How Many Steiner Terminals Can You Connect in 20 Years?

Ralf Borndörfer, Nam-Dũng Hoang, Marika Karbstein, Thorsten Koch, and Alexander Martin

Abstract Steiner trees are constructed to connect a set of terminal nodes in a graph. This basic version of the Steiner tree problem is idealized, but it can effectively guide the search for successful approaches to many relevant variants, from both a theoretical and a computational point of view. This article illustrates the theoretical and algorithmic progress on Steiner tree type problems on two examples, the Steiner connectivity and the Steiner tree packing problem.

1 Introduction

The *Steiner tree problem* (STP) is one of the showcases of combinatorial optimization. It deals with finding a best connection of a number of vertices in a network and can formally be stated as follows:

> *Given a weighted graph $G = (V, E, c)$ and a non-empty set of vertices $T \subseteq V$ called* terminals, *find an edge set S^* such that $(V(S^*), S^*)$ is a tree of minimal weight that spans T.*

The STP is extensively covered in the literature, see [33, 35] for an introduction and [32] for a state-of-the-art survey on models and solution techniques. Many

R. Borndörfer (✉) · M. Karbstein · T. Koch
Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, 14195 Berlin, Germany
e-mail: borndoerfer@zib.de

M. Karbstein
e-mail: karbstein@zib.de

T. Koch
e-mail: koch@zib.de

N.-D. Hoang
Faculty of Mathematics, Mechanics, and Informatics, Vietnam National University, 334 Nguyen Trai, Hanoi, Vietnam
e-mail: hoangnamdung@hus.edu.vn

A. Martin
Department Mathematik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstr. 11, 91058 Erlangen, Germany
e-mail: alexander.martin@math.uni-erlangen.de

papers claim real-world applications, especially in VLSI-*design* and *wire-routing*, but also in *telecommunication and traffic network design*. Such applications usually refer to generalizations of the STP, which still require to connect a number of terminals, but by more than one tree, by something that is not precisely a tree, a constrained tree, etc. Such stipulations have always made Steiner trees a lively and exciting research topic for theoreticians and practitioners alike. Theoretical progress on the STP will typically serve as a blueprint to approach a particular Steiner tree problem; this knowledge is supplemented by specific developments, which in turn can advance the general theory. This is the Martin Grötschel way of optimization.

We illustrate (t)his approach in this article using two examples, the Steiner connectivity problem (SCP) and the Steiner tree packing problem (STPP). The SCP, discussed in Sect. 2, generalizes the STP by connecting the terminals using a set of paths instead of edges; this is motivated by line planning problems in public and rail transit, see [3]. The Steiner connectivity problem serves as an example that often results can be transferred from the basic Steiner tree problem to a more general setting, however, not all results and not always in a completely straightforward way. The STPP, studied in Sect. 3, deals with connecting several sets of terminals by several trees that cannot share edges; this models the internal wiring in a chip. The Steiner tree packing problem is difficult because it integrates a connection and a packing aspect; it is particularly well suited to illustrate the algorithmic progress of the last 20 years. However, chips are much larger now and the challenge has also grown. Steiner trees will therefore remain an exciting topic, at least until Martin Grötschel's 100th birthday—can there be a better message?

2 The Steiner Connectivity Problem

Transportation and networks have always been two of Martin Grötschel's favorite topics and it was therefore inevitable that at some point he would get involved in a research project on designing the line system of a public transportation system. Identifying origin-destination demand points with terminals, and lines with hyperedges, this leads directly to a hypergraph version of the Steiner tree problem, which we denote the Steiner connectivity problem, see [4, 23]. How difficult is this? In such a case, Martin Grötschel will always advocate a thorough investigation with a careful look at details, which can or cannot make a big difference. In particular, when it comes to graphs, digraphs, and hypergraphs, he becomes furious about "sloppy notation" that doesn't distinguish between uv, (u, v), and $\{u, v\}$, because results do *not* automatically carry over between these cases. Martin Grötschel is correct, and the ability to seamlessly shift his attention from little details to the grand picture is without doubt one of his greatest strengths.

A formal description of the *Steiner connectivity problem* (SCP) is as follows. We are given an undirected graph $G = (V, E)$, a set of *terminal nodes* $T \subseteq V$, and a set of elementary *paths* $\mathcal{P}$ in G. The paths have nonnegative costs $c \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$. The problem is to find a set of paths $\mathcal{P}' \subseteq \mathcal{P}$ of minimal cost $\sum_{p \in \mathcal{P}'} c_p$ that *connect the*

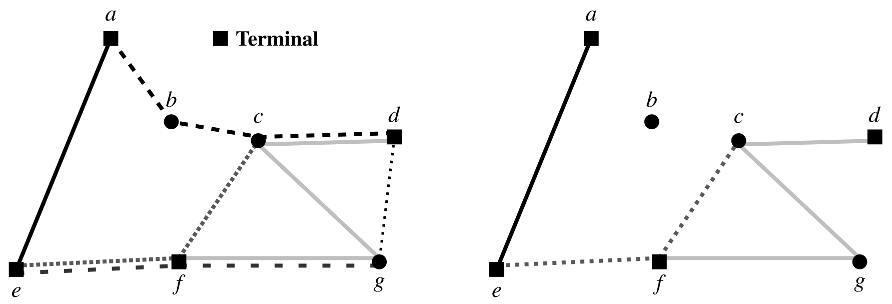

Fig. 1 Example of a Steiner connectivity problem. *Left*: A graph with four terminal nodes ($T = \{a, d, e, f\}$) and six paths ($\mathcal{P} = \{p_1 = (ab, bc, cd), p_2 = (ef, fg), p_3 = (ae), p_4 = (ef, fc), p_5 = (gd), p_6 = (fg, gc, cd)\}$). *Right*: A feasible solution with three paths ($\mathcal{P}' = \{p_3, p_4, p_6\}$)

terminals, i.e., such that for each pair of distinct terminal nodes $t_1, t_2 \in T$ there exists a path q from t_1 to t_2 in G such that each edge of q is covered by at least one path of $\mathcal{P}'$. We can assume w.l.o.g. that every edge is covered by a path, i.e., for every $e \in E$ there is a $p \in \mathcal{P}$ such that $e \in p$; in particular, G has no loops. Figure 1 gives an example of a Steiner connectivity problem and a feasible solution. Identifying the paths $\mathcal{P}$ with hyperedges in the hypergraph $(V, \mathcal{P})$ leads to an equivalent statement in terms of hypergraphs, in which the terminals have to be connected by a minimum cost set of hyperedges.

The Steiner connectivity problem is a good example for an extension study. Indeed, many results on Steiner trees can be generalized to the hypergraph setting, but not all, and not all directly. We illustrate this for two cases, namely, the 2-terminal case, and the "all-terminal case", where all nodes are terminals. The latter generalizes spanning trees to "spanning sets", but, in contrast to the graphical case, is $\mathcal{NP}$-hard. The first deals with finding a shortest hyperpath; we will see that structural properties of shortest paths carry over to this situation by proving the companion theorem to Menger's theorem for hypergraphs. A further polyhedral and computational investigation of the Steiner connectivity problem can be found in [4].

2.1 The All-Terminal Case and the Greedy Algorithm

The all-terminal case $T = V$ of the Steiner tree problem is a simple minimum spanning tree problem. In the Steiner connectivity setting, however, this case is hard. This is because of a strong relation between the SCP and the set covering problem that will be discussed now.

