{0, 1\} \quad \text{for all } i \in T_k \quad \text{and } \quad k = 1, \ldots, n,$$  

(6)

where $S_i$ is the set of edges contained in Steiner tree $i$, $T_k$ is the set of all Steiner trees for net $k$ and $T_k(e)$ is the set of Steiner trees for net $k$ containing edge $e$. Decision variable $y_i$ is a 0–1 variable where $y_i = 1$ if and only if Steiner tree $i$ is chosen. Constraints (4) mean that only one Steiner tree for each net must be chosen. Constraints (5) have the same meaning as constraints (2). The main drawback of this kind of formulation (SBKP2) is the number of variables involved. The number of variables for each net is in general exponential in the size of the input. But, the linear programming relaxation of this formulation (SBKP2) gives a tight lower bound. The linear programming relaxations of the formulations can be obtained by replacing constraints (3) and (6) with non-negativity constraints, respectively. For convenience, we denote the LP relaxation of SBKP1 and that of SBKP2 by LSBKP1 and LSBKP2, respectively. Let $ST_k$ and $\text{conv}(ST_k)$ be the set of binary vectors each of which represents a Steiner tree for net $k$ and the convex hull of $ST_k$, respectively. Consider the following linear program.

**LP1:**

$$\min \sum_{k=1}^{n} \sum_{e \in E} x^k_e$$

s.t.  

(2) and $x^k \in \text{conv}(ST_k)$ for all $k = 1, \ldots, n$.

**Proposition 1.** The LP relaxation of SBKP2 (LSBK2) and LP1 have the same optimal value.

**Proof.** Suppose that $\bar{x} = (\bar{x}^1, \bar{x}^2, \ldots, \bar{x}^n)$ is a feasible solution to LP1, where each $\bar{x}^k$ is a $|E|$-component vector for net $k$. Then each vector $\bar{x}^k$ in $\bar{x}$ can be expressed as a convex combination of
some extreme points of \( \text{conv}(S^1) \). That is,

\[
\bar{x}^k = \sum_{i=1}^{m_k} \bar{\lambda}_k^i q_k^i \quad \text{for all } k = 1, \ldots, n,
\]

where \( \bar{\lambda}_k^i \geq 0, \sum_{i=1}^{m_k} \bar{\lambda}_k^i = 1, \) and \( q_k^1, \ldots, q_k^m \) are extreme points of \( \text{conv}(S^1) \). Since \( q_k^1, \ldots, q_k^m \) are extreme points of \( \text{conv}(S^1) \), each \( q_k^i \) represents a Steiner tree for net \( k \). Let \( y_k \) be the variable in LSBKP2 corresponding to the Steiner tree represented by \( q_k^i \). To obtain a feasible solution, we simply assign the value \( \bar{\lambda}_k^i \) to \( y_k \), \( i = 1, \ldots, m_k \) and \( k = 1, \ldots, n \) and set the others of \( y_k \)'s to zero. Then the obtained solution is feasible to LSBKP2.

Conversely, from a given feasible solution to LSBKP2 a feasible solution to LP1 can be obtained similarly. Hence LSBKP2 and LP1 have the same objective value.

Let \( Z_{LP1}, Z_{LP2} \) and \( Z_{LP3} \) denote the optimal objective value of LP1, LSBKP1, and LSBKP2, respectively. Since every feasible solution of LP1 is also a feasible solution to LSBKP1, \( Z_{LP1} \leq Z_{LP2} \). Therefore, LSBKP2 gives a lower bound as tight as the one provided by LSBKP1 since \( Z_{LP1} \leq Z_{LP2} = Z_{LP1} \). In view of the bounds of the LP relaxations, SBKP2 is more attractive than SBKP1.

### 3. Overview of our algorithm

In this section, we give a brief explanation of our algorithm to solve the switchbox routing problem in knock-knee routing model (SBKP). Since our formulation (SBKP2) has exponentially many variables (columns), we use a branch-and-price approach, which is a generalization of branch-and-bound with LP relaxations, allowing column generation to be applied throughout the branch-and-bound tree [13]. This approach has been applied successfully to solve many large-scale mixed integer programming problems (for instance, see Johnson et al. [14] and Savelsbergh [15]). In this approach, only part of the column is included in the LP relaxation in order to handle the LP relaxation efficiently. Moreover, most of the other columns will be zero in an optimal solution. To check the optimality of an LP solution, a subproblem, called the column generation problem or the pricing problem is solved to find columns to enter the basis. The column generation problem is a separation problem for the dual LP. If such columns are found, the LP is reoptimized with a subset of those columns added. This process continues until no such columns are found. Then if the final solution is not integral, branching occurs. The column generation is performed similarly throughout the branch-and-bound tree. For a comprehensive survey of this approach, refer to Barnhart et al. [13]. The pure branch-and-price approach can be strengthened by simultaneously applying the cutting plane approach. For example, Park et al. [16] could solve the bandwidth packing problem very efficiently by such an approach.

