The Rectilinear Class Steiner Tree Problem for Intervals on two Parallel Lines

*†‡

Edmund Ihler §

Abstract

We consider a generalization of the RECTILINEAR STEINER TREE problem, where our input is classes of required points instead of simple required points. Our task is to find a minimum rectilinear tree connecting at least one point from each class. We prove that the version, where all required points lie on two parallel lines, called the RECTILINEAR CLASS STEINER TREE (*channel*) problem, is NP-hard. But we give a linear time algorithm for the case where the points of each required class are clustered, and the classes consist of non overlapping intervals of points.

1 Introduction

Given pairwise disjoint sets $R_1, \ldots, R_c$ of *required points* in the plane, called *(required) classes*, find a minimum rectilinear tree connecting at least one point of each class. This problem is called RECTILINEAR CLASS STEINER TREE problem. In general the problem is NP-hard because the restricted version where classes contain exactly one point is the well known RECTILINEAR STEINER TREE problem [GJ77]. This has been studied intensively in the past (for an overview see e.g. [SW91]), and has a wide variety of applications in different fields such as in physical VLSI circuit design and

*Part of this research was conducted while the author was attending a research initiative at the Leonardo Fibonacci Institute, Povo, Italy.

†This work was partly supported by the Deutsche Forschungsgemeinschaft under grant Wid 810/5.

‡Extended abstract appeared in: *Graph-Theoretic Concepts in Computer Science, WG92*, No. 657 of *Lecture Notes in Computer Science*, pages 123–134, Springer, 1993. Full version to appear in: Mathematical Programming, 63, 1994.

§Institut für Informatik, Universität Freiburg, Rheinstr. 10–12, D-79104 Freiburg; email: ihler@informatik.uni-freiburg.de

communication network design. The advantage of the more general version using classes is that it gives us additional freedom, since it allows a point to lie in different possible positions. In routing applications for VLSI-chips, where many nets have to be packed, standard algorithms for single nets can be applied in different ways. The application simply use them to compute trees for the different nets sequentially, or partially simultaneously [KPS90]. Sequential techniques are also applicable for version using classes to model the additional freedom. Unfortunately because of the NP-hardness for both problems, in general only heuristic or approximation algorithms are practical, but some restrictions that reduce the complexity of the problems are useful in practice. For example in the detailed routing of physical VLSI layouts, so called *channel routing* is of great importance [Len90]. Here, the terminals of the nets lie on one of two parallel lines, which defines a *channel*. Using required classes we have an improved model which produces shorter wire lengths. For example, if an AND-gate cell has its pins on the border of the channel, then there are several possibilities to apply classes: *logically equivalent* pins, *equipotential pins*, and pins that can be moved along the channel border, each define a separate class. What does this mean? Assuming that we want to compute different nets sequentially, and that the first must be connected to an input pin of the AND-gate, we have the freedom to choose an arbitrary input pin of this gate. So can let these logically equivalent pins be modeled by a class. Clearly this class will have to be reduced by one member for the second net. For equipotential pins the situation is even better. These pins are connected within the layout of the gate, and if we model these pins with a class, then this class does not change for the subsequent nets. Very often the position of a pin is not fixed to a point, but can be an arbitrary point within some interval on the channel border. A class then, representing this pin contains not one point, but an entire interval. Clearly the order in which we process the nets influences the quality of the routing. But what we are interested in is not to find an optimal order. We investigate here, how we can use classes of points instead of single points for the computation of a single net of minimum total length in an efficient way.

In this paper we investigate the RECTILINEAR CLASS STEINER TREE problem for *class channels*, where the required classes are defined by non overlapping intervals on one of two parallel lines. This special case is called RECTILINEAR CLASS STEINER TREE (*channel*) problem. In Sect. 2, after giving a precise definition of the problem, we prove that the problem is reducable to a discrete graph version, where the input is a *class ladder graph*, i.e. the Hanangraph of the boundary points of the required intervals of a class channel. This graph problem is called CLASS STEINER TREE (*ladder*) problem and is a restricted version of a more general graph problem, the CLASS STEINER TREE problem, that was introduced by [RW89] and remains NP-hard and MAX SNP-hard [Ihl92] for trees of bounded diameter.

Further we prove in Sect. 3 that our restricted version of the problem is also NP-hard. This consequently implies NP-hardness for the RECTILINEAR CLASS STEINER TREE (*channel*) problem. Fortunately, for many applications the required points of the classes are found in clusters. This case can be translated into the restriction that for any given vertical line dividing any class channel, the number of classes that have a point on the right, and also on the non right side of this line can be bounded by some constant k; we call this a *constant vertical class cut*. This requirement is natural for channel routing for standard cells. Here each class originates from a unique cell, and each cell can generate only a constant number of classes, because its size is bounded. Note that the constant vertical class cut requirement is a more general restriction than a constant bounded diameter for the channel or for the classes. The algorithm we present in Sect. 4 is a generalization of the dynamic programming algorithm for the special case of the class channel where each class consists of a unique point [AGH77]. We present a slightly modified version of this algorithm for class ladder graphs in Sect. 4.1 and extend it for general classes in Sect. 4.2. The time complexity of this algorithm is $O(mk2^k)$ and works equally well for class ladder graphs and class channels.

2 The class channel

Given two parallel, w.l.o.g. horizontal lines in a plane. We define a *class channel* as a finite, non empty set of non overlapping intervals, lying on these two lines. We call these intervals *required intervals* and their points *required points*. The set of all required intervals is partitioned into c subsets. This partitions of the set of required points R into the subsets $R_1, \ldots, R_c$, called *(required) classes* (see upper part of Fig. 3). A *rectilinear class Steiner tree* of a class channel for the classes $R_1, \ldots, R_c$, is a tree that connects at least one point of each required class, and is composed of a finite number of horizontal and vertical line segments. Here, three or more line segments can meet at a point $v \notin R$. Such a point v is called a *Steiner point*. A *minimum rectilinear class Steiner tree* for $R_1, \ldots, R_c$ is a rectilinear class Steiner tree of minimum total length, i.e. the sum of the lengths of all its line segments is minimum among all such trees, where the length of a line segment is given by the L_1-metric.

Definition 1 The problem of finding a minimum rectilinear class Steiner tree for any given class channel is called the RECTILINEAR CLASS STEINER TREE *(channel)* problem.

Note that for $|R_i| = 1$, $1 \leq i \leq c$, where each class consists of a unique required interval of length 0, the problem is simplified to the well known RECTILINEAR STEINER TREE (*channel*) problem. From the Theorem of

Hanan [Han66], it is well known that the general Rectilinear Steiner Tree problem is disrete. Consider the grid formed by the vertical and horizontal lines containing at least one required point of R, and let $H(R)$ be the set of all vertical and horizontal line segments $[v, w]$ where v and w are grid points and $[v, w]$ contains no further grid point. We call the elements of $H(R)$ *Hanan line segments*. Hanan's theorem says that for each finite set of points R there is a minimum rectilinear Steiner tree for R that can be created from a subset of the Hanan line segments $H(R)$. We want to prove something similar for class channels. When we compare this with the Rectilinear Steiner Tree (*channel*) problem we notice two differences. First, there is the additional complication of having multiple points in each class, and second, we allow as input an infinite number of required points because we are considering intervals. Nevertheless it is still possible for the Rectilinear Class Steiner Tree (*channel*) problem to consider only a finite number of rectilinear class Steiner trees. Before we claim such a result, let us define more formally how a class channel is encoded.