Proposition 1 *The Steiner connectivity problem is $\mathcal{NP}$-hard for $T = V$, even for unit costs.*

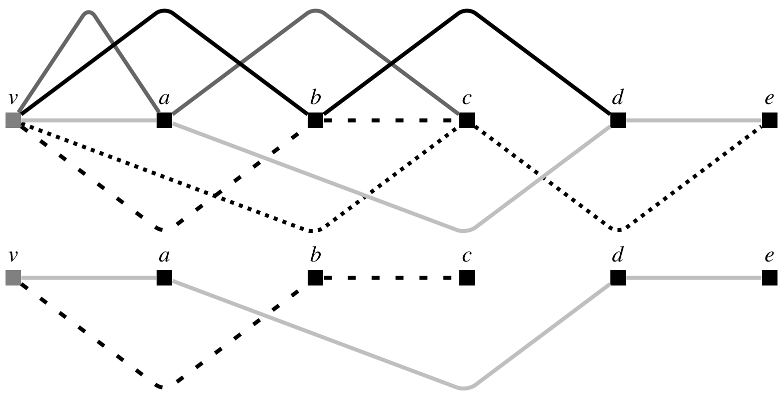

Fig. 2 *Top*: A Steiner connectivity instance corresponding to a set covering instance with $S = \{a, b, c, d, e\}$ and $\mathcal{M} = (\{a, c\}, \{b, d\}, \{b, c\}, \{c, e\}, \{a, d, e\})$. *Bottom*: A minimal solution for the Steiner connectivity problem corresponding to the minimal cover $\mathcal{M}' = (\{b, c\}, \{a, d, e\})$

Proof We reduce the set covering problem to the all-terminal Steiner connectivity problem. In a set covering problem we are given a finite set S and a set $\mathcal{M} \subseteq 2^S$. The problem is to find a subset $\mathcal{M}' \subseteq \mathcal{M}$ of minimal cardinality $|\mathcal{M}'|$ such that for all $s \in S$ there exists an $M \in \mathcal{M}'$ with $s \in M$.

Given a set covering instance, we define an all-terminal Steiner connectivity instance in a graph $G = (V, E)$ as follows: The nodes are $V = S \cup \{v\} = T$ with v being one extra node. Let us write $V = \{s_0, s_1, s_2, \ldots\}$, where $v = s_0$. All nodes are terminal nodes. We first assume that G is a complete graph and later remove all edges that are not covered by paths after their construction. For each set $M \in \mathcal{M}$ order the elements in M arbitrarily and construct a path beginning in node v and passing through all nodes of M in the given order, compare with Fig. 2. The cost of each such path is 1.

It is easy to see that a cover $\mathcal{M}'$ with at most k elements exists if and only if a set of paths exists that connects V with cost at most k, $k \geq 0$. □

Corollary 1 *SCP is strongly $\mathcal{NP}$-hard for $|T| = |V| - k$, k constant.*

Proof We add k isolated nodes to the graph G in the proof of Proposition 1. □

Proposition 2 *There is no polynomial time α-approximation algorithm for SCP with $\alpha = \gamma \cdot \log |V|$, $\gamma \leq 1$, unless $\mathcal{P} = \mathcal{NP}$.*

Proof The transformation in Proposition 1 is approximation preserving, since there exists a cost preserving bijection between the solutions of the set covering instance and its corresponding Steiner connectivity instance. It has been shown that the set covering problem is not approximable in the sense that there exists no polynomial time approximation algorithm with approximation factor smaller than logarithmic (in the number of nodes) unless $\mathcal{P} = \mathcal{NP}$, see Feige [10]. □

The proof of Proposition 1 shows that the set covering problem can be transformed to the all-terminal Steiner connectivity problem. On the other hand, the all-terminal Steiner connectivity problem can be interpreted as a submodular set covering problem. Recall that a function $z : 2^N \to \mathbb{R}$ from a set $N = \{1, \dots, n\}$ to the reals is *submodular* if the following inequalities hold:

$$z(A) + z(B) \geq z(A \cup B) + z(A \cap B) \quad \forall A, B \subseteq N.$$

The problem

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j : z(S) = z(N) \right\}$$

is called the *submodular set covering problem* if z is a nondecreasing submodular function. We call this problem *integer-valued* if $z : 2^N \to \mathbb{Z}$. Let $N = \mathcal{P}$ and define for $\mathcal{P}' \subseteq \mathcal{P}$

$$z(\mathcal{P}') = |V| - \text{number of connected components in } \big(V, E(\mathcal{P}')\big),$$

where $E(\mathcal{P}')$ denotes the set of edges covered by the paths in $\mathcal{P}'$; $z(\mathcal{P}')$ can be interpreted as the maximum number of edges in $(V, E(\mathcal{P}'))$ containing no cycle. Note that this definition corresponds to the rank function for an edge set in a graphical matroid, i.e., $z(\mathcal{P}') = \text{rank}(E(\mathcal{P}'))$, see, e.g., Oxley [31]. The function z is, therefore, a nondecreasing, integer-valued, submodular set function; this follows since $E(\mathcal{P}') \subseteq E(\mathcal{P}'')$ for $\mathcal{P}' \subseteq \mathcal{P}''$. Note that $z(\mathcal{P}') = z(N) = z(\mathcal{P}) = |V| - 1$ means that $\mathcal{P}'$ connects V. Hence, the Steiner connectivity problem can be seen as an integer-valued submodular set covering problem. We have $z(p) = |p|$ for $p \in \mathcal{P}$ and $z(\emptyset) = 0$. For such problems, there exists a greedy algorithm that works fairly well:

Theorem 1 (Wolsey [37], 1982) *There is a greedy heuristic that gives an* $H(k) = \sum_{i=1}^{k} \frac{1}{i}$ *approximation guarantee for integer-valued submodular set covering problems, where* $k = \max_{j \in N} z(\{j\}) - z(\emptyset)$.

Such a greedy algorithm therefore also gives an approximation guarantee of $H(k) = \sum_{i=1}^{k} \frac{1}{i}$ for the all-terminal Steiner connectivity problem if all paths contain at most k edges. This bound is asymptotically optimal, see Feige [10] and compare with Proposition 2.

Wolsey's result generalizes an earlier one of Chvátal [8] who showed that a greedy algorithm gives an $H(k) = \sum_{i=1}^{k} \frac{1}{i}$ approximation guarantee for the set covering problem, where k is the largest column sum. Chvátal's reasoning can be extended to an elementary proof involving combinatorial counting arguments that are interesting in their own right. It goes as follows.

The proof analyzes the greedy heuristic in Algorithm 1. This procedure starts in an initial state in which each single node forms a smallest possible *(connected)*

Algorithm 1: Greedy heuristic for the SCP

```
Input  : A connected graph $G = (V, E)$, a set of paths $\mathscr{P}$ with costs $c \in \mathbb{R}^{\mathscr{P}}_{\geq 0}$.
Output: A set of paths $\mathscr{P}' \subseteq \mathscr{P}$ that connects all nodes.
1 $B^0 := \{\{v\} | v \in V\}$, $\mathscr{P}^0 := \emptyset$, $i := 1$
2 while $|B^{i-1}| > 1$ do
3     $p(i) := \arg\min_{p \in \mathscr{P}} \{\frac{c_p}{N(p,i)} : N(p, i) > 0\}$
4     $\mathscr{P}' := \mathscr{P}^i := \mathscr{P}^{i-1} \cup \{p(i)\}$
5     $B^i := (B^{i-1} \setminus \{b_1, \ldots, b_j\}) \cup \{b_1 \cup \ldots \cup b_j\}$ with
      $\{b_1, \ldots, b_j\} := \{b \in B^{i-1} : p(i) \in \mathscr{P}, p(i) \cap b \neq \emptyset\}$
6     $i := i + 1$
7 end
```

component. The algorithm then chooses in each iteration a path that minimizes the ratio of cost over the number of components that are connected by the path minus one. These connected components are merged into a new connected component. The algorithm terminates when everything has been merged into a single connected component.

We use the following notation. Let B^i be the set of connected components and $\mathscr{P}^i$ the set of chosen paths after iteration i of Algorithm 1. Note that in each iteration at least two connected components are merged, i.e., $|B^i|$ decreases strictly with increasing i. Let us further denote by

$$\begin{aligned} N(p,i) &= z\big(\mathscr{P}^{i-1} \cup \{p\}\big) - z\big(\mathscr{P}^{i-1}\big) \\ &= \operatorname{rank}\big(E\big(\mathscr{P}^{i-1} \cup \{p\}\big)\big) - \operatorname{rank}\big(E\big(\mathscr{P}^{i-1}\big)\big) \\ &= \text{no. of conn. comp. in } \big(V, E\big(\mathscr{P}^{i-1}\big)\big) \\ &\quad - \text{no. of conn. comp. in } \big(V, E\big(\mathscr{P}^{i-1} \cup \{p\}\big)\big) \end{aligned}$$

the *component reduction number* of path p and iteration i, i.e., if p were chosen in iteration i, the total number of connected components would reduce by $N(p,i)$. Note that $N(p,i)$ is nonincreasing for increasing i, i.e., $N(p,1) \geq \ldots \geq N(p,n)$ where n is the last iteration of Algorithm 1. Let $\mathscr{P}' = \{p(1), \ldots, p(n)\}$. Algorithm 1 then computes a solution of cost $c(\mathscr{P}') = \sum_{i=1}^{n} c_{p(i)}$. Let, further, $\mathscr{P}_{\text{opt}} = \{o_1, \ldots, o_m\}$ be an optimal V-connecting set. Finally, we denote by $H(k) = \sum_{i=1}^{k} \frac{1}{i}$ the sum of the first k terms of the harmonic series.