We start our algorithm by reducing the problem size using critical cut that is defined in Grötschel et al. [2]. In this step, for each net we identify the edges that cannot be used to remove the edges from the underlying graph. Next, we construct the initial linear program using solutions obtained by a heuristic given in the literature [17–19]. Then we solve the LP relaxation of SBKP2 (LSBKP2) to optimality at node 0 in the enumeration tree using the column generation method.
The column generation method is for finding the columns to enter the basis. If there exist columns to enter the basis, we add the columns found into the LP relaxation and reoptimize it. Whenever the LP relaxation, LSBKP2 is reoptimized after some columns are added, the primal-heuristic is performed to obtain a good feasible solution which provides an upper bound of the optimal solution of SBKP2. Otherwise, i.e., if there are no columns to enter the basis, the LP solution is optimal. If the optimal solution of LSBKP2 is integral or the primal-heuristic solution is equal to \( \lceil \text{lpvalue} \rceil \) where lpvalue is the optimal value of LSBKP2, then stop — we obtain the optimal solution of SBKP2. Otherwise, we use branch-and-price procedure which is a generalization of branch-and-bound with LP relaxations, allowing column generation to be applied throughout the branch-and-bound tree. Best bound rule introduced in Nemhauser and Wolsey [20] is used for branching node selection. The overall procedure of the algorithm is presented in Fig. 4. In the following sections, we explain the details of the algorithm.
4. Column generation procedure

As stated in Section 3, only part of the columns (variables) is contained in the initial LP relaxation. The restricted LP relaxation of SBKP2 is denoted by LSBKP2. To check the optimality of the solution of LSBKP2, a column generation problem is solved to find columns to enter the basis. Suppose that we are given an LSBKP2. Let \( \lambda, \pi \) be the dual variable vector for constraints (4) and (5) in LSBKP2, respectively. Then the optimality condition is

\[
\dot{\lambda}_k + \sum_{e \in S_k} \pi_e \leq |S_k| \quad \text{for all} \; i \in T_k \quad \text{and} \quad \pi_e \leq 0 \quad \text{for all} \; e \in E \quad \text{and} \quad k = 1, \ldots, n. \tag{7}
\]

The optimality condition (7) is equivalent to \( \sum_{e \in S_k} (1 - \pi_e) \geq \lambda_k \), for all \( i \in T_k \) and \( \pi_e \leq 0 \), for all \( e \in E \) and \( k = 1, \ldots, n \). Hence, the column generation problem can be stated as follows.

For all \( k = 1, \ldots, n \),

\[
\text{CGP:} \quad \min \sum_{e \in E} (1 - \pi_e)z_e \quad \text{s.t.} \quad z \in S_k^{[E]},
\]

where \( S_k^{[E]} \) denotes the set of 0–1 vectors each of which represents a Steiner tree for net \( k \). The column generation problem (CGP) is to find a minimal Steiner tree for each net \( k \) in the graph \( G \) with edge weight \( w_e = 1 - \pi_e \), for all \( e \in E \). If the value of a minimal Steiner tree for net \( k \) is less than \( \lambda_k \), the corresponding column can be entered into the basis. Otherwise, the current solution is optimal with respect to net \( k \). Many heuristics and some optimal algorithms for the Steiner tree problem have been developed. Some useful surveys of the Steiner problems are available in Hwang and Richards [21] and Winter [22]. Fortunately, there exist some optimal algorithms for the Steiner problem in planar graph. Dreyfus and Wagner [12] developed a dynamic programming algorithm with time complexity \( O(3^v + 2^v \log v + v^2) \). Note that it is an exponential time algorithm. Erickson et al. [23] improved this algorithm for the case that all terminals are on the outer face of a graph. The time complexity of the algorithm of Erickson et al. [23] is \( O(vt^3 + (v \log v)t^2) \). Note that it is a polynomial time algorithm.

We use the above two algorithms to solve CGP. At node 0 in the branch-and-bound tree, since all terminals are on the outer face of a graph, we solve CGP by the algorithm of Erickson et al. [23] in polynomial time. But, when the optimal solution of LSBKP2 does not satisfy integrality, we apply branch-and-price procedure to obtain the integer optimal solution. The key to develop a branch-and-price procedure is identifying a branching rule that eliminates the current fractional solution without making the column generation problem intractable after branching. We have developed two branching rules. One of them, branching rule 1 is to separate the solution space into two subspaces which are obtained by forcing an edge \( e \) to be in the Steiner tree for net \( k \) or not. Suppose that we are in the branch in which edge \( e \) is used for net \( k \) and both the end nodes of the edge \( e \) are not on the outerface. Then the two end nodes are included in the set of terminal nodes. Then, since all terminals are not on the outer face of \( G \), CGP becomes \( NP \)-hard. Hence, we cannot help applying the algorithm of Dreyfus and Wagner [12], which is an exponential time algorithm. The other rule, branching rule 2 is to separate the solution space into two subspaces only by specifying the nets that do not use an edge \( e \in E \). In this case, for the specified nets, we simply delete
the corresponding edges in the graph. Then, we can solve CGP by the algorithm of Erickson et al. [23] in polynomial time since all terminals are still on the outerface of a graph. In the next section, detailed explanations for the branching rules are given.

5. Branching rules

The conventional branching rule based on variable dichotomy is inappropriate to a branch-and-price approach. Comprehensive explanation can be found in Savelsbergh [15]. For our problem a similar explanation as in Savelsbergh [15] is given as follows. Suppose that we branch on a fractional variable \( y_i \) and we solve the LSBKP2 in the branch where the variable \( y_i \) is fixed to zero. Then, it is possible (and quite likely) that the optimal solution of column generation problem will be the same Steiner tree represented by the variable \( y_i \). In this case, we must find the Steiner tree having the second lowest weight. In worst case, the depth \( k \) in the branch-and-bound tree, we have to find the Steiner tree having the \( k \)th lowest weight. Therefore, we need a branching rule that is compatible with the column generation problem. That is, we must be able to modify the column generation problem so that the columns that are infeasible to the branching constraints will not be generated [15]. As stated in the previous section, we have developed two branching rules. Branching rule 1 is very intuitive, but causes some changes in the column generation problem, which makes it more difficult. Branching rule 2 does not cause any change, but is not as intuitive as branching rule 1.