Let $a_1, \ldots, a_m$ be discrete points on the top line from the left side to the right side, and let $b_1, \ldots, b_m$ be discrete points on the bottom line. We require that for each i, $1 \leq i \leq m$, there is a vertical line that contains a_i and b_i. Let our disjoint required intervals be represented by a binary relation $I \subseteq V_a^2 \cup V_b^2$, where $V_a = \{a_1, \ldots, a_m\}$, $V_b = \{b_1, \ldots, b_m\}$, and $V = V_a \cup V_b$, where $(v, w) \in I$ means that $[v, w]$ is an interval on one of the horizontal lines. A pair $(v, v) \in I$ represents an interval of length 0. The points of V are partitioned by a function $\rho : V \to \{0, 1, \ldots c\}$, where we require that all points of V in the same required interval be mapped to the same value, i.e. $\forall (v, w) \in I \; \forall u \in [v, w] \cap V : \rho(v) = \rho(u) = \rho(w)$. A value $\rho(v) = 0$ indicates a Steiner point. Because R is the set of all points that lie on required intervals, $R = \bigcup_{(v,w) \in I} [v, w]$. The classes are defined by ρ, i.e. $R_i = \bigcup_{(v,w) \in I \wedge \rho(v) = i} [v, w]$, $1 \leq i \leq c$. A tree T *represents* class R_i, if T contains a point of R_i. All classes are assumed to be nonempty, i.e. $\rho(V) \supseteq \{1, \ldots, c\}$. Finally, we define a class channel with the tuple $((a_1, \ldots, a_m), (b_1, \ldots, b_m), I, \rho)$, where the components adhere to the requirements above. We are now able to formulate the following lemma:

Lemma 1 *Given any class channel $((a_1, \ldots, a_m), (b_1, \ldots, b_m), I, \rho)$, then (i) there is a minimum rectilinear class Steiner tree for $R_1 \cap V, \ldots, R_c \cap V$ that consists of a subset of the Hanan line segments $H(V)$, and (ii) if T is a minimum rectilinear class Steiner tree for $R_1 \cap V, \ldots, R_c \cap V$, then T is a minimum rectilinear class Steiner tree for $R_1, \ldots, R_c$, too.*

Proof: The idea for proving part (ii) is the following: Assume that we are able to modify a minimum rectilinear class Steiner tree T_1 for the classes $R_1, \ldots, R_c$ giving a tree T_m such that T_m satisfies two conditions: (1) T_m represents each class $R_i \cap V$, $1 \leq i \leq c$, and (2) the total length does not increase, i.e. $l(T_m) \leq l(T_1)$. Then this means that T_m is a minimum rectilinear class Steiner tree for the classes $R_1, \ldots, R_c$, too. On the other hand T_m is also a minimum rectilinear class Steiner tree for the classes $R_1 \cap V, \ldots, R_c \cap V$. To see this, assume a shorter rectilinear class Steiner tree T for $R_1 \cap V, \ldots, R_c \cap V$. T then would also be a rectilinear class Steiner tree for $R_1, \ldots, R_c$, and therefore $l(T) < l(T_1)$, and hence a contradiction. So each minimum rectilinear class Steiner tree for $R_1 \cap V, \ldots, R_c \cap V$ has the length of T_1, this then proves part (ii) of this lemma.

So all we have to do now, is to find such a tree T_1 and a modification procedure for carrying out the process above. We know that there is a minimum rectilinear class Steiner tree for $R_1, \ldots, R_c$ containing by definition some tuple $(r_1, \ldots, r_c) \in R_1 \times \ldots \times R_c$. A minimum rectilinear class Steiner tree for the classes $\{r_1\}, \ldots, \{r_c\}$ is also then a rectilinear class Steiner tree for $R_1, \ldots, R_c$. In addition Hanan's theorem guarantees a minimum rectilinear class Steiner tree T_1 for $\{r_1\}, \ldots, \{r_c\}$ that consists of a subset of the Hanan line segments $H(\{r_1, \ldots, r_c\})$. Clearly this proves part (i) of this lemma.

To prove part (ii), we transform T_1 iteratively to T_m. In the iteration i, $1 \leq i \leq m-1$, tree T_i is changed to T_{i+1} by replacing some of the line segments of T_i that lie between the vertical line segments $[a_i, b_i]$ and $[a_{i+1}, b_{i+1}]$, excluding the vertical line segments themselves. We call this region of the plane *slice*(i) and the set of line segments of T_i inside slice(i), $S(i)$. By the definition of I we know that no required interval has its endpoint inside slice(i). Therefore either all points of the interval $[a_i, a_{i+1}] \setminus \{a_i, a_{i+1}\}$ (resp. $[b_i, b_{i+1}] \setminus \{b_i, b_{i+1}\}$) are Steiner points, or all points of $[a_i, a_{i+1}]$ (resp. $[b_i, b_{i+1}]$) belong to the same class. So at most two points of our fixed tuple $(r_1, \ldots, r_c)$ lie in slice(i), one on the top line, and one on the bottom line. Therefore there are at most two vertical line segments in $S(i)$, since T_1 lies on $H(\{r_1, \ldots, r_c\})$ (see Fig. 1). It can easily be seen by case analysis when T_1 is

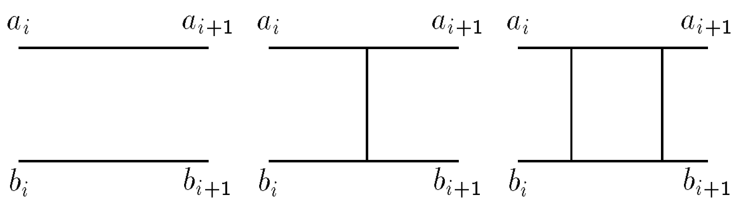


Figure 1: The three possibilities for $H(\{r_1, \ldots, r_c\})$ in slice(i).

minimized, at most one vertical line segment is allowed in $S(i)$, and only a few topologies for $S(i) \subseteq H(\{r_1, \ldots, r_c\})$ are possible. Our intended modification of $S(i)$ becomes obvious when looking at the possible topologies symbolized in Fig. 2: Shift a vertical line segment to the left and, if necessary, replace a horizontal line segment by its counterpart on the opposite side, in order to reconnect the tree. By induction it can be seen that any T_m constructed by this iterative modifications satisfies the conditions (1) and (2) stated at the