In order to analyze the greedy algorithm, we derive a lemma concerning the sum of the component reduction numbers of the optimal paths in iteration $i \in \{1, \ldots, n\}$. This number is always greater than or equal to the sum of the component reduction numbers of the paths that are chosen by the greedy algorithm.

Lemma 1 *In Algorithm 1 holds*

$$\sum_{o\in\mathcal{P}_{\mathrm{opt}}} N(o,i) \geq \sum_{j=i}^{n} N\big(p(j),j\big) \quad \forall i=1,\dots,n. \tag{1}$$

Proof Consider the right hand side of inequality (1). We get

$$\begin{aligned}\sum_{j=i}^{n} N\big(p(j),j\big) &= z\big(\mathcal{P}^{i-1}\cup\{p(i)\}\big)-z\big(\mathcal{P}^{i-1}\big)+z\big(\mathcal{P}^{i}\cup\{p(i+1)\}\big)-z\big(\mathcal{P}^{i}\big)\\ &\quad+\ldots+z\big(\mathcal{P}^{n-1}\cup\{p(n)\}\big)-z\big(\mathcal{P}^{n-1}\big)\\ &= z\big(\mathcal{P}^{n}\big)-z\big(\mathcal{P}^{i-1}\big)=|V|-1-z\big(\mathcal{P}^{i-1}\big)\\ &= \text{no. of conn. comp. in } \big(V,E\big(\mathcal{P}^{i-1}\big)\big)-1.\end{aligned}$$

The claim then follows since each V-connecting set has to connect all connected components in $(V, E(\mathcal{P}^{i-1}))$. □

Proposition 3 *The greedy Algorithm 1 gives an $H(k)$ approximation guarantee for Steiner connectivity problems, where $k=\max_{p\in\mathcal{P}}|p|$ is the maximum path length, i.e.,*

$$c\big(\mathcal{P}'\big) \leq \sum_{p\in\mathcal{P}_{\mathrm{opt}}} H\big(|p|\big)c_p \leq H(k)c(\mathcal{P}_{\mathrm{opt}}).$$

Proof The idea of the proof is as follows. In a first step (assignment), we assign the path $p(i)\in\mathcal{P}'$ (added to $\mathcal{P}'$ in iteration $i=1,\dots,n$ in Algorithm 1) to a subset of optimal paths $O(i)\subseteq\mathcal{P}_{\mathrm{opt}}$. In a second step (bounding), we show that the cost of path $p(i)$ can be bounded from above by the cost of the paths in $O(i)$. In a third step (summation), we show that the cost of each path of the optimal solution $\mathcal{P}_{\mathrm{opt}}$ is used at most $H(k)$ times in the bounding step.

1. *Step: Assignment*. Consider Algorithm 2. It assigns to each path $p(i)$, $i=1,\dots,n$, of the greedy algorithm (passed in reverse order), a set $O(i)\subseteq\mathcal{P}_{\mathrm{opt}}$ of optimal paths. The component reduction value $N(p(i),i)$ for each path $p(i)$, $i=1,\dots,n$, is distributed to the paths $o\in O(i)$. To this purpose values $\upsilon(o,i)$ are computed such that $\upsilon(o,i)>0\Leftrightarrow o\in O(i)$. More precisely, in each iteration i a set $O(i)\subseteq\mathcal{P}_{\mathrm{opt}}$ is chosen such that

$$\sum_{o\in O(i)} \upsilon(o,i)=N\big(p(i),i\big) \quad \forall i=1,\dots,n. \tag{2}$$

Here, the values $\upsilon(o,i)$, $o\in O$, $i=1,\dots,n$, satisfy the following condition

$$\sum_{j=i}^{n} \upsilon(o,j)\leq N(o,i) \quad \forall o\in O(i), i=1,\dots,n. \tag{3}$$

Algorithm 2: Assigning optimal paths to the paths of the greedy algorithm

```
$\upsilon(o,i) := 0, \forall o \in \mathcal{P}_{\text{opt}}, \forall i = 1,\dots,n$

for $i = n$ to 1 do
    $O(i) := \emptyset, z := 0$
    while $z < N(p(i),i)$ do
        Choose $o \in \mathcal{P}_{\text{opt}} \setminus O(i)$ with $N(o,i) - \sum_{j=i}^{n} \upsilon(o,j) > 0$
        $\upsilon(o,i) := \min\{N(o,i) - \sum_{j=i}^{n} \upsilon(o,j), N(p(i),i) - z\}$
        $z := z + \upsilon(o,i)$
        $O(i) := O(i) \cup \{o\}$
    end
end
```

Lemma 1 ensures that these values $\upsilon(o,i)$, $i = 1,\dots,n$, exist.

2. *Step: Bounding*. Consider the path $p(i)$, $i \in \{1,\dots,n\}$, in iteration i of Algorithm 1 and the corresponding set $O(i) = \{o_1,\dots,o_h\}$ defined in Algorithm 2. Path $p(i)$ achieves the minimum in the ratio test in Step 3 of Algorithm 1. Using this fact and equation (2), the cost of $p(i)$ can be bounded as follows

$$\left.\begin{array}{l} \left.\begin{array}{c} \frac{c_{p(i)}}{N(p(i),i)} \le \frac{c_{o_1}}{N(o_1,i)} \\ \vdots \\ \frac{c_{p(i)}}{N(p(i),i)} \le \frac{c_{o_1}}{N(o_1,i)} \end{array}\right\} \upsilon(o_1,i) \text{ times} \\ \qquad\qquad\vdots \\ \left.\begin{array}{c} \frac{c_{p(i)}}{N(p(i),i)} \le \frac{c_{o_h}}{N(o_h,i)} \\ \vdots \\ \frac{c_{p(i)}}{N(p(i),i)} \le \frac{c_{o_h}}{N(o_h,i)} \end{array}\right\} \upsilon(o_h,i) \text{ times} \end{array}\right\} N\big(p(i),i\big) \text{ times.}$$

Hence, we have $c_{p(i)} \le \sum_{o \in O(i)} \frac{c_o}{N(o,i)} \upsilon(o,i)$.

3. *Step: Summation.* We finally consider how often the costs of a path $o \in \mathcal{P}_{\text{opt}}$ are used in the bounding step. We have $N(p,i) \in \{0,1,2,\dots,|p|\}$, $\forall p \in \mathcal{P}$, $i = 1,\dots,n$, hence, the total cost for a path $o \in O \subseteq \mathcal{P}$ in the bounding step can be rewritten as

$$\sum_{i=1}^{n} \frac{c_o}{N(o,i)} \upsilon(o,i) = \frac{c_o}{1} a_1 + \frac{c_o}{2} a_2 + \ldots + \frac{c_o}{|o|} a_{|o|}. \tag{4}$$

Here, the coefficients

$$a_k = \sum_{\substack{i=1 \\ N(o,i)=k}}^{n} \upsilon(o,i), \quad k = 1,\dots,|o|,$$