5.1. Branching rule 1

Consider a binary variable \( x_e^k \). It denotes whether edge \( e \) is used for net \( k \) or not. The binary variable \( x_e^k \) can be expressed by binary variables \( y_i \)'s as follows:

\[
x_e^k = \sum_{i \in I_k(e)} y_i \quad \text{for all } e \in E \quad \text{and } k = 1, \ldots, n.
\]

It is clear that if \( x_e^k \) is binary, then the corresponding \( y_i \)'s are also binary. Hence, we can use a branching rule with respect to the binary variables \( x_e^k \)'s instead of one with the \( y \) variables. If \( x_e^k \) is fixed to zero, we simply delete edge \( e \) from the graph when we generate a Steiner tree for net \( k \) not containing edge \( e \). If \( x_e^k \) is fixed to one, we must find a Steiner tree for net \( k \) which contains edge \( e \). To do this, we set the edge weight \( w_e \) to zero and declare the two end nodes of edge \( e \) as terminal nodes. The following proposition guarantees that in case \( x_e^k = 1 \) the Steiner tree generated contains edge \( e \).

**Proposition 2.** For a given graph \( G = (V, E) \), an acyclic edge set \( E' = \{(u, v) \in E \mid w_{uv} = 0\} \) and a node set \( N' = \{v \in V \mid \text{there exists a node } u \in \delta(v) \text{ such that } w_{uv} = 0\} \), an optimal Steiner tree for a terminal node set \( N( \supset N') \) must contain all the edges in \( E' \).

**Proof.** Suppose that an optimal tree \( T^* \) does not contain all the edge in \( E' \). Note that in the column generation problem every edge weight, \( w_e (= 1 - \pi_e) \) is positive except the edges in \( E' \) since the dual variable, \( \pi_e \leq 0 \) for all \( e \in E \). We will prove using induction on \( |E'| \).
(i) For the case $|E'| = 1$, add the edge $uv$ to $T^*$. Then, there exists only one cycle in the new graph $T^* + uv$ and let the cycle be $(uv, e_1, e_2, \ldots, e_k)$. Remove an arbitrary edge $e_i, 1 \leq i \leq k$ from $T^* + uv$. Then, the resulting graph $T_{new}$ is also a Steiner tree for $N$. Let the total weight of a Steiner tree $T$ be $W(T)$. Then, $W(T_{new}) - W(T^*) = w_{uv} - w_e < 0$. This contradicts that $T^*$ is an optimal Steiner tree.

(ii) Assume that for the case $|E'| \leq m$, the optimal Steiner tree for $N$ contains all the edges in $E'$ and consider the case when $|E'| = m + 1$. Let $E'_1$ be the set of edges that are in $E'$ but not in $T^*$. For each edge $w \in E'_1$, add the edge $wv$ to $T^*$ and set $E'_1 = E'_1 \setminus \{uv\}$. Then, there exists only one cycle in the new graph $T^* + wv$. Let the cycle be $(wv, e_1, e_2, \ldots, e_k)$. At least one of edges $e_i, 1 \leq i \leq k$ has positive weight. Otherwise, all the edges in the cycle have zero weight. This implies that there exists an optimal Steiner tree not containing all the edges in $E'$ when $|E'| = k (\leq m)$. Now, remove an edge having positive weight in the cycle. Repeat the above process until $E'_1 = \emptyset$. Let $E_2$ be the set of removed edges. Then, the resulting graph $T_{new}$ is also a Steiner tree for $N$ and it is clear that $W(T_{new}) - W(T^*) = 0$. This contradicts that $T^*$ is an optimal Steiner tree.

Branching rule 1 is easy to understand, but makes the column generation problem more difficult. In case $x^k_e = 0$, we have no trouble in using the polynomial time algorithm of Erickson et al. [23]. However, in case $x^k_e = 1$, since newly added terminal nodes may not be on the outerface of $G$, there does not exist any polynomial time algorithm. Therefore, we cannot help using an exponential time algorithm of Dreyfus and Wagner [12]. But, the main drawback of the algorithm is that as the number of terminals increases, time required to generate a column increases exponentially. Therefore, it takes much time to generate a column when the number of branching with $x^k_e = 1$ becomes large, i.e., many nodes are generated. But, we can expect that a few nodes may be generated because LSBKP2 may give a tight lower bound.

Now, we present branching rule 1 in detail. For a given optimal solution $y^*$ to LSBKP2', we first calculate $x^k_e$ for all $e \in E$ and $k = 1, \ldots, n$ using Eq. (8). Then choose the most fractional (closest to 0.5) variable $x^k_e$ among $x^k_e$'s. From the current node in the branch-and-bound tree, we make two child nodes such that the first node represents the case where Steiner trees for net $k^*$ do not contain edge $e^*$ and the second node represents the case where Steiner trees for net $k^*$ contain edge $e^*$. That is, for the first node, we require

$$y_i = 0 \quad \text{for all } i \in T_{k^*}(e^*)$$

and for the second node, we require

$$y_i = 0 \quad \text{for all } i \in T_{k^*} \setminus T_{k^*}(e^*)$$

To satisfy the above requirements, for each Steiner tree variable $y_i$ already generated, we set the upper bound of the variable $y_i$ to zero if $i \in T_{k^*}(e^*)$ in the first node. In the second node, we set the upper bound of the variable $y_i$ to zero if $i \in T_{k^*} \setminus T_{k^*}(e^*)$. The column generation procedure is performed as explained before.

5.2. Branching rule 2

For a given optimal solution $y^*$ to LSBKP2', define $U(y^*, e)$ for all $e \in E$ as the set of nets for which at least one Steiner tree variable using edge $e$ have a positive value, i.e. $y^*_e > 0, i \in T_{k^*}(e)$.  