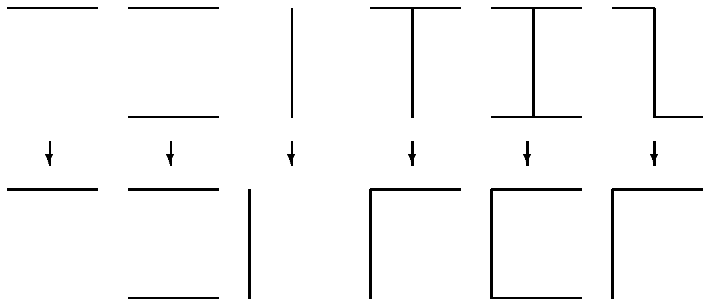


Figure 2: The possible topologies of $S(i)$ and its replacement.

begin of this proof. □

Corollary 1 *Given any class channel $((a_1, \dots, a_m), (b_1, \dots, b_m), I, \rho)$, there exists a minimum rectilinear class Steiner tree for $R_1, \dots, R_c$ that consists of a subset of the Hanan line segments $H(V)$.* □

Similary it is possible to prove that for the classes $R_i \cap \{v, w \mid (v, w) \in I\}$, $1 \leq i \leq c$, consisting of the boundary points on the required intervals. There are also several trivial cases of class channels, which have to be handled separately, but each of these can be solved in linear time $O(m)$. Nevertheless, since this stricter lemma does not improve our subsequent results, Lemma 1 is sufficient.

We now know, that for any given class channel we can restrict ourselves to the subclasses $R_1 \cap V, \dots, R_c \cap V$ and the Hanan line segments of $H(V)$. This is our motivation for considering a closely related graph problem. Note that each class channel induces an edge weighted graph $(V, H(V), l)$ by its sets V and $H(V)$, where $l : H(V) \to I\!R^+$ is the length function for the edges. Therefore the points of V are *nodes* and the line segments of $H(V)$ *edges*. Function l is induced by the L_1-metric. We call a tuple $((V, H(V), l), \rho)$ defined by a class channel a *class ladder graph*, since the graph looks like a ladder (see Fig. 3). Let the classes of a class ladder graph be the sets of nodes $R_i = \{v \in V \mid \rho(v) = i\}$, $1 \leq i \leq c$. This means that the classes of a class ladder graph consist of the nodes in the classes of the corresponding class channel. If a tree T is a subgraph of G and T contains at least one node from each class, then T is a *class Steiner tree*. A class is said to be *represented* in a subgraph of G, if the subgraph contains a node from this class. A class Steiner tree of minimum total length is called *minimum class Steiner tree*.

Definition 2 The problem of finding a minimum class Steiner tree for any given class ladder graph is called the CLASS STEINER TREE *(ladder)* problem.

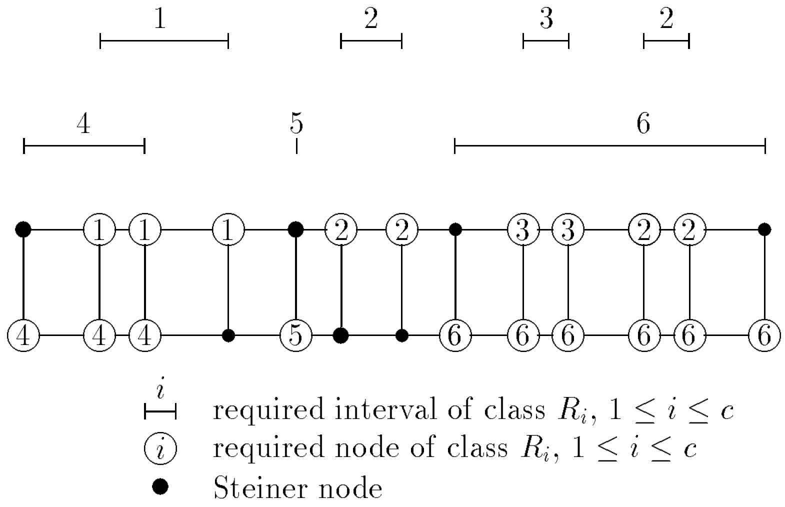


Figure 3: The Hanangraph of a class channel is a ladder graph.

Note, that many class channels lead to the same class ladder graph, and that there is always a class channel among them where all the required line segments have length 0. It is obvious that the tuple $((a_1, \ldots, a_m), (b_1, \ldots, b_m), I, \rho)$ representing a class channel can also encode the corresponding class ladder graph, where I is redundant. Moreover, if we interprete a given class channel as a class ladder graph and can compute a minimum class Steiner tree, then this tree can immediately be reinterpreted as a minimum rectilinear class Steiner tree for the class channel immediately. In other words, an algorithm for the Class Steiner Tree (*ladder*) problem is an algorithm for the Rectilinear Class Steiner Tree (*channel*) problem and the time complexities are the same. If the number of classes c is bounded by a constant, or one side of the class channel contains no required nodes, then the Class Steiner Tree (*ladder*) problem is solvable in polynomial time [Ihl91, Ihl92]. Remember that the Class Steiner Tree ($|R_i|$=1, *ladder*) problem, where each class consists of a single node, is solvable even in linear time [AGH77]. In this case we will sometimes denote an input (G, ρ) by $(G, \bigcup_{i=1}^{c} R_i)$. But in general (the decision version of) the Class Steiner Tree (*ladder*) problem is NP-complete as we will prove in the next section. It is also an obvious consequence that Rectilinear Class Steiner Tree (*channel*) is NP-complete.

3 NP-completeness

In this section we will prove that the Class Steiner Tree (*ladder*) problem (more precisely the corresponding decision problem) is NP-complete, even for classes which contain at most three nodes. To this end, we reduce the problem

3SAT5 to it. An instance of **3SAT5** problem is a collection $\{c_1, \ldots, c_s\}$ of clauses on a finite set $\{u_1, \ldots, u_n\}$ of boolean variables, where $|c_i| = 3$ for $1 \leq i \leq s$ and at most 5 clauses contain either u_i or $\bar{u}_i$ for $1 \leq i \leq n$. The question is, whether there is a *truth assignment* for the variables which satisfies all clauses. This problem is NP-complete [GJ79].