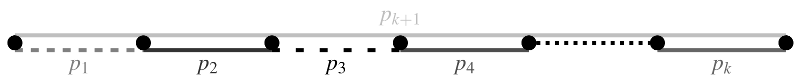

Fig. 3 Worst case Steiner connectivity example for the greedy algorithm

are sums of the values $\upsilon(o,i)$. Note that $N(o,i)$ is increasing for decreasing i. Let $s_k \in \{1,\dots,n\}$ be the smallest iteration index of Algorithm 1 such that $N(o,s_k)=k$, $k=1,\dots,|o|$ (for $k=|o|$ we have $s_k=1$), if such an index exists. Then equation (3) implies

$$a_k \le \sum_{j=1}^{k} a_j \le k, \quad k=1,\dots,|o|. \tag{5}$$

This follows immediately if $a_k=0$, i.e., if $N(o,i)\neq k$ for all $i=1,\dots,n$. Otherwise

$$\sum_{j=1}^{k} a_j = \sum_{j=1}^{k} \sum_{\substack{i=1\\N(o,i)=j}}^{n} \upsilon(o,i) = \sum_{i=s_k}^{n} \upsilon(o,i) \le N(o,s_k) = k.$$

The term $\frac{c_o}{k}$ decreases with increasing k and is maximal for $k=1$, and (5) implies $a_1 \le 1$. This means that the sum (4) is maximal for $a_1=1$. Repeating this argument for a_2, etc., the sum (4) is maximal if all coefficients a_k, $k=1,\dots,|o|$, are 1. We then get

$$\sum_{i=1}^{n} \frac{c_o}{N(o,i)} \upsilon(o,i) \le \frac{c_o}{1} + \frac{c_o}{2} + \ldots + \frac{c_o}{|o|} \le c_o H\bigl(|o|\bigr).$$

Putting everything together, we get

$$\begin{aligned} c\bigl(\mathcal{P}'\bigr) &= \sum_{i=1}^{n} c_{p(i)} \le \sum_{i=1}^{n} \sum_{o\in O(i)} \frac{c_o}{N(o,i)} \upsilon(o,i) \\ &\overset{\upsilon(o,i)=0 \text{ if } o\notin O(i)}{=} \sum_{o\in\mathcal{P}_{\text{opt}}} \sum_{i=1}^{n} \frac{c_o}{N(o,i)} \upsilon(o,i) \le \sum_{o\in\mathcal{P}_{\text{opt}}} H\bigl(|o|\bigr) c_o \\ &\le H(k)c(\mathcal{P}_{\text{opt}}). \end{aligned}$$

□

Figure 3 shows a worst-case example for the greedy heuristic. We have k paths p_i consisting of one edge with cost $c_{p_i}=\frac{1}{i}$, $i=1,\dots,k$, and one path consisting of k edges with cost $c_{p_{k+1}}=1+\varepsilon$, $\varepsilon>0$. The greedy algorithm takes the paths $p_1,\dots,p_k$ in reverse order at a total cost of $H(k)$. The optimal solution contains only path p_{k+1} with a cost of $1+\varepsilon$.

2.2 The 2-Terminal Case and the Companion Theorem to Menger's Theorem

The 2-terminal case of the Steiner connectivity problem is to find a shortest set of paths connecting two given nodes. This is equivalent to finding a shortest hyperpath in a hypergraph that arises by interpreting paths as hyperedges. This problem can in turn be transformed into an ordinary shortest path problem in a graph replacing each path by a clique with edge weights equal to the path weight. In other words, shortest hyperpaths behave just like shortest paths. Turning to several st-paths, it is also known that Menger's theorem, stating that the maximum number of edge disjoint st-paths equals the minimum number of edges in an st-cut, generalizes to hypergraphs, see Frank [11]. In the graph case, a further result is known, the *companion theorem* to Menger's theorem, that is obtained by interchanging the roles of st-paths and st-cuts, see Robacker [36]. Both results together establish paths and cuts in graphs as a blocking pair, see Fulkerson [12]. We show in this section that the companion to Menger's theorem also holds for hypergraphs. More precisely, we prove the following theorem.

Theorem 2 *The minimum cardinality of an st-hyperpath is equal to the maximum number of hyperedge-disjoint st-hypercuts.*

This result does not follow from the above mentioned shortest path transformation. In fact, we show a stronger result, namely, that the inequality system of the cut formulation to find a shortest st-hyperpath (or 2-terminal Steiner connecting set) is totally dual integral. This extends the blocking property of paths and cuts to hypergraphs and establishes a complete structural similarity between paths and 2-terminal Steiner connecting sets.

We use the following notation. Let $H = (V, \mathcal{E})$ be a connected undirected hypergraph with costs $c_e \in \mathbb{R}$ for all hyperedges $e \in \mathcal{E}$, and s and t be two different nodes of H. Consider the following linear program

$$
\begin{aligned}
\text{(SH)} \qquad & \min \sum_{e\in\mathcal{E}} c_e x_e \\
& \text{s.t.} \sum_{e\in\delta(W)} x_e \geq 1 \quad \forall s \in W \subseteq V \setminus \{t\} \qquad (6) \\
& \qquad x_e \geq 0 \quad \forall e \in \mathcal{E}.
\end{aligned}
$$

Here, we have a variable x_e for each hyperedge $e \in \mathcal{E}$. For $W \subseteq V \setminus \{t\}$, $s \in W$, an *st-hypercut* $\delta(W) = \{e \in \mathcal{E} | e \cap W \neq \emptyset, e \cap (V \setminus W) \neq \emptyset\}$ is the set of all hyperedges having at least one node in each shore. The inequality (6) guarantees for each st-hypercut that the sum of the x-values over all edges in this hypercut is at least 1. We will see that for nonnegative costs the program (SH) always has an optimal solution that is an *st-hyperpath*, i.e., a minimum cost set of hyperedges that connects s and t.

Algorithm 3: Primal-dual shortest hyperpath algorithm

```
Input  : A connected hypergraph $H = (V, \mathcal{E})$, costs $c \in \mathbb{R}^{\mathcal{E}}_{\geq 0}$, $s, t \in V$.
Output: A shortest $st$-hyperpath $x \in \{0, 1\}^{\mathcal{E}}$.
$d(s) := 0$, $d(v) := \infty\ \forall v \in V \backslash \{s\}$, $p(v) := s$, $e(v) := \emptyset\ \forall v \in V$
$i := 1$, $v_0 := s$, $W_0 := \emptyset$, $y_W := 0\ \forall W \subseteq V \setminus \{t\}, s \in W$, $x_e := 0\ \forall e \in \mathcal{E}$
while $t \notin W_{i-1}$ do
    $v := \arg\min\{d(w) | w \in V \setminus W_{i-1}\}$
    for all $f \in \mathcal{E} \setminus \delta(W_{i-1})$ with $v \in f$ do
        for all $w \in f \setminus W_{i-1}$ do
            if $d(w) > d(v) + c_f$ then
                $d(w) := d(v) + c_f$, $p(w) := v$, $e(w) := f$
            end
        end
    end
    $v_i := v$
    $y_{W_{i-1}} := d(v_i) - d(v_{i-1})$
    $W_i := W_{i-1} \cup \{v_i\}$
    $i := i + 1$
end
$k := 1$, $u_k := t$
while $u_k \neq s$ do
    $x_{e(u_k)} := 1$, $u_{k+1} := p(u_k)$, $k := k + 1$
end
```

For nonnegative costs, the primal-dual Algorithm 3 computes an optimal integral solution for program (SH) with $x_e \leq 1, \forall e \in \mathcal{E}$. It generalizes Dijkstra's algorithm to the hypergraph setting and has a better complexity than the "clique transformation method" mentioned at the beginning of the section. Its main purpose, however, is to show that the inequality system of program (SH) is totally dual integral (TDI).

Theorem 3 *The inequality system of program* (*SH*) *is TDI.*

Proof Program (SH) has the following dual

$$\begin{aligned} \max \ & \sum_{W \in \mathcal{W}} y_W \\ \text{s.t.} \ & \sum_{W \in \mathcal{W}: e \in \delta(W)} y_W \leq c_e \quad \forall e \in \mathcal{E} \\ & y_W \geq 0 \quad \forall W \in \mathcal{W}, \end{aligned} \tag{7}$$

where $\mathcal{W} = \{W \subseteq V \backslash \{t\} | s \in W\}$. If $c_e < 0$ for an $e \in \mathcal{E}$ then (SH) has no finite solution since x is not bounded from above; with $x_e \to \infty$ we can improve the objective arbitrarily. This means that we can assume a nonnegative integer cost vector in the following. We prove the claim for this case by showing that the primal-dual shortest hyperpath Algorithm 3 constructs optimal integral solutions x for (SH) and y for (7) with the same objective value.