Clearly, if \( y^* \) is integral, \(|U(y^*, e)| \leq 1\) for all \( e \in E \). Although the converse is not true, we have the following positive result.

**Proposition 3.** Suppose that an optimal solution \( y^* \) to LSBK2’ with the objective value \( Z^* \) is given. If \(|U(y^*, e)| \leq 1\) for all \( e \in E \), \( y^* \) is either integral or there exists an integral feasible solution with the same objective value.

**Proof.** Suppose that \(|U(y^*, e)| \leq 1\) for all \( e \in E \), but \( y^* \) is a fractional solution. Let \( T^*_k \) be the set of variables for net \( k \) having positive values, i.e., \( T^*_k = \{i \in T_k: y^*_i > 0\} \).

For each net \( k = 1, \ldots, n \), if \(|T^*_k| = 1\), there exists only one non-zero variable for net \( k \). Otherwise, we select any variable among the elements of \( T^*_k \) and fix it to one and the others to zero. Considering \(|U(y^*, e)| \leq 1\), it is obvious that in the newly obtained solution two or more nets cannot contain the same edge (feasible solution). The variables in \( T^*_k \) should have the same objective coefficient. Otherwise, this contradicts that \( y^* \) is optimal solution. Hence, we can obtain an integral solution having the same objective value \( Z^* \). \( \Box \)

By the above proposition, we have only to check if \(|U(y^*, e)| \leq 1\) for all \( e \in E \) instead of checking the integrality of the given optimal solution \( y^* \) to LSBKP2’.

Now, we explain branching rule 2 in detail. For a given optimal solution \( y^* \) to LSBKP2’, we find \( e^* \in E \) such that \( e^* = \text{argmax}_{e \in E}|U(y^*, e)| \). If there are two or more edges having the same maximum cardinality of \( U(y^*, e) \), we choose the edge among them having the largest value of \( \sum_{k=1}^n x^*_k \). In other words, we choose the edge used maximally by the Steiner trees. Let \( U(y^*, e^*) = \{k_1, \ldots, k_m\} \). If \(|U(y^*, e^*)| = 1\), we do not need to branch by Proposition 3. Otherwise, we divide \( U(y^*, e^*) \) into two disjoint sets \( U_1(y^*, e^*) \) and \( U_2(y^*, e^*) \), such that \( U_1(y^*, e^*) = \{k_1, \ldots, k_{\lfloor m/2 \rfloor}\} \) and \( U_2(y^*, e^*) = U(y^*, e^*) \setminus U_1(y^*, e^*) \). Then two child nodes are generated in the branch-and-bound tree such that the Steiner trees for each net \( k \in U_1(y^*, e^*) \) cannot use the edge \( e^* \) in the first node and those for each net \( k \in U_2(y^*, e^*) \) cannot use the edge \( e^* \) in the second node. That is, for the first node, we require

\[
y_i = 0 \quad \text{for all} \quad i \in T_k(e^*) \quad \text{and} \quad k \in U_1(y^*, e^*)
\]

and for the second node, we require

\[
y_i = 0 \quad \text{for all} \quad i \in T_k(e^*) \quad \text{and} \quad k \in U_2(y^*, e^*).
\]

To satisfy the above requirements, for each Steiner tree variable \( y_i \) already generated, we first set the upper bound of the variable to zero if \( i \in T_k(e^*) \) and \( k \in U_1(y^*, e^*) \) in the first node. We perform similar bound setting in the second node. Then the current solution \( y^* \) does not satisfy either of the conditions, hence excluded, and none of the feasible integer solutions in the current node are excluded from further consideration. Further, for each net \( k \in U_1(y^*, e^*) \) we solve the column generation problem in the graph obtained from deleting \( e^* \) from the given graph \( G \) in the first node. For the second node, the same scheme is applied. Since the column generation problems can be solved by using the algorithm of Erickson et al. [23] over modified graphs, they remain tractable after branching.
6. Primal-heuristic

In this section, we describe our primal-heuristic in detail. The primal-heuristic is developed to obtain a tight upper bound on the optimal solution. The rationale behind our primal-heuristic is that the information of the LP solution will be of great help in obtaining a good feasible solution. A large value of \( x^k_e \) means that edge \( e \) is likely to be used for net \( k \). This idea is exploited as follows. For a given fractional solution \( \tilde{y} \) to LSDKP2, we calculate \( x^k_e \)'s using Eq. (8). For a given graph \( G = (V, E) \), assign weight \( w_e(1 - x^k_e) \) to edge \( e \) for all \( e \in E \). Then, to reduce the running time of the primal-heuristic we determine heuristically a Steiner tree for net \( k \) over the modified graph instead of finding the optimal Steiner tree.

Our primal-heuristic consists of two phases. In the first phase, for a given fractional solution \( \tilde{y} \), for each \( k \), we find a variable having the largest value among the Steiner tree variables for net \( k \)--we denote it by \( \tilde{y}^k_i \). If the value of \( \tilde{y}^k_i \) is greater than or equal to 0.9, we include the Steiner tree corresponding to the variable \( \tilde{y}^k_i \) in the primal-heuristic solution. Such Steiner trees cannot share an edge, otherwise the fractional solution, \( \tilde{y} \) does not satisfy the capacity constraint for the shared edge. Hence, we can include the Steiner trees in the primal-heuristic solution without violating the feasibility. In the second phase, we consider the remaining nets. We sort the remaining nets in decreasing order of \( \tilde{y}^k_i \)'s and find heuristically Steiner trees for the remaining nets in the sorted sequence. Details of the primal-heuristic are given as follows.