Theorem 1 *The* Class Steiner Tree *(ladder) problem is NP-complete for classes which consist of at most three nodes.*

Proof: We compute a class ladder graph (G, ρ) and a positive integer b from an input for the **3SAT5** problem in the following way: For each *literal* $l \in \{u_1, \bar{u}_1, \ldots, u_n, \bar{u}_n\}$ construct a *literal component* $L(l)$ of G that is a class ladder graph itself. On the top row of $L(l)$ there are the *top nodes* $a_1(l), \ldots, a_6(l)$ and on the bottom row the *literal node* $b_1(l)$ and the *clause nodes* $b_2(l), \ldots, b_6(l)$. For each literal l let the horizontal edges $(a_i(l), a_{i+1}(l))$, $(b_i(l), b_{i+1}(l))$, $1 \leq i \leq 5$, of $L(l)$ have a length of 1, and let the vertical edges $(a_i(l), a_i(l))$, $1 \leq i \leq 6$, of $L(l)$ have a length of $D = 5n + 1$. We now connect

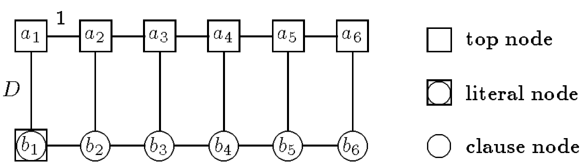
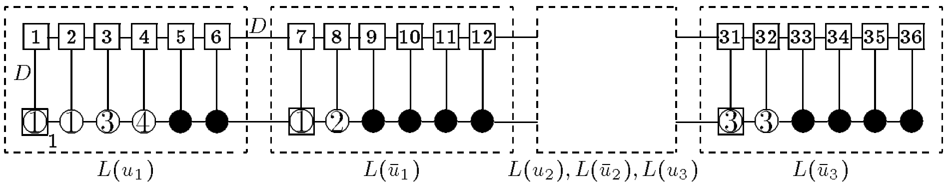


Figure 4: The literal component $L(l)$ of literal l.

the literal components by horizontal edges $(a_6(u_i), a_1(\bar{u}_i))$, $(b_6(u_i), b_1(\bar{u}_i))$, $1 \leq i \leq n$, and $(a_6(\bar{u}_i), a_1(u_{i+1}))$, $(b_6(\bar{u}_i), b_1(u_{i+1}))$, $1 \leq i \leq n-1$, where each of these edges has a length of D (see Fig. 4). The result is a long class ladder graph built from the sequence $L(u_1), L(\bar{u}_1), \ldots, L(u_n), L(\bar{u}_n)$. For each top node of each literal component we create a *top class*, i.e. $\rho(a_i(u_j)) = (j-1)12 + i$ and $\rho(a_i(\bar{u}_j)) = (j-1)12 + 6 + i$, $1 \leq i \leq 6$, $1 \leq j \leq n$, with the number of top classes being $m = 12n$. For each pair of literal nodes $b_1(u_i)$, $b_1(\bar{u}_i)$ we create an additional *literal class*, i.e. $\rho(b_1(u_i)) = \rho(b_1(\bar{u}_i)) = m + i$, $1 \leq i \leq n$. With the remaining nodes $b_2(l), \ldots, b_6(l)$, $l \in \{u_i, \bar{u}_i \mid 1 \leq i \leq n\}$, we create s *clause classes*. For each clause c_i, $1 \leq i \leq s$, and for each literal l contained in c_i, we select node v from $\{b_2(l), \ldots, b_6(l)\}$ that has not yet been selected, and let $\rho(v) = m+n+i$. This is possible because each literal occurs in at most 5 clauses. Let the nodes v not selected by this procedure are Steiner nodes, i.e. $\rho(v) = 0$. Finally we let $b = 15n^2 + 13n - 1$. Clearly this whole computation can be done in polynomial time (see Fig. 5).

Assume a truth assignment for an input of the **3SAT5** problem, then a minimum class Steiner tree T of G can be constructed from the horizontal edges between the top nodes of G. If a literal l is true by the truth assignment,



$\boxed{i}$ top node of R_i, $1 \leq i \leq m$ ⓘ clause node of R_{m+n+i}, $1 \leq i \leq s$
ⓘ literal node of R_{m+i}, $1 \leq i \leq$ ● Steiner node
$c_1 = \{u_1, u_2, u_3\}$; $c_2 = \{\bar{u}_1, u_2, u_3\}$; $c_3 = \{u_1, u_2, \bar{u}_3\}$; $c_4 = \{u_1, \bar{u}_2, u_3\}$
$n = 3$; $s = 4$; $m = 12n = 36$; $D = 5n + 1 = 16$

Figure 5: An example of a transformed instance of the **3SAT5** problem.

then we also add the edges $(a_1(l), b_1(l))$, $(b_i(l), b_{i+1}(l))$, $1 \leq i \leq 5$. All top classes and all literal classes are represented in this tree, since for each variable one of its two literals is true. In addition all clause classes are represented in T, since each clause contains a true literal and each clause node, corresponding to a true literal, is connected to T. Clearly the total length of T is $2n5 + (2n-1)D + n(D+5) = b$.

To prove the converse we begin by assuning there exists a class Steiner tree T', where the total length is $l(T) \leq b$. We modify T' by replacing all horizontal edges of length D on the bottom row that connect literal components, i.e. $(b_6(u_i), b_1(\bar{u}_i))$, $1 \leq i \leq n$, and $(b_6(\bar{u}_i), b_1(u_{i+1}))$, $1 \leq i \leq n-1$. First we delete one such edge $(b_6(l), b_1(l'))$ from T'. This disconnects T', but inserting either $(a_6(l), a_1(l'))$, $(a_6(l), b_6(l))$, or $(a_1(l'), b_1(l'))$ reconnects the tree. This modification is possible, because $a_6(l)$ and $a_1(l')$ are the unique nodes in their classes and are therefore represented in T'. After all possible replacements of this type we call the resulting tree T. Clearly $l(T') = l(T)$. We now know that exactly n literal components $L(l)$ may have a vertical edge $(a_i(l), b_i(l))$, $1 \leq i \leq 6$. To show this, we assume that at least $n+1$ such vertical edges are in T. This leads to a total length $l(T) > b$, since we have $m+n+s$ classes and therefore $m+n+s-1$ edges, and since we know that all top nodes must be in T. Therefore $2n-1$ horizontal edges of length D are in T as are at least $n+1$ vertical edges of length D. This implies $l(T) \geq m+n+s-1+(2n-1)(D-1)+(n+1)(D-1) = 15n^2+13n-1+s > b$, for reasonable inputs containing at least one clause, i.e. $s > 0$. So at most, there are only n literal nodes in T. On the other hand at least n literal nodes are in T, because each class must be represented by at least one node in T. Therefore for each u_i, $1 \leq i \leq n$, only the clause nodes from one of the two literal components, $L(u_i)$, $L(\bar{u}_i)$ can be connected to T. Therefore we get a valid truth assignment, if we assign the value true to all variables u_i, $1 \leq i \leq n$, where T contains the corresponding literal node $b_1(u_i)$, and false to all others. This truth assignment satisfies all clauses, because each clause is represented in a literal component $L(l)$, where l is set to true. This completes the proof. $\square$

Corollary 2 *The* RECTILINEAR CLASS STEINER TREE *(channel) problem is NP-complete for classes which consist of at most three single points.* $\square$

Corollary 2 implies that the RECTILINEAR CLASS STEINER TREE (*channel*) problem is intractable. Fortunately, as we will see in the next section, a quite natural restriction leads to a polynomial time algorithm. For each vertical cut $\{(a_i, a_{i+1}), (b_i, b_{i+1})\}$ in the corresponding class ladder graph of a class channel, $1 \leq i \leq m-1$, k_i is the number of classes that are represented in both of the resulting components. The maximum, $\max\{k_i \mid 1 \leq i \leq m-1\}$ is called the *vertical class cut.* The mentioned restriction requires, that for each class channel, the vertical class cut must be bounded by some constant k. Recall that this is natural for applications, whenever the spreads in the classes are limited, such as the case for channel routing in physical VLSI-design (see Sect. 1).