The algorithm adds nodes v_i to sets $W_{i-1} = \{v_1, \dots, v_{i-1}\}$ in the order of increasing distance $d(v_i)$ from $s = v_0$ $(= v_1)$, i.e., $d(v_{i-1}) \le d(v_i) < \infty$, $i = 1, \dots, h$, with h being the last iteration of the while loop 3. This produces a sequence of nested st-hypercuts $\delta(W_i)$, $i = 1, \dots, h-1$.

We first show that y is a solution of program (7). Lines 2 and 13 imply $y \ge 0$. In fact, the variables y_W can take positive values only for $W \in \{W_1, \dots, W_{h-1}\}$.

It remains to show that

$$\sum_{W \in \mathcal{W}: e \in \delta(W)} y_W \le c_e \quad \forall e \in \mathcal{E}. \tag{8}$$

Let $e \in \mathcal{E}$. If $v_i \notin e$ for all $i = 1, \dots, h-1$, then $e \notin \delta(W_i)$, $i = 1, \dots, h-1$, i.e., $\sum_{W \in \mathcal{W}: e \in \delta(W)} y_W = 0 \le c_e$. Otherwise, let $1 \le i < h$ be the minimal index smaller than h such that $v_i \in e$, i.e., $e \notin \delta(W_j)$ for $1 \le j < i < h$ but $e \in \delta(W_i)$, and let $i \le \ell \le h-1$ be the maximal index such that $e \in \delta(W_j)$ for $i \le j \le \ell$. Then inequality (8) becomes

$$\begin{aligned}\sum_{W \in \mathcal{W}: e \in \delta(W)} y_W &= \sum_{j=i}^{\ell} y_{W_j} = \sum_{j=i}^{\ell} d(v_{j+1}) - d(v_j)\\ &= d(v_{\ell+1}) - d(v_i) \le c_e.\end{aligned}$$

For the last inequality we distinguish the cases $v_{\ell+1} \in e$ and $v_{\ell+1} \notin e$. The first case follows since $v_i \in e$. In the second case, $\ell + 1 = h$, i.e., $v_{\ell+1} = v_h = t$ and there exists a node $w \in e$ with $w \notin W_{h-1}$. Since $d(v_{\ell+1}) = d(t) \le d(w)$ and $w, v_i \in e$, the second case follows analogously.

Now we show that x is a solution of program (SH). Due to the definition of x we have $x \ge 0$. We have to show that

$$\sum_{e \in \delta(W)} x_e \ge 1 \quad \forall s \in W \subseteq V \backslash \{t\}. \tag{9}$$

Consider the nodes $t = u_1, \dots, u_k = s$ computed in the while loop starting in line 18 and an st-hypercut $\delta(W)$. Let i be the largest index with $u_i \notin W$ and $u_{i+1} \in W$. This index exists since $u_1 = t \notin W$ and $u_k = s \in W$. Then we have $x_{e(u_i)} = 1$, $e(u_i) \in \delta(W)$, and inequality (9) is satisfied.

The objective value of program (7) is

$$\sum_{i=1}^{h-1} y_{W_i} = \sum_{i=1}^{h-1} d(v_{i+1}) - d(v_i) = d(v_h) - d(v_1) = d(t) - d(s) = d(t).$$

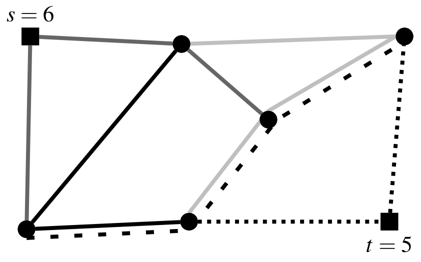

Fig. 4 Example for a Grötschel 65-index of 3: The minimum cardinality of a 65-hyperpath is $|\{p_{\text{gray}}, p_{\text{lightgray}}, p_{\text{dotted}}\}| = 3$. This equals the maximum number of hyperedge-disjoint 65-hypercuts $|\{\{p_{\text{gray}}\}, \{p_{\text{black}}, p_{\text{dashed}}, p_{\text{lightgray}}\}, \{p_{\text{dotted}}\}\}| = 3$

We, finally, get for the objective value of program (SH)

$$\begin{aligned} d(t) = d(u_1) &= d(u_2) + c_{e(u_1)} = d(u_3) + c_{e(u_2)} + c_{e(u_1)} = \dots \\ &= d(u_k) + \sum_{i=1}^{k} c_{e(u_i)} = 0 + \sum_{e \in \mathcal{E}} c_e x_e, \end{aligned}$$

i.e., x and y have the same objective value and are therefore optimal for (SH) and (7). Since c_e is integral, it follows that $d(v_i)$ is integral for $i = 0, \dots, h$. Therefore, y_{W_i}, $i = 1, \dots, h-1$, is also integral (line 13). This shows the claim for $c_e \geq 0$, $e \in \mathcal{E}$. $\square$

Setting $c \equiv 1$ yields Theorem 2, a combinatorial result on hypergraph connectivity which has, to the best of our knowledge, not been considered before. Denote the maximum number of hyperedge-disjoint st-hypercuts the *Grötschel st-index* of a hypergraph. We can then restate Theorem 2 as follows:

Theorem 4 (Grötschel st-index Theorem) *The minimum cardinality of an st-hyperpath is equal to the Grötschel st-index.*

Figure 4 gives an illustration. As this result is derived from a careful analysis of the hypergraph vs. the graph case, we feel that it fits very well to dedicate this Theorem to Martin Grötschel on the occasion of his 65th birthday.

Interchanging the roles of st-hyperpaths and st-hypercuts yields Menger's theorem for hypergraphs, see [11].

Theorem 5 *The minimum cardinality of an st-hypercut is equal to the maximum number of hyperedge-disjoint st-hyperpaths.*

Grötschel's and Menger's Theorems 4 and 5 are therefore companion theorems indeed.

3 The Steiner Tree Packing Problem

When Martin Grötschel came in 1989 to Alexander Martin and suggested the topic of packing Steiner trees in graphs as a Ph.D. thesis, he became immediately very curious about it. Martin Grötschel further supported it by saying that this is a topic for the next decades. He claimed that very few is known when it comes to packing problems in general, one reason among others is that it is open on how to integrate and exploit dual information. And he seems to be right until today. We all know that some progress has been made when it comes to classical packing problems such as the set packing or the bin packing problem. But for the STPP, when the objects to be packed have no fixed shape and are flexible in size and structure in dependence on the other objects to be packed, the problem seems to be harder by orders of magnitudes. None of the successful techniques like preprocessing or heuristics for the single Steiner tree problem work anymore, and new ideas must be developed. With very few exceptions, cf. [14–19], the STPP is still open for wonderful discoveries, supporting once more the great ability of Martin Grötschel to identify future trends and challenging problems.

The Steiner tree packing problem (STPP) looks at the following situation. Instead of having one set of terminals, we have N non-empty disjoint sets $T_1, \ldots, T_N$, called *Nets*, that have to be "packed" into the graph simultaneously, i.e., the resulting edge sets $S_1, \ldots, S_N$ have to be pairwise disjoint. In these applications, G is usually some sort of 3D grid graph. [19, 20, 27] give detailed explanations of the modeling requirements in VLSI-design. From a theoretical point of view, much less is known, see [7, 26].

Three routing models for 2D or 3D grid graphs are of particular interest:

Channel routing: Here, a complete rectangular grid graph is used. The terminals of the nets are exclusively located on two opposing borders. The size of the routing area is not fixed in advance. All nets have only two terminals, i.e., $|T_i| = 2$.

Switchbox routing: We are given a complete rectangular grid graph. The terminals may be located on all four sides of the graph. Thus, the size of the routing area is fixed.

General routing: In this case the grid graph may contain holes or have a non-rectangular shape. The size of the routing area is fixed and the terminals may be located arbitrarily.

The intersection of the nets is an important issue in Steiner tree packing. Again three different models are possible:

Manhattan (Fig. 5(a)) Consider some (planar) grid graph. The nets must be routed in an edge disjoint fashion with the additional restriction that nets that meet at some node are not allowed to bend at this node, i.e., so-called *Knock-knees* are not allowed. This restriction guarantees that the resulting routing can be laid out on two layers at the possible expense of causing long detours.