(Initialization) Set \( S = \emptyset \) and for \( k = 1, \ldots, n \), set \( S_k = \emptyset \).

[Phase 1] The following steps are performed:

Step 1: Sort the nets in decreasing order of \( \tilde{y}^k_i \), where \( \tilde{y}^k_i = \max_{i \in T_k, \tilde{y}_i} \). Let the ordered sequence be \( (1', \ldots, n') \)

Step 2: Find the largest index \( p' \) satisfying that \( \tilde{y}^k_{p'} \geq 0.9 \).

Step 3: For \( k = 1', \ldots, p' \), set \( S_k = \{ \text{edges of the Steiner tree corresponding to the variable } \tilde{y}^k_{p'} \} \) and \( S = S \cup S_k \).

[Phase 2] For \( k = (p + 1)', \ldots, n' \), the following steps are performed:

Step 1: Set \( R_k = N_k \) and construct a new graph \( G_k = (V, E_k) \) with edge weight \( w_e = 1 - x^k_e \) for all \( e \in E_k \), where \( E_k = E\setminus S \).

Step 2: If \( S_k = \emptyset \), determine \( s^*, t^* \) such that \( Sh_k(s^*, t^*) = \min_{s,t \in N_k} Sh_k(s,t) \), where \( Sh_k(s,t) \) is the value of the shortest path between the nodes \( s \) and \( t \) in \( G_k \).

If \( Sh_k(s^*, t^*) = \infty \), go to TERMINATION.

Else set \( S_k = S_k \cup \{ \text{edges in the } s^* - t^* \text{ path} \} \) and \( R_k = R_k \setminus \{ s^*, t^* \} \).

Else determine \( s^* \in R_k, t^* \in V(S_k) \) such that \( Sh_k(s^*, t^*) = \min_{s \in R_k, t \in V(S_k)} Sh_k(s,t) \).

If \( Sh_k(s^*, t^*) = \infty \), go to TERMINATION.

Else set \( S_k = S_k \cup \{ \text{edges in the } s^* - t^* \text{ path} \} \) and \( R_k = R_k \setminus \{ s^* \} \).

Remove the edges in the \( s^* - t^* \) path from the graph \( G_k = (V, E_k) \).

If \( R_k \neq \emptyset \), repeat Step 2.

Else, set \( S = S \cup S_k \) and go to Step 3.
Step 3: If $k = n'$, report a feasible solution $S$ and STOP.
   Else, set $k = (k + 1)'$ and go to Step 1.

TERMINATION) Report “No feasible solution found” and STOP.

In the second phase, we use a simple heuristic based on the shortest path algorithm to find a Steiner tree, although an optimal Steiner tree can be found in polynomial time. There are two major reasons. One is the possibility that the primal-heuristic fails to find a feasible solution. Due to this, we decided to perform the primal-heuristic as many times as possible, i.e., whenever the LP relaxation is reoptimized after some columns are added. Hence the primal-heuristic needs to be done in a short time. The other reason is that the primal-heuristic is only a heuristic. There is no guarantee that we can obtain a feasible solution when the optimal Steiner tree is found in the second phase. To our knowledge, a crucial point in obtaining a feasible solution in the primal-heuristic lies in the sequence of the Steiner trees found, not in whether the optimal Steiner tree is found or not.

If a feasible primal-heuristic solution is found and its value is less than the value of the current incumbent solution, then we add the Steiner tree variables in the primal-heuristic solution to the current LP relaxation and the current incumbent solution is updated. Of course, there is no guarantee that our primal-heuristic gives a good solution. However, we have obtained good upper bounds through our primal-heuristic for our test problems.

7. Implementation details

7.1. Initial solution

To use the column generation procedure, we need to have some initial feasible columns in LSBKP2. We have applied our branch-and-price algorithm on some test problems, which are explained in Section 7. For these problems, there are heuristic solutions reported in the literature [17–19]. These heuristic solutions are used as the initial solutions. For some problems, the heuristics failed in obtaining a feasible solution. i.e., terminals in some nets were not connected (see Burstein and Pelavin [17] and Luk [19]). In this case, to obtain the feasibility of an initial LP, artificial variables for such nets are introduced as in the next section. The objective value of an initial feasible solution is used as an upper bound on the optimal solution in the branch-and-price procedure.

7.2. Artificial variables

To guarantee that an instance of LSBKP2 is always feasible regardless of which Steiner trees are added in the current LSBKP2, artificial variables are introduced as follows. For each net, we introduce an artificial variable into the formulation with a sufficiently large objective coefficient $M$, a coefficient 1 in the corresponding row of constraints (4) and coefficient 0 in all the other
constraints. Then, constraints (4) become as follows.

\[ \sum_{i \in T_k} y_i + a_k = 1 \quad \text{for all } k = 1, \ldots, n. \]

We may start the column generation procedure with only artificial variables. But, this method may be inferior to the one starting with the columns of an initial solution in terms of the convergence speed.

7.3. Reducing problem size

Using the critical cuts, Grötschel et al. [2] have found the edges that cannot be contained in a Steiner tree for a net. By the same procedure in Grötschel et al. [2], we can delete some edges from the underlying graph for each net. This implies that the number of possible Steiner trees for each net may be reduced. More detailed explanation and the algorithm to find the critical cut can be found in Grötschel et al. [11,2].

7.4. Removing unnecessary columns

We need to remove unnecessary columns, if any, because too many columns cause some difficulties in solving linear programs. After solving LSBKP2 to optimality at node 0 in the enumeration tree, we can remove unnecessary columns using the following proposition.