4 Linear time algorithm

In this section, we first modify the [AGH77] algorithm so that it handles an instance of the CLASS STEINER TREE ($|R_i|$=1, *ladder*) problem having exactly one node in each class. This is the starting point for our more general algorithm, which is expanded to handle an unbounded number of nodes in each class. The idea is the following: Assume that T^* is a minimum class Steiner tree for the class ladder graph (G, ρ). First we pick an arbitrary tuple $(r_1, \ldots, r_c) \in R_1 \times \cdots \times R_c$ from nodes in T^*. Each class is represented in T^* by a node in this tuple. We now modify the input to be (G, ρ'), so that each class R_i consists of the unique node r_i and all other nodes of G are Steiner nodes, i.e. $\rho'(r_i) = i$ and $\rho'(v) = 0$, $1 \leq i \leq c$, $v \in V \setminus \{r_1, \ldots, r_c\}$. If we are able to solve the CLASS STEINER TREE ($|R_i|$=1, *ladder*) problem for (G, ρ'), then we can also solve the CLASS STEINER TREE (*ladder*) problem for (G, ρ). Clearly we do not know of such a tuple $(r_1, \ldots, r_c)$, but if we were to solve the problem for all possible tuples in $R_1 \times \cdots \times R_c$, giving us the minimum tree out of all solution trees, then this tree is also a minimum class Steiner tree for the initial input (G, ρ). Since we know this problem is NP-hard, we know we will have to consider more than a polynomial number of such tuples. But we will see later a bounded vertical class cut allows us to consider only a linear number of such tuples.

4.1 A well known algorithm

In order to explain the [AGH77] algorithm (see also [AS90]) that we modified for our purpose, we present an outline. Recall that $R = \bigcup_{i=1}^{c} R_i$ and that $R_i = \{r_i\}$, $1 \leq i \leq c$, for any input (G, ρ') of the Class Steiner Tree ($|R_i|$=1, *ladder*) problem. Let $R(i) = \{a_1, \ldots, a_i, b_1, \ldots, b_i\} \cap R$ be the required nodes not on the right side of (a_i, b_i). Let $T_a(i)$, $T_b(i), T_{ab}(i)$ be subtrees of G, where minimum class Steiner tree $T_a(i)$ (resp. $T_b(i), T_{ab}(i)$) is a minimum class Steiner tree of $(G, R(i) \cup \{a_i\})$ (resp. $(G, R(i) \cup \{b_i\})$, $(G, R(i) \cup \{a_i, b_i\})$), if $R \cap R(i) \neq R$, and otherwise any minimal tree from T_a, T_b, T_{ab} is a minimum class Steiner tree for (G, R). We call such a triple $T(i) = (T_a(i), T_b(i), T_{ab}(i))$ a *minimal triple*. Clearly, a minimal tree from $T_a(m), T_b(m), T_{ab}(m)$ is a minimum class Steiner tree for (G, ρ'). Since we want to compute iteratively the minimal triple $T(m)$. Therefore, we use a function *trees* that computes a minimal triple $T(i+1)$ from a minimal triple $T(i)$ and the difference $R(i+1) \setminus R(i) = \{a_i, b_i\} \cap R$. Clearly, a class that is represented in one component of a minimal triple, is represented in each of the other components, therefore we call this class *represented in the minimal triple*. In contrast to [AGH77] the input (G, ρ') may contain *redundant vertical edges* that are incident to two Steiner nodes, i.e. an edge (a_i, b_i) where $\rho(a_i) = \rho(b_i) = 0$, $1 \leq i \leq m$. Here is the outline of our algorithm:

algorithm ST(G, ρ');
begin
  $(T_a, T_b, T_{ab})(0) :=$ *emptytreetriple*;
  for $i := 1$ **to** m **do**
    $(T_a, T_b, T_{ab})(i) :=$ *trees*$(R \cap \{a_i, b_i\}, (T_a, T_b, T_{ab})(i-1))$;
  ST:=*min*$(\{T_a(m), T_b(m), T_{ab}(m)\})$
end.

In brief, the ST algorithm begins by initializing $T(0)$ by the *emptytreetriple*, a triple of empty trees (T_a, T_b, T_{ab}), where $l(T_a) = l(T_b) = l(T_{ab}) = 0$. After scanning all m vertical edges of G, the function *min* is used to pick the shortest tree of the three trees $T_a(m), T_b(m), T_{ab}(m)$. The main part of ST algorithm is hidden in the function *trees*.

function *trees*(P, T);
begin
  if not *triple-complete*(T) **then**
    if $T =$ *emptytreetriple* **then**
      case
        (I) $a_i \in P$ and $b_i \notin P$: **begin**
          $T_{ab} := T_b := (a_i, b_i); T_a := (a_i, a_i)$ **end**;
        (II) $a_i \notin P$ and $b_i \in P$: **begin**

$T_{ab} := T_a := (a_i, b_i); T_b := (b_i, b_i)$ **end**;
(III) $a_i \in P$ and $b_i \in P$:
$T_{ab} := T_a := T_b := (a_i, b_i)$;
(IV) $a_i \notin P$ and $b_i \notin P$: **emptystatement**
end
else
case
(I) $a_i \in P$ and $b_i \notin P$: **begin**
$T_{ab} := T_b :=min(\{T_{ab} \cup (a_{i-1}, a_i) \cup (b_{i-1}, b_i),$
$T_a \cup (a_{i-1}, a_i) \cup (a_i, b_i), T_b \cup (b_{i-1}, b_i) \cup (a_i, b_i)\})$;
$T_a := T_a \cup (a_{i-1}, a_i)$ **end**;
(II) $a_i \notin P$ and $b_i \in P$: **begin**
$T_{ab} := T_a :=$ same as T_{ab} in case (I);
$T_b := T_b \cup (b_{i-1}, b_i)$ **end**;
(III) $a_i \in P$ and $b_i \in P$:
$T_{ab} := T_a := T_b :=$ same as T_{ab} in case (I);
(IV) $a_i \notin P$ and $b_i \notin P$: **begin**
$T_{ab} :=$ same as in case (I);
$T_a := T_a \cup (a_{i-1}, a_i)$;
$T_b := T_b \cup (b_{i-1}, b_i)$ **end**
end;
update(T);
trees $:= (T_a, T_b, T_{ab})$
end;

The function *trees* returns the minimal triple T unmodified, if T already contains all classes. Therefore a counter is maintained for each $T(i)$ by the procedure *update*. This counter stores the current number of classes represented in $T(i)$. The boolean function *triple-complete* checks this counter. If T does not yet contain all classes, the function *trees* distinguishes between the case where T is the *emtptytreetriple* and where it represents one or more classes. At most two edges are added to each component of T. The following theorem is easily proved by induction similar to [AGH77].