Knock-knee (Fig. 5(b)) Again, some (planar) grid graph is given and the task is to find an edge disjoint routing of the nets. In this model Knock-knees are possible.

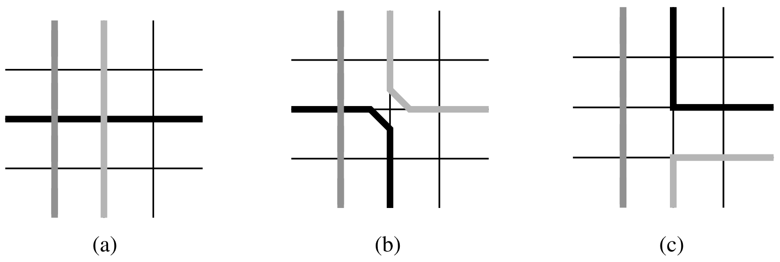

Fig. 5 STPP intersection models. **a** Manhattan model. **b** Knock-knee model. **c** Node disjoint model

Very frequently, the wiring length of a solution is smaller than in the Manhattan model. The main drawback is that the assignment to layers is neglected.

Node disjoint (Fig. 5(c)) The nets have to be routed in a node disjoint fashion. Since no crossing of nets is possible in a planar grid graph, this requires a multi-layer model, i.e., a 3D grid graph.

While channel routing usually involves only a single layer, switchbox and general routing problems are typically multi-layer problems. Using the Manhattan and Knock-knee intersection is a way to reduce the problems to a single layer. Accordingly, the multi-layer models typically use the node disjoint intersection. While the multi-layer model is well suited to reflect reality, the resulting graphs become quite large. We consider two possibilities to model multiple layers; a third possibility is to use a single-layer model with edge capacities greater than one:

k-crossed layers (Fig. 6(a)) A k-dimensional grid graph (i.e., k copies of a grid graph are stacked on top of each other and corresponding nodes are connected by perpendicular lines, so-called *vias*) is given, where k denotes the number of layers. This is called the k-layer model in [27].

k-aligned layers (Fig. 6(b)) This model is similar to the crossed-layer model, but in each layer there are only connections in one direction, either east-to-west or north-to-south. [27] calls this the *directional* multi-layer model. [26] indicate that for $k = 2$ this model resembles the technology used in VLSI-wiring best. It is mentioned in [2] that current technology can use a much higher number of layers (20 and more).

Note that for switchbox routing there is a one-to-one mapping between feasible solutions for the Manhattan one-layer model (MOL) and the node disjoint two-aligned-layer model (TAL), assuming that there are never two terminals on top of each other, i.e., connected by a via.

For the general routing model, this mapping might not be possible. If a terminal is within the grid, there is no easy way to decide the correct layer for the terminal in the two-layer model.

Unfortunately, in the seven "classic" instances given by [6, 9, 29] two terminals are connected to a single corner in several cases. This stems from the use of *connec-*

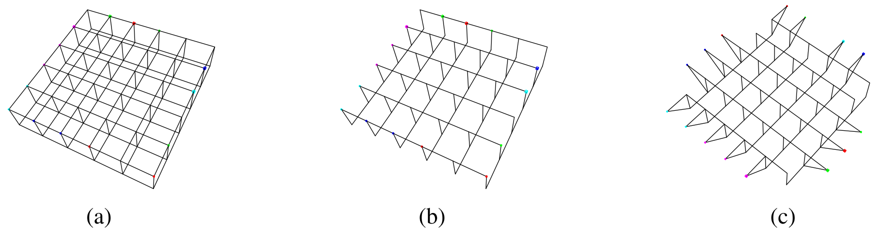

Fig. 6 STPP modeling taxonomy. **a** Multi-crossed layers. **b** Multi-aligned layers. **c** With connectors

tors, i.e., the terminal is outside the grid and connected to it by a dedicated edge. In the multi-layer models there has to be an edge from the terminal to all permissible layers (Fig. 6(c)).

The Knock-knee one-layer model can also be seen as an attempt to approximate the node disjoint two-crossed-layer model. But mapping between these two models is not as easy. [5] have designed an algorithm that guarantees that any solution in the Knock-knee one-layer model can be routed in a node disjoint four-crossed-layer model, but deciding whether three layers are enough has been shown to be $\mathcal{NP}$-complete by [28].

For our computational investigations we will use a multicommodity flow formulation [25] that was proposed by [38] for the STP. Given a weighted bidirectional grid digraph $G = (V, A, c)$ and sets $T_1, \ldots, T_N$, $N > 0$, $|T_n| > 0$ of terminals, we arbitrarily choose a root $r_n \in T_n$ for each $n \in \mathcal{N} := \{1, \ldots, N\}$. Let $R = \{r_n | n \in \mathcal{N}\}$ be the set of all roots and $T = \bigcup_{n \in \mathcal{N}} T_n$ be the union of all terminals. We introduce binary variables x^n_{ij} for all $n \in \mathcal{N}$ and $(i, j) \in A$, where $x^n_{ij} = 1$ if and only if arc $(i, j) \in S_n$. Additionally, we introduce binary variables y^t_{ij}, for all $t \in T \setminus R$. For all $i \in V$, we define $\delta^+_i := \{(i, j) \in A\}$ and $\delta^-_i := \{(j, i) \in A\}$. For all $t \in T_n$, $n \in \mathcal{N}$, we define $\sigma(t) := n$. The following formulation models all routing choices for any number of layers, crossed and aligned, with Knock-knee intersection:

$$\min \sum_{n \in \mathcal{N}} \sum_{(i,j) \in A} c^n_{ij} x^n_{ij} \tag{10}$$

$$\sum_{(i,j) \in \delta^-_j} y^t_{ij} - \sum_{(j,k) \in \delta^+_j} y^t_{jk} = \left\{ \begin{array}{ll} 1 & \text{if } j = t \\ -1 & \text{if } j = r_{\sigma(t)} \\ 0 & \text{otherwise} \end{array} \right\} \quad \text{for all } j \in V, t \in T \setminus R \tag{11}$$

$$0 \le y^t_{ij} \le x^{\sigma(t)}_{ij} \quad \text{for all } (i, j) \in A, t \in T \setminus R \tag{12}$$

$$\sum_{n \in \mathcal{N}} (x^n_{ij} + x^n_{ji}) \le 1 \quad \text{for all } (i, j) \in A \tag{13}$$

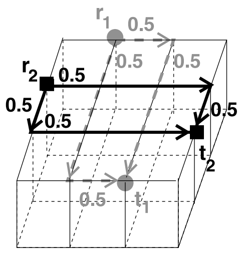

Fig. 7 LP relaxation solution violates (17)

$$x_{ij}^n \in \{0,1\} \quad \text{for all } n \in \mathcal{N}, (i,j) \in A \tag{14}$$

$$y_{ij}^t \in \{0,1\} \quad \text{for all } t \in T \setminus R, (i,j) \in A \tag{15}$$

To use node disjoint intersection we have to add:

$$\sum_{n\in\mathcal{N}} \sum_{(i,j)\in\delta_j^-} x_{ij}^n \le \begin{cases} 0 & \text{if } j \in R \\ 1 & \text{otherwise} \end{cases} \quad \text{for all } j \in V. \tag{16}$$

3.1 Valid Inequalities

For the node disjoint crossed-layer model there is a class of simple but very effective cuts, which are introduced in [21]. Consider a STPP problem of two nets, each has two terminals as in Fig. 7. The flows shown in the picture correspond to an optimal solution of the LP relaxation. However, it can be seen that if none of the two flows (r_1,t_1) and (r_2,t_2) leaves the upper layer, they have to cross each other, i.e., the node disjoint intersection condition is violated. In the following we construct cuts, which cut off the fractional solution in Fig. 7.