**Proposition 4.** Let \( Z_{IN} \) and \( Z_{LP} \) be an incumbent solution value to SBKP2 and the optimal solution value to LSBKP2, respectively. Let \( r_i \) denote the reduced cost of a variable \( y_i \) when LSBKP2 is solved to optimality at node 0. If \( y_i \) is nonbasic at lower bound in the optimal solution of LSBKP2 and \( r_i > Z_{IN} - Z_{LP} \), then there exists an optimal solution to the integer program SBKP2 with \( y_i \) at its lower bound.

**Proof.** Since \( y_i \) is a binary variable, \( y_i \) must be 0 or 1 in the optimal solution to SBKP2. Let \( Z^* \) denote the optimal value of LSBKP2 when \( y_i \) is fixed to 1. Then, from the tableau of LSBKP2, we can see that \( Z^* \geq Z_{LP} + r_i \). This implies \( Z^* > Z_{IN} \), since \( r_i > Z_{IN} - Z_{LP} \). Hence, there are no optimal solutions to SBKP2 with \( y_i = 1 \) and it means that there exists an optimal solution to SBKP2 with \( y_i = 0 \). \( \square \)

We can remove the columns satisfying the condition of Proposition 4 after solving LSBKP2 to optimality at node 0. In order to remove many columns, we need a good incumbent solution. This is one of the reasons why we need a primal-heuristic.

7.5. Solving the LP relaxation of SBKP2

When we solve the LP relaxation of SBKP2 by column generation, a primal simplex method is a natural choice for reoptimization than a dual simplex method since the added columns definitely
violate the dual feasibility, but the current basic solution still satisfies the primal feasibility. However, in our preliminary tests, dual simplex method seems to be more efficient than primal to our problems. One major reason is that our problem is highly degenerate in the primal. The other formulation of SBKP introduced in Section 2 has similar difficulties in optimizing LP relaxation. The LP relaxation of SBKP1 (LSBK1) is highly degenerate in both primal and dual [2]. To overcome such difficulties Grötschel et al. [10] developed an elaborate perturbation method. But, in our case, the LP relaxation is highly degenerate only in primal. Therefore, we used the dual simplex method instead of perturbation method. Whenever optimizing a linear program using dual simplex method, we must solve it from the beginning again. So a dual feasible solution should be found in a short time. Fortunately, since objective coefficients are all positive, we expect that CPLEX will not have any trouble in finding dual feasible solutions within allowable time, even after adding the columns (CPLEX callable mixed integer library is used as an LP solver). In experimental results, it appeared that the dual simplex method is more appropriate than the primal to our problem.

8. Computational results

We have solved the switchbox routing problems on which Grötschel et al. [10] tested their cutting plane algorithm. These problems are standard examples in this area. The data of test problems are summarized in Table 1.

In Table 1, column 1 presents the name used in the literature. The height and width of the underlying complete rectangular grid graph are given in columns 2 and 3, respectively. Column 4 presents the number of nets. Columns 5–9 describe the distribution of nets; more precisely, column 5 gives the number of two-terminal nets and column 6 gives the number of three-terminal nets and so on. The last column gives the reference to the paper the test problem is taken from. More detailed explanations of the test problems can be found in Grötschel et al. [2]. Based on the distribution of nets, we can estimate approximately the relative difficulty of the test problem. The problem becomes more difficult as the number of nets with many terminals increases. The dense

Table 1
Data of test problems [2]

<table>
<thead>
<tr>
<th>Name</th>
<th>H</th>
<th>B</th>
<th>N</th>
<th>Distribution of the nets</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Difficult</td>
<td>15</td>
<td>23</td>
<td>24</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>More difficult</td>
<td>15</td>
<td>22</td>
<td>24</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Terminal intensive</td>
<td>16</td>
<td>23</td>
<td>24</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Dense</td>
<td>17</td>
<td>15</td>
<td>19</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>Augmented dense</td>
<td>18</td>
<td>16</td>
<td>19</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>Modified dense</td>
<td>17</td>
<td>16</td>
<td>19</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>Pedagogical</td>
<td>16</td>
<td>15</td>
<td>22</td>
<td>14</td>
<td>4</td>
</tr>
</tbody>
</table>
problems (dense, augmented dense, modified dense) seem to be more difficult than the other test problems.

At the beginning of our algorithm, we reduce the problem size by finding critical cuts (see Section 6.3). The effects of reducing the problem size are reported in Table 2. In Table 2, columns 2 and 3 give the number of edges and the number of deleted edges in graphs for all nets. Columns 4 (5) states the number of generated variables without (with) reducing process when we solve LSBKP2 at node 0. We may expect that deleting some edges from the underlying graph reduces the number of generated variables. But, the numbers in Table 2 imply that this process has little positive effect on the number of generated variables (in some problems, more variables are generated). However, this reduces the number of possible choices in branching and the number of constraints in the linear program, hence we decided to apply this process.