Theorem 2 *ST Algorithm solves the problem* Class Steiner Tree *($|R_i|$=1, ladder) in time $O(m)$.* □

4.2 Extending the algorithm for classes

Recall that our basic idea was to apply algorithm ST to each tuple of $R_1 \times \cdots \times R_c$, but we can do better. Again assume that $(r_1, \ldots, r_c)$ is an unknown tuple of nodes, defining a minimum class Steiner tree T^* of (G, ρ). For this tuple the ST algorithm computes a sequence $T^*(0), T^*(1), \ldots, T^*(m)$ of minimal triples for (G, ρ'). Finding a minimal solution requires only that we have a sequence of minimal triples $T(0), T(1), \ldots, T(m)$ providing that each $T(i) = (T_a(i), T_b(i), T_{ab}(i))$, $1 \leq i \leq c$, satisfies the following two conditions:

$$\begin{array}{llcll} \textit{class condition:} & \mathcal{R}_{repr}(T(i)) & = & \mathcal{R}_{repr}(T^*(i)), & \\ \textit{length condition:} & l(T_\nu(i)) & \leq & l(T^*_\nu(i)), & \nu \in \{a, b, ab\}. \end{array}$$

Here $\mathcal{R}_{repr}(T)$ is the set of classes represented by a minimal triple T. If we knew a sequence $T(0), T(1), \ldots, T(m)$ where each of its minimal triples satisfied the class and the length condition, then the component of $T(m)$ having the minimum total length would be a minimum class Steiner tree of (G, ρ). Here we assume that $T(i-1)$ is a minimal triple that satisfies the class and length conditions and $P(i) = \{a_i, b_i\} \cap \{r_1, \ldots, r_c\}$. The function *trees* then adds the same edges to $T(i-1)$ as it would have added to $T^*(i-1)$ while computing $T^*(i)$. $P(i)$ is unknown, but we do know $P(i) \subseteq \{a_i, b_i\}$: We call v an *endnode*, if $v \in R \cap \{a_i, b_i\}$ and $R_{\rho(v)} \cap \{a_1, \ldots, a_i, b_1, \ldots, b_i\} = R_{\rho(v)}$, and we call a subset $P(i)$ *valid* (denoting this by $P(i) \subseteq_{valid} \{a_i, b_i\}$), if it satisfies the following three conditions:

(1) $P(i)$ does not contain a Steiner node

(2) $P(i)$ does not contain a required node v, if $R_{\rho(v)} \in \mathcal{R}_{repr}(T(i-1))$

(3) if some required node $v \in \{a_i, b_i\}$ is an endnode and $R_{\rho(v)} \notin \mathcal{R}_{repr}(T(i-1))$, then exactly one node of its class $R_{\rho(v)}$ has to be in $P(i)$.

We do not yet know the right $P(i)$, but there are at most four possibilities for each step i, $1 \leq i \leq m$. The natural way to overcome this lack of information, is to apply the function *trees* to $T(i-1)$ and each valid $P(i)$. It turns out, that this technique leads to a sequence of sets of minimal triples $\mathcal{T}(0), \mathcal{T}(1), \ldots, \mathcal{T}(m)$ instead of a sequence of simply minimal triples. If we apply the function *trees* to each minimal triple in $\mathcal{T}(i-1)$ and each valid $P(i)$, thus computing $\mathcal{T}(i)$, and if there is a minimal triple $T(i-1) \in \mathcal{T}(i-1)$ satisfying the class and the length conditions, then there also is a minimal triple $T(i) \in \mathcal{T}(i)$ satisfying these conditions. To help visualize this, recall the observations that we made above. Unfortunately, this technique suffers from the increasing number of minimal triples, so we need to reduce the number of minimal triples in each step i.

Therefore it is important to know which classes are represented in a minimal triple in the sequence $T^*(0), T^*(1), \ldots, T^*(m)$. Let $\mathcal{R}_{closed}(i)$ be the set of classes $\{R_j \mid \{a_1, \ldots, a_i, b_1, \ldots, b_i\} \cap R_j = R_j\}$, i.e. the classes where none of the nodes lie on the right side of the vertical edge (a_i, b_i). Further we define $\mathcal{R}_{open}(i)$ to be the set of classes $\{R_j \mid \{a_1, \ldots, a_i, b_1, \ldots, b_i\} \cap R_j \neq \emptyset \wedge R_j \notin \mathcal{R}_{closed}(i)\}$, i.e. the classes where at least one node lies on the right side of the vertical edge (a_i, b_i) and at least one node does not. Obviously there is always a subset $\mathcal{R}(i) \in \mathcal{R}_{open}(i)$ such that $\mathcal{R}_{closed}(i) \cup \mathcal{R}(i)$

is the set of the classes that are represented in $T^*(i)$, $1 \leq i \leq m$. Now let $\mathcal{T}(\mathcal{R}, i) = \{T \in \mathcal{T}(i) \mid \mathcal{R}_{repr}(T) \cap \mathcal{R}_{open}(i) = \mathcal{R}\}$ (we write $\mathcal{R}$ instead of $\mathcal{R}(i)$ for simplicity). The iterative computation of $\mathcal{T}(i)$ guarantees that for each $\mathcal{R} \in \mathcal{R}_{open}(i)$, $\mathcal{T}(\mathcal{R}, i) \neq \emptyset$. Moreover, we define $\mathcal{T}_\nu(\mathcal{R}, i)$, $\nu \in \{a, b, ab\}$, to be the set of the ν-components in the minimal triples of $\mathcal{T}(\mathcal{R}, i)$. Our technique for bounding the number of the minimal triples during each iteration is to replace for each subset $\mathcal{R} \subseteq \mathcal{R}_{open}(i)$ the set $\mathcal{T}(\mathcal{R}, i)$ with the set $\{T'(\mathcal{R}, i)\}$, where each component of the minimal triple $T'(\mathcal{R}, i)$ is of minimum total length, i.e. $T'(\mathcal{R}, i) = (\min \mathcal{T}_a(\mathcal{R}, i), \min \mathcal{T}_b(\mathcal{R}, i), \min \mathcal{T}_{ab}(\mathcal{R}, i))$. Obviously even after this modification, $\mathcal{T}(i)$ still contains a minimal triple that satisfies the class and the length conditions. Finally, since $\mathcal{R}_{open}(m) = \emptyset$ we have only one $T'(\emptyset, m)$ in the end, so all we need to do is to extract the component of minimum total length. This tree then is a minimum class Steiner tree for the input (G, ρ). The following outline of the algorithm CST summarizes our ideas:

algorithm CST(G, ρ);
begin
  $(T_a, T_b, T_{ab})(0) := \mathit{emptytreetriple}$; $\mathcal{R}_{open}(0) := \emptyset$;
  for $i := 1$ **to** m **do**
    $\mathcal{T}(i) := \emptyset$;
    $\mathit{compute}(\mathcal{R}_{open}(i))\mathit{from}(\mathcal{R}_{open}(i-1))$;
    $\{$*Block 1:*$\}$
    foreach $T \in \mathcal{T}(i-1)$ **do**
      foreach $P \subseteq_{valid} \{a_i, b_i\}$ **do**
        $\mathcal{T}(i) := \mathcal{T}(i) \cup \mathit{trees}(T, P)$;
    $\{$*Block 2:*$\}$
    foreach $\mathcal{R} \in \mathcal{R}_{open}(i)$ **do**
      $\mathcal{T}(i) := (\mathcal{T}(i) \setminus \mathcal{T}(\mathcal{R}, i)) \cup \{\mathit{min}(\mathcal{T}_a(\mathcal{R}, i)), \mathit{min}(\mathcal{T}_b(\mathcal{R}, i)), \mathit{min}(\mathcal{T}_{ab}(\mathcal{R}, i))\}$;
  CST$:= \mathit{min}(\mathcal{T}_a(\emptyset, m) \cup \mathcal{T}_b(\emptyset, m) \cup \mathcal{T}_{ab}(\emptyset, m))$.

Executing this algorithm CST may take more than polynomial time, since the Class Steiner Tree (*ladder*) problem is NP-hard. But we will prove that the CST algorithm needs only linear time, if we restrict its input to class ladder graphs that have a vertical class cut bounded by some constant k. This implies that the number of open classes is bounded for each i in the input class ladder graph, where $|\mathcal{R}_{open}(i)| \leq k$, $1 \leq i \leq m$.

Theorem 3 *The CST Algorithm solves the* Class Steiner Tree *(ladder) problem in time $O(mk2^k)$.*

Proof: From its construction, it can clearly be seen that the algorithm is correct. The following is a proof of the time complexity stated above. First we need to define the data structures we will use for minimal triples and component trees (for all standard data structures, see e.g. [OW90]). A minimal triple $T(i)$ is a record containing: (1) for each component tree $T_\nu(i)$, $\nu \in \{a, b, ab\}$, a pointer to the edge list and the total length of $T_\nu(i)$, (2) a linked list of the open classes represented in $T(i)$, i.e. $\mathcal{R}_{repr}(T(i)) \cap \mathcal{R}_{open}(i)$, sorted in the order induced by ρ, (3) a counter for the current number of classes represented in $T(i)$, i.e. $|\mathcal{R}_{repr}(T(i))|$, and (4) an additional pointer for creating lists of minimal triples (see Fig 6). To store the edge lists of the

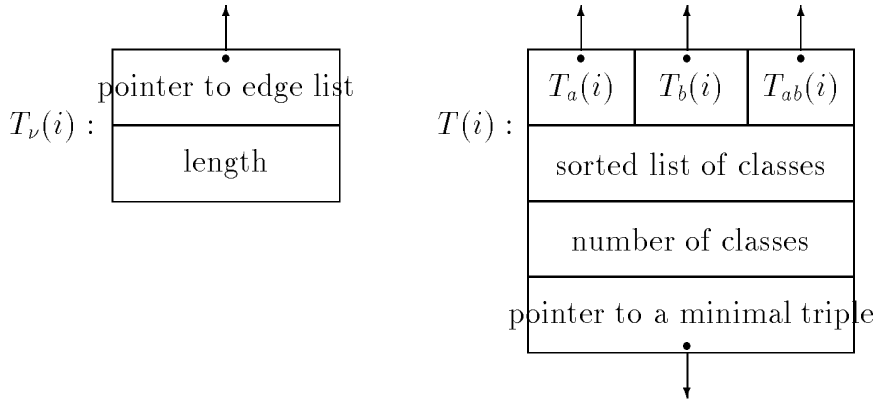


Figure 6: A minimal triple $T(i)$ and its components $T_\nu(i)$.

component trees, we use a single rooted *edge tree*, where the nodes represent lists of edges. If the T_ν pointer of a minimal triple T points to a node v in the edge tree, then the edges of T_ν can be obtained by following the path from v to the root. An edge tree begins as a root containing an empty list of edges. Each of the component trees in *emptytreetriple* points to this root and has a length 0, the list of classes is empty, and the number of represented classes is 0. Whenever the function *trees* is called with a minimal triple T, it creates at most three new leaves in the edge tree then returns a modified copy of T. Therefore the most time intensive operation is copying the list representing $\mathcal{R}_{repr}(T(i)) \cap \mathcal{R}_{open}(i)$ then to modify it in the procedure *update*, all of which is possible in time $O(k)$. After the modification we decide in time $O(1)$, whether a class should be closed or not. To do this we use the boolean array *endnode* to encode whether a node is an endnode. This array is initialized by prescanning the class ladder graph backwards, which takes time $O(m)$. We implement $\mathcal{R}_{open}(i)$ using a list of classes which is sorted in the order induced by ρ. It then tames time $O(k)$ to compute $\mathcal{R}_{open}(i)$ from $\mathcal{R}_{open}(i-1)$ takes time $O(k)$, if we also use the array *endnode* here.

The data structure representing the sets of minimal triples $\mathcal{T}(i)$ and $\mathcal{T}(\mathcal{R}, i)$, $1 \le i \le m$, $\mathcal{R} \subseteq \mathcal{R}_{open}(i)$, is built around an array $[0, \ldots, 2^k - 1]$ which is used as a hash table. Our hash function is an injective function $2^{\mathcal{R}_{open}(i)} \to \{0, \ldots, 2^k - 1\}$ which operates as follows: We store an additional weight as well as the classes in the list representing $\mathcal{R}_{open}(i)$. W.l.o.g.

let $R_1, \ldots, R_{|\mathcal{R}_{open}(i)|}$ be the list of classes, then the weights are defined by $w(R_j) = 2^{j-1}$, $1 \leq j \leq |\mathcal{R}_{open}(i)|$. The iterative computation of $\mathcal{R}_{open}(i)$ and the recomputation of the related weights together take time $O(k)$ for each step i, $1 \leq i \leq m$. We use the hashfunction $\mathcal{R} \mapsto \sum_{j=1}^{|\mathcal{R}_{open}(i)|} w(R_j)$ for numbering the subsets of $\mathcal{R} \subseteq \mathcal{R}_{open}(i)$, so each subset of the set of open classes refers to a different field in the hash table. These fields serve as headers for the related lists $\mathcal{T}(\mathcal{R}, i)$. When we create such a structure for minimal triples, the fields of the hash table represent empty lists. This takes time $O(2^k)$. To insert a minimal triple $T(i)$ we compute the hash function for $\mathcal{R} = \mathcal{R}_{repr}(T(i)) \cap \mathcal{R}_{open}(i)$ and insert $T(i)$ into the list $\mathcal{T}(\mathcal{R}, i)$. $\mathcal{T}(i)$ is the union of all lists referenced by the hash table. Whenever $|\mathcal{R}_{open}(i)| < k$, that means that some of the fields in the hash table do not correspond to some $\mathcal{R} \subseteq \mathcal{R}_{open}(i)$. To handle this case we maintain a separate list containing pointers to all $\mathcal{T}(\mathcal{R}, i)$ that are not empty, i.e. where at least one minimal triple was inserted. Creating and maintaining this list takes time $O(1)$ per insertion. Therefore a complete insertion of a minimal triple in the data structure takes time $O(k)$, because the computation of the hash function is the most time consuming step (see Fig 7). Its now easy to see how we can