A pair $(s_1,t_1), (s_2,t_2) \in T \times (T\backslash R)$ are called crossed if these four terminals lie on the boundary and in the same layer, $\sigma(s_1) = \sigma(t_1)$, $\sigma(s_2) = \sigma(t_2)$, $\sigma(s_1) \neq \sigma(s_2)$, and, moreover, the line segments (s_1,t_1) and (s_2,t_2) cross each other. Let $\mathcal{C}$ be the set of all crossing pairs and for each node v we denote by v_z the layer number of node v. Then the following inequality is valid for (10):

$$\sum_{\substack{ij\in A \\ i_z=(r_1)_z, j_z\neq i_z}} y_{ij}^{t_1} + y_{ij}^{t_2} \ge 1, \quad \forall\big((r_1,t_1),(r_2,t_2)\big) \in \mathcal{C}(R), \tag{17}$$

where $\mathcal{C}(R) := \{((s_1,t_1),(s_2,t_2)) \in \mathcal{C} | s_1, s_2 \in R\}$. Equation (17) means that at least one of the two flows (r_1,t_1) and (r_2,t_2) has to leave the layer containing these terminals.

A triple $(r_1,t_1), (r_2,t_2), (r_3,t_3) \in R \times (T\backslash R)$ is called a crossing triple if each two of them are a crossing pair in $\mathcal{C}(R)$. Let $\mathcal{CT}$ be the set of all crossing triples then the following inequality is valid for (10):

$$\sum_{\substack{ij \in A \\ i_z = (r_1)_z, j_z \neq i_z}} y_{ij}^{t_1} + y_{ij}^{t_2} + y_{ij}^{t_3} \geq 2, \quad \forall \big((r_1, t_1), (r_2, t_2), (r_3, t_3) \big) \in \mathcal{CT}. \tag{18}$$

This is a direct implication from summing up the three corresponding inequalities of type (17) taking into account that the variables have to be integral.

Based on the same principle, we can derive more valid cuts involving both variables x and y. Again, we assume that all terminals lie on the boundary. Let u and v be two terminals in one layer of a net n and the root of this net does not belong to the layer containing u and v. If there exist a root r and a terminal t of another net such that (u, v) and (r, t) cross each other, i.e., $((u, v), (r, t)) \in \mathcal{C}$, then the following inequality is valid for (10)

$$\sum_{\substack{ij \in A \\ j_z = t_z, j_z \neq i_z}} y_{ij}^{t} + x_{ij}^{n} \geq 2. \tag{19}$$

The following valid cut is similar to (19) but two terminal pairs and three terminals of a third net are involved. We consider an arbitrary net n with at least three terminals and four terminals r_1, t_1, r_2 and t_2 of two other nets n_1 and n_2, $\sigma(r_1) = \sigma(t_1) = n_1 \neq n$, $\sigma(r_2) = \sigma(t_2) = n_2 \neq n$, $n_1 \neq n_2$, with $r_1, r_2 \in R$. If there exist three terminals u, v and w of net n lying in the same layer such that

$$\forall \{s, t\} \subset \{u, v, w\}, s \neq t : \big((s, t), (r_1, t_1) \big) \in \mathcal{C} \text{ or } \big((s, t), (r_2, t_2) \big) \in \mathcal{C}, \tag{20}$$

then the following inequality is valid:

$$\sum_{\substack{ij \in A \\ j_z = (t_1)_z, j_z \neq i_z}} y_{ij}^{t_1} + y_{ij}^{t_2} + x_{ij}^{n} \geq 2. \tag{21}$$

Since all terminals lie on the boundary, each line segment between two terminals, which crosses an edge of a "triangle" of three terminals (including the case that they lie in a line), crosses exactly two edges of this triangle. Therefore, condition (20) just ensures that the line segments (r_1, t_1) and (r_2, t_2) cross the triangle (u, v, w) in two different pairs of edges. Inequality (21) means that the total number of vias used by the flows (r_1, t_1) and (r_2, t_2) and the net n to enter the layer containing these terminals is at least two. The proof can be found in [21]. Moreover, one can prove that, if u, v and w satisfy the above condition and u, v and w do not lie in the same layer as the root of net n then the following inequality is valid:

$$\sum_{\substack{ij \in A \\ j_z = (t_1)_z, j_z \neq i_z}} y_{ij}^{t_1} + y_{ij}^{t_2} + x_{ij}^{n} \geq 3. \tag{22}$$

The proof can also be found in [21].

The next type of valid inequality does not require any of the terminals to be the root of the net they belong to. For arbitrary three terminals u_1, v_1 and w_1 of a net n_1, and arbitrary three terminals u_2, v_2 and w_2 of a net $n_2 \neq n_1$, if

$$\forall\{s_1, t_1\} \subset \{u_1, v_1, w_1\}, \exists\{s_2, t_2\} \subset \{u_2, v_2, w_2\} : \big((s_1, t_1), (s_2, t_2)\big) \in \mathcal{C}, \tag{23}$$

and

$$\forall\{s_2, t_2\} \subset \{u_2, v_2, w_2\}, \exists\{s_1, t_1\} \subset \{u_1, v_1, w_1\} : \big((s_1, t_1), (s_2, t_2)\big) \in \mathcal{C}, \tag{24}$$

i.e., the two triangles (u_1, v_1, w_1) and (u_2, v_2, w_2) cross each other, then the following inequality is valid for (10):

$$\sum_{\substack{ij \in A \\ j_z = (u_1)_z, j_z \neq i_z}} x_{ij}^{n_1} + x_{ij}^{n_2} \geq 2 + \delta_1 + \delta_2, \tag{25}$$

where δ_i is 1 if the root of net n_i does not lie in the same layer as u_i, v_i and w_i, and 0 otherwise.

3.2 *Heuristics*

As we will see in the following the above described model provides very strong lower bounds in practice. Nevertheless, current IP solvers often not only fail to solve the IP model to optimality, it is even very time-consuming to find a feasible solution.

However, based on the special structure of the grid graph we can find optimal solutions for all of our considered instances in reasonable time. The following is an extension to [21] and describes the heuristics we developed to improve the solution of the STPP. The heuristics presented in this section are based on solving the IP of relaxed problems. There are two kinds of relaxed problems. The first one is also the STPP problem using the multicommodity flow formulation (10) but on a subgraph of the original grid graph. The second one is the original multicommodity flow IP where some variables are fixed based on a given feasible solution of the original problem. In the following we call these two kinds of heuristics phase 1 and phase 2, respectively. The solving process of the STPP starts with phase 1 and then executes phase 2. The two phases of the heuristics are presented in detail below.

3.2.1 Heuristics Phase 1

Computational results show that feasible solutions of the multi-aligned layers can be found easily if they exist. This motivates us to consider a heuristic process, where instead of starting solving the original multi-crossed layers model, we solve several STP problems corresponding to some sparser underlying grids, e.g., the multi-aligned layers grid. After each step we obtain a feasible solution. Then we add some

Algorithm 4: Heuristics phase 1

```
Step 1: Solve the STPP on the multi-aligned layers grid.
Step 2: Add all missing edges in the set $S(l_2, r_2, b_2, t_2)$ to the grid graph in Step 1. Find a good solution of the STPP on this new grid graph using the solution of Step 1 as starting solution.
Step 3: Add all missing edges in the set $S(l_3, r_3, b_3, t_3)$ to the grid graph in Step 2. Find a good solution of the STPP on this new grid graph using the solution of Step 2 as starting solution.
```

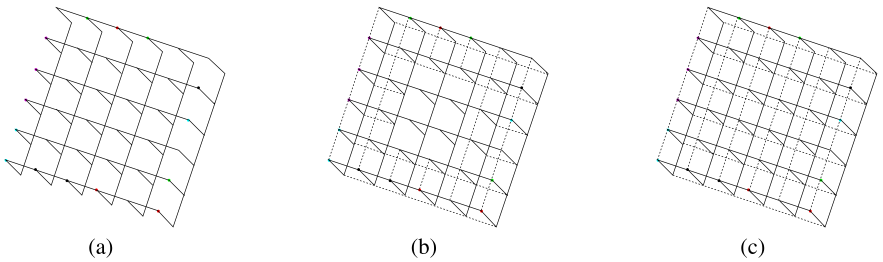

Fig. 8 STPP heuristics phase 1—underlying grid graphs. **a** Step 1. **b** Step 2. **c** Step 3

edges to the grid and resolve the STP problem which corresponds to the new grid, using the solution from the previous step as a start value for the optimization problem. The question is which edges should be added in each step. There is no optimal strategy at the moment. Algorithm 4 describes our implemented algorithm, where $S(l, r, b, t)$ is the set of all edges lying outside $[l, K - r] \times [b, L - t]$, i.e.,

$$S(k, l) := \left\{(i, j) \in E \;\middle|\; (i_x, i_y), (j_x, j_y) \notin [l, K - r] \times [b, L - t]\right\},$$

with (i_x, i_y) is the coordinate of the vertex i in the layer which it belongs to, $[0, K] \times [0, L]$ is the given grid, and E is the set of edges in the multi-crossed layer model.