When LSBKP2 is solved to optimality at node 0, we can remove some unnecessary columns using Proposition 4 (see Section 5). The results are shown in Table 3. Columns 2 and 3 present the
Table 4
Results of branch and price with branching rule 1 (HP 9000 715/50)

<table>
<thead>
<tr>
<th>Example</th>
<th>Best solution</th>
<th>LP value</th>
<th>Gap (%)</th>
<th>Iterations</th>
<th>Total columns</th>
<th>Nodes</th>
<th>CPU-time (m:s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficult</td>
<td>464</td>
<td>464</td>
<td>0.0</td>
<td>68</td>
<td>935</td>
<td>1</td>
<td>32:26</td>
</tr>
<tr>
<td>More difficult</td>
<td>452</td>
<td>452</td>
<td>0.0</td>
<td>58</td>
<td>816</td>
<td>1</td>
<td>25:20</td>
</tr>
<tr>
<td>Terminal intensive</td>
<td>536</td>
<td>535</td>
<td>0.19</td>
<td>616</td>
<td>4336</td>
<td>32</td>
<td>853:00</td>
</tr>
<tr>
<td>Dense</td>
<td>441</td>
<td>438</td>
<td>0.68</td>
<td>841</td>
<td>10,000*</td>
<td>10</td>
<td>1296:58</td>
</tr>
<tr>
<td>Augmented dense</td>
<td>469</td>
<td>467</td>
<td>0.43</td>
<td>1435</td>
<td>10,000*</td>
<td>35</td>
<td>1661:55</td>
</tr>
<tr>
<td>Modified dense</td>
<td>452</td>
<td>452</td>
<td>0.0</td>
<td>201</td>
<td>3089</td>
<td>1</td>
<td>131:49</td>
</tr>
<tr>
<td>Pedagogical</td>
<td>331</td>
<td>331</td>
<td>0.0</td>
<td>37</td>
<td>318</td>
<td>2</td>
<td>6:44</td>
</tr>
</tbody>
</table>

*Means that we solved the problem until the number of generated columns is less than or equal to 10,000.

Table 5
Results of branch and price with branching rule 2 (HP 9000 715/50)

<table>
<thead>
<tr>
<th>Example</th>
<th>Best solution</th>
<th>LP value</th>
<th>Gap (%)</th>
<th>Iterations</th>
<th>Total columns</th>
<th>Nodes</th>
<th>CPU-time (m:s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal intensive</td>
<td>536</td>
<td>535</td>
<td>0.19</td>
<td>241</td>
<td>2324</td>
<td>15</td>
<td>222:35</td>
</tr>
<tr>
<td>Dense</td>
<td>441</td>
<td>438</td>
<td>0.68</td>
<td>1042</td>
<td>10,000*</td>
<td>10</td>
<td>1604:01</td>
</tr>
<tr>
<td>Augmented dense</td>
<td>469</td>
<td>467</td>
<td>0.43</td>
<td>1730</td>
<td>10,000*</td>
<td>70</td>
<td>1670:25</td>
</tr>
<tr>
<td>Pedagogical</td>
<td>331</td>
<td>331</td>
<td>0.0</td>
<td>124</td>
<td>493</td>
<td>54</td>
<td>22:37</td>
</tr>
</tbody>
</table>

*Means that we solved the problem until the number of generated columns is less than or equal to 10,000.

value of the best feasible solution among the primal-heuristic solution and initial solution and that of optimal solution to LSBKP2, respectively. For dense problem, the initial solution is infeasible and we could not find any primal-heuristic solution at node 0. The last column shows the number of removed columns and the proportion of unnecessary columns to the generated columns in percentage terms. If the number in column 3 is close to that in column 2, i.e., a good feasible solution is obtained, then many unnecessary columns can be removed and this enables us to handle the LP relaxation more efficiently. We observed this phenomena in some of test problems (Terminal intensive, Augmented dense).

The results we have obtained with our branch-and-price algorithm are shown in Tables 4 and 5. We have applied two branching rules given in Section 4. Tables 4 and 5 present the results with branching rules 1 and 2, respectively. Table 6 presents the results Grötschel et al. [2] obtained with the branch-and-cut algorithm on the same problems, which we include here for comparison. In Table 5, those problems for which an integer optimal solution is found at node 0 are not presented. Best solution means the value of the best feasible solution. LP value denotes the objective function value of the linear program at node 0. We can obtain the lower bounds on problems by rounding up these values. Gap is the percentage deviation of the best solution from the lower bound. This value is calculated as the following: gap = (best solution value − ⌈LP value⌉)/⌈LP value⌉. In
Table 6
Results of branch and cut by Grötschel et al. [2] (SUN 4/50)

<table>
<thead>
<tr>
<th>Example</th>
<th>Best solution</th>
<th>LP value</th>
<th>Gap (%)</th>
<th>Iterations</th>
<th>Total variables</th>
<th>Nodes</th>
<th>CPU-time (m:s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficult</td>
<td>464</td>
<td>464</td>
<td>0.0</td>
<td>69</td>
<td>13424</td>
<td>3</td>
<td>1564:15</td>
</tr>
<tr>
<td>More difficult</td>
<td>452</td>
<td>452</td>
<td>0.0</td>
<td>53</td>
<td>12502</td>
<td>1</td>
<td>983:23</td>
</tr>
<tr>
<td>Terminal intensive</td>
<td>537</td>
<td>536</td>
<td>0.19</td>
<td>163</td>
<td>11815</td>
<td>13</td>
<td>3755:44</td>
</tr>
<tr>
<td>Dense</td>
<td>441</td>
<td>438</td>
<td>0.68</td>
<td>119</td>
<td>4251</td>
<td>4</td>
<td>1017:43</td>
</tr>
<tr>
<td>Augmented dense*</td>
<td>469</td>
<td>467</td>
<td>0.43</td>
<td>105</td>
<td>7620</td>
<td>1</td>
<td>4561:41</td>
</tr>
<tr>
<td>Modified dense</td>
<td>452</td>
<td>452</td>
<td>0.0</td>
<td>51</td>
<td>5652</td>
<td>1</td>
<td>387:03</td>
</tr>
<tr>
<td>Pedagogical</td>
<td>331</td>
<td>331</td>
<td>0.0</td>
<td>77</td>
<td>7839</td>
<td>5</td>
<td>251:58</td>
</tr>
</tbody>
</table>

*For the problems, no optimal solution is obtained.