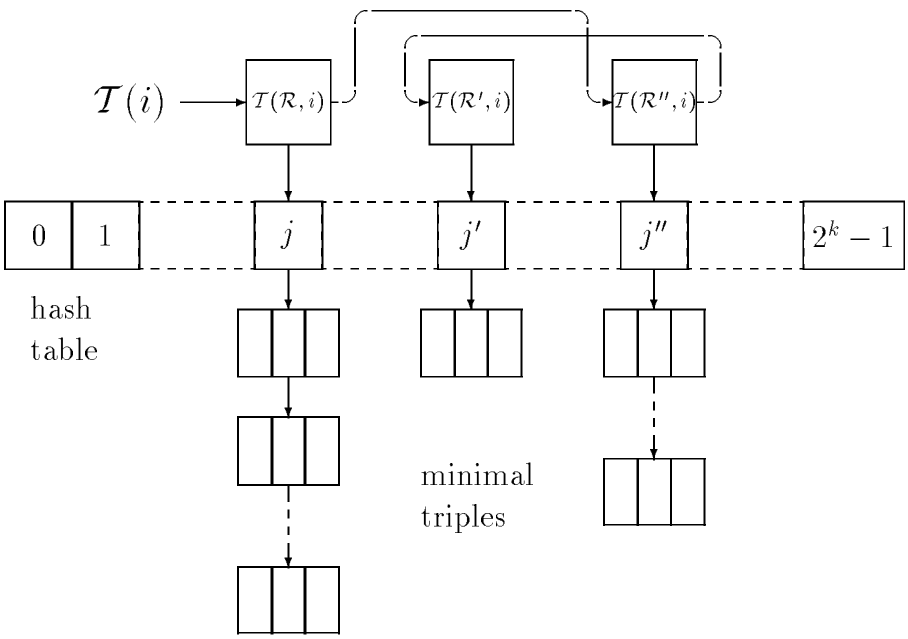


Figure 7: The data structure for the sets $\mathcal{T}(i)$, $\mathcal{T}(\mathcal{R}, i)$

compute all valid subsets $P(i) \subseteq_{valid} \{a_i, b_i\}$ in time $O(k)$ by using the array *endpoint* and the list representing $\mathcal{R}_{repr}(T(i)) \cap \mathcal{R}_{open}(i)$ in the record representing $T(i)$. We know that $\mathcal{T}(i-1)$ contains at most 2^k minimal triples at the begining of each execution of Block 1 in the algorithm CST. Altogether Block 1 takes time $O(k2^k)$.

In Block 2 we reduce the maximum number of minimal triples in $\mathcal{T}(i)$ from $4 \cdot 2^k$ to 2^k. The factor 4 originates from the fact that there are always at most four possible compositions for any subset $P(i) \subseteq_{valid} \{a_i, b_i\}$. Using our previously defined data structure for sets of minimal triples, we compute a minimal triple for each set $\mathcal{T}(\mathcal{R}, i)$ such that each component is a minimum taken from all minimal triples in $\mathcal{T}(\mathcal{R}, i)$. We just have to go once through the list $\mathcal{T}(i)$ and each of the sublists $\mathcal{T}(\mathcal{R}, i)$. This means Block 2 takes time $O(2^k)$ and the total time complexity of the CST algorithm is $O(mk2^k)$. □

Corollary 3 *The CST algorithm solves the* RECTILINEAR CLASS STEINER TREE *(channel) problem in time $O(mk2^k)$.* □

Moreover our algorithm is of linear time complexity in solving the problems CLASS STEINER TREE (*ladder*) and RECTILINEAR CLASS STEINER TREE (*channel*), if the vertical class cut of the input is bounded by a constant k.

5 Conclusion

We have presented a specific algorithm for solving a problem, which can be summarized as follows: Our input is a set of non intersecting intervals on two parallel lines in the Manhattan-plane and this set of intervals is in turn partitioned into subsets. The subsets of points falling within each of these intervals are called required classes. The task is then to find a minimum rectilinear class Steiner tree that contains at least one point from each required class. Our algorithm works in linear time, if we have a constant bounded vertical class cut, but in general the problem is NP-hard.

Our result was recently extended by [Rei91] for classes that lie on the border of a rectangle, called *class switchbox*. A further possibility to generalize our result would be to allow classes on several parallel lines. We would expect the complexity of the resulting algorithm exponential with regard to the number of lines.

Acknowledgment

We would like to warmly thank M. Sarrafzadeh for motivating us in our work and for his valuable comments.

References

[AGH77] A. V. Aho, M. R. Garey, and F. K. Hwang. Rectilinear Steiner trees: Efficient special-case algorithms. *Networks*, 7:37–58, 1977.

[AS90] P. K. Agarwal and M.-T. Shing. Algorithms for special cases of rectilinear Steiner trees: 1. Points on the boundary of a rectilinear rectangle. *Networks*, 20:453–485, 1990.

[GJ77] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[Han66] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM J. Appl. Math.*, 14(2):255–265, 1966.

[Ihl91] E. Ihler. Bounds on the quality of approximate solutions to the Group Steiner Problem. In *Graph-Theoretic Concepts in Computer Science, WG90*, volume 484 of *Lecture Notes in Computer Science*, pages 109–118. Springer, 1991.

[Ihl92] E. Ihler. The complexity of approximating the Class Steiner Tree problem. In *Graph-Theoretic Concepts in Computer Science, WG91*, volume 570 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 1992.

[KPS90] B. Korte, H. J. Prömel, and A. Steger. Steiner trees in VLSI-layout. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows, and VLSI-Layout*, pages 185–214. Springer, 1990.

[Len90] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley & Sons, 1990.

[OW90] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen.* BI-Wiss.-Verl., 1990.

[Rei91] G. Reich. Shortest trees for intervals of points in the switchbox. Technical report, Institut für Informatik, Universität Freiburg, 1991.

[RW89] G. Reich and P. Widmayer. Approximate minimum spanning trees for vertex classes. Technical report, Institut für Informatik, Universität Freiburg, 1989.

[SW91] M. Sarrafzadeh and C.K. Wong. New directions in the rectilinear Steiner tree problem. Technical report, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, and IBM Watson Research Center, Yorktown Heights, New York, 1991.