Tables 4 and 5, columns 5 and 6 give the number of column generation iterations and the number of generated columns, respectively. In Table 6, columns 5 and 6 give the number of cutting plane iterations and the number of total variables, respectively. In Tables 4–6, column 7 presents the number of nodes solved in the branch-and-bound tree. Finally, the last columns of all tables report on running times in minutes.

LSBK2 gives a very tight lower bound as we expected in Section 2 (the largest gap is 0.68%, see Tables 4 and 5). These lower bounds are the same as Grötschel et al. [2] except for terminal intensive problem. In terminal intensive problem, the branch-and-cut algorithm of Grötschel et al. [2] gives tighter lower bound. The cutting planes used by Grötschel et al. [2] are Steiner partition inequality, alternating cycle inequality and critical cut. The Steiner partition inequality is a cutting plane for the convex hull of a Steiner tree and since the associated separation problem is \textit{NP-hard}, Grötschel et al. [2] solved it heuristically. The alternating cycle inequality is a cutting plane for the convex hull of Steiner tree packing with two nets and the associated separation problem was solved heuristically due to involved difficulty. The critical cut is used to fix some variables to zero as stated in Section 6.3. Now, consider two facts; one is that LSKB2 describes fully the convex hull of a Steiner tree. The other is that our algorithm gives the same lower bounds as Grötschel et al. [2] except for terminal intensive problem. Based on these observations, we may give some remarks regarding the strength of the cutting planes used in Grötschel et al. [2] in view of lower bound. First, the cutting planes (Steiner partition inequalities, alternating cycle inequalities, and critical cuts) do much for achieving a good lower bound since the convex hull of a Steiner tree can not be described fully by only the Steiner cut inequality. Second, if the Steiner partition inequality describes well the convex hull of a Steiner tree although its separation problem is not solved optimally, the alternating cycle inequality does not make a large contribution to the lower bound. Otherwise, it may compensate for deficiency of Steiner partition inequality.

For all problems, we obtained the same best solution value as Grötschel et al. [2] did. Especially, for terminal intensive problem, we obtained the optimal solution with value 536, while Grötschel et al. [2] stated that the optimal value is 537. We discussed this result with Martin [11,2] and he solved the problem once more to get the same optimal value. For dense and augmented dense problem, we failed to obtain the optimal solution because, due to the limit of memory, we set the
maximum number of columns included in the linear program to 10,000. Grötschel et al. [2] could not obtain optimal solutions for these problems, either.

We solved the test problems on a HP 9000 715/50 and Grötschel et al. [2] solved them on a SUN 4/50. The information of these two machines is as follows: The CPU clock, million instructions per second (MIPS) and million floating point operations per second (MFLOPS) of HP 9000 715/50 are 50 MHz, 62 and 13, respectively. The CPU clock, MIPS and FLOPS of SUN 4/50 are 40 MHz, 28.5 and 4.2. With only this information, it is unreasonable for us to give some statements for the speeds of two machines. However, even assuming that HP 9000 715/50 is three times faster than SUN 4/50, our algorithm is competitive to Grötschel et al. [2] in running time for all problems except dense and augmented dense problem for which they could not find optimal solutions either.

Now, we discuss the results of branching rules 1 and 2. With the results in Tables 4 and 5, we cannot determine which branching rule is better. Dense and augmented problems are difficult to solve irrespective of the branching rule applied. For the terminal intensive problem, branching rule 2 is better than branching rule 1. But, branching rule 1 is better than branching rule 2 for the pedagogical problem. As stated in Sections 4 and 5, depending on the branching rule applied, the time complexity of column generation problem is different. The column generation problem can be solved in polynomial time with branching rule 2 but not with branching rule 1. However, the effect of different time complexities did not appear apparently in our results.

9. Conclusion

In this paper, we proposed an algorithm based on column generation procedure to solve the Steiner tree packing problem. We explained how the LP relaxation of the problem, which has exponentially many variables, is solved by column generation procedure. We also developed branching rules which make the column generation problem tractable after branching.

Computational results show that the algorithm solved test problems more efficiently than Grötschel et al. [2] even though our formulation has exponentially many variables. We can expect that our algorithm can be used to solve practically sized problems.

As stated earlier, there are many possible models for routing problem in VLSI design since the routing problem can be defined by combining two criteria. Our model is for only one of these possible models. The other models can be very interesting topics for further research.

References


Gue-woong Jeong is a Ph.D. student of industrial engineering at the Korea Advanced Institute of Science and Technology (KAIST). He received the B.S. and M.S. degrees in industrial engineering from KAIST. His research interests include network reliability, integer programming and combinatorial optimization.

Kyungzik Lee is a senior member of technical staff at the Department of Postal Technology of Electronics and Telecommunications Research Institute. He finished his undergraduate study at the department of industrial engineering of Seoul National University in 1993 and received M.S. and Ph.D. degrees in industrial engineering from KAIST in 1995 and 1998, respectively. His current research interests include the theory of integer programming and combinatorial optimization, telecommunication network planning and design, and supply chain management and logistics information systems.

Sungsoo Park received the B.S. and M.S. degrees in industrial engineering from Seoul National University in 1980 and 1982, and the Ph.D. degree in operations research from Cornell University in 1989, respectively. Since 1990, he has been with the department of industrial engineering at KAIST. His research interests include integer programming and combinatorial optimization, telecommunication network planning and design, mathematical models in production and distribution systems.

Kyungchul Park is a senior member of technical staff in the Telecommunication Network Lab of Korea Telecom. He received his Ph.D. degree in industrial engineering from KAIST. His current research interest includes combinatorial optimization, integer programming and telecommunication network design.