Figure 8 demonstrates the grid graphs in the three steps, where the added edges are dotted.

It is still unknown what is the best way to choose the parameters (l_2, r_2, b_2, t_2) and (l_3, r_3, b_3, t_3). For the instances in Sect. 3.3 we choose $l_2 = r_2 = b_2 = t_2 = 3$ for pedabox-2 and $l_2 = r_2 = b_2 = t_2 = 5$ for the other instances. In step 3, to obtain the parameters l_3, r_3, b_3, t_3, we increase the corresponding parameters l_2, r_2, b_2, t_2 from step 2 by at most 2. For example, we choose $l_3 = r_3 = 7$ and $b_3 = t_3 = 6$ for the instances difficult-2 and more-diff-2.

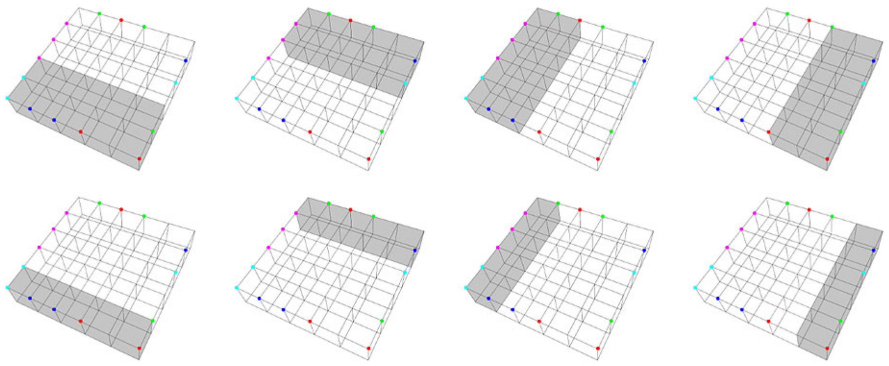

Fig. 9 STPP heuristics phase 2—fixing regions

3.2.2 Heuristics Phase 2

Having a feasible solution from heuristics phase 1 we can start the fixing heuristics. Figure 9 shows an example for the procedure of these heuristics. This figure reads from left to right and from top to bottom. Let us start with the picture on top left. We fix the variables corresponding to the edges in the gray region to the values of the given feasible solution for those variables. Then we solve the IP (10) with those fixings and obtain a possibly better feasible solution. With the newly obtained feasible solution we go to the second step described by the second picture in the top of Fig. 9, where again the fixing area is colored by gray, and so on. The given feasible solution from the beginning is improved step by step, where each step uses the best feasible solution obtained in the previous step for fixing. At the moment there is no common rule for the fixing region and the number of steps. Algorithm 5 presents the pseudo code of phase 2 in our implemented code. We execute four steps with the sequence of the fixing regions as the four pictures either on the left side or on the right side of Fig. 9, respectively, as follows. Let $K \times L$ be the size of the grid. Without loss of generality, we assume that $K \leq L$. Otherwise we have to modify the following definition accordingly by switching the role of K and L. The four fixing regions are defined as

$$F_l := \left\{ (i,j) \in E \;\middle|\; i_y, j_y \leq \left\lfloor \frac{K}{p_l} \right\rfloor \right\}, \quad l = 1, 3$$

$$F_l := \left\{ (i,j) \in E \;\middle|\; i_y, j_y \geq K - \left\lfloor \frac{K}{p_l} \right\rfloor \right\}, \quad l = 2, 4,$$

where $3 \leq p_1, p_2 \leq 4$ and $5 \leq p_3, p_4 \leq 6$ are chosen depending on instances. For our considered instances, this procedure gives us already optimal solutions. However, for larger instances, we may need to execute more steps and/or use more types of fixing regions, e.g., all eight steps in Fig. 9.

Algorithm 5: Heuristics phase 2

```
for $i = 1$ to 4 do
    Choose $F_i$ as the fixing region and the solution obtained from phase 1 in
    the case $i = 1$ and from the previous loop $i - 1$ otherwise as the solution
    used for fixing. Find a good solution of the corresponding relaxed problem.
end
```

Table 1 Results for the Knock-knee-one-layer model

Name	Size	N	$\|T\|$	B&B Nodes	Time [s]	LP relaxation	Arcs
aug-dense-1	16×18	19	59	63	1,649	466.5	469
dense-1	15×17	19	59	150	1,199	438.0	441
difficult-1	23×15	24	66	1	17	464.0	464
mod-dense-1	16×17	19	59	1	29	452.0	452
more-diff-1	22×15	24	65	1	12	452.0	452
pedabox-1	15×16	22	56	1	7	331.0	331
termintens-1	23×16	24	77	1	96	535.0	536

3.3 Computational Results

In this section, we present computational results obtained by generating the integer program resulting from the directed multicommodity flow formulation with ZIMPL [24] and then solving it with CPLEX 12.3 for the STPP instances taken from [30]. All computations are done on a 48 GB RAM dual quad-core Intel Xeon X5672 at 3.20 GHz with TurboBoost active and Hyperthreading deactivated. Since the crossed-layer model proved to be much harder to solve, we used all eight cores, while just one core was utilized for the other models. Still, for the crossed-layer models we will use minutes as the unit for reporting time, in contrast to seconds for the other models. As we had expected from earlier experiments, the MCF-Cuts [1, 34] introduced by CPLEX 12 had no impact on solving the instances. The reason is that the models used in this paper are not capacitated. If not noted otherwise, CPLEX was used in default mode with integer optimality gap tolerance set to 0.0.

3.3.1 Results for the Knock-Knee One-Layer Model

Table 1 shows the results for the Knock-knee one-layer model. *B&B Nodes* denotes the number of Branch-and-Bound nodes including the root node evaluated by CPLEX. The column labeled *Time* shows the consumed CPU time in seconds. LP *relaxation* lists the objective function value of the initial LP relaxation of the root node before any cuts applied by CPLEX. Finally, *arcs* is the total number of arcs used in the optimal solution which for one-layer models is equivalent to the optimal objective value.

As we can see from the table, the LP relaxation of the flow model is rather strong. This is in line with other reported results including [22, 30]. Since for *difficult-1*, *mod-dense-1*, *more-diff-1*, and *pedabox-1* the relaxation already provides the optimal value, it is possible to solve these instances without any branching. For *termintens-1* the relaxation is one below the optimum, but CPLEX is able to push the lower bound up by generating Gomory rounding and 0–1/2-Chvátal-Gomory cuts. The number of B&B nodes and therefore the computing time depends very much on the branching decisions taken. During our experiments the solutions were always found in the tree and not by heuristics. By using improved settings, like switching off the heuristics and just trying to move the best bound, the number of nodes needed for *aug-dense-1* and *dense-1* can be at least halved.

3.3.2 Results for the Node Disjoint Multi-aligned-layer Model

Table 2 shows results for the node disjoint multi-aligned-layer model. Since this is a multi-layer model we have to assign costs to the vias. These are given in the column labeled *Via-cost*. The column labeled *LP relaxation* gives the objective value of the initial LP relaxation. The next three columns list the numbers of vias, "regular" arcs, and vias+arcs in the optimal solution.

In case of unit via costs, the objective value of the LP relaxation is equal to the objective value of the optimal integer solution for all instances except for *more-diff-2*. The value of the LP relaxation for *more-diff-2* is 518.6 (optimal 522). This is weaker than the value reported in [19], which indicates that some of the strengthening cuts used by [19] to tighten the undirected partitioning formulation can also be used to tighten the directed flow formulation. On the other hand, for *pedabox-2* the relaxation is stronger than reported. The instances where the LP relaxation does not reach the optimum are different ones from the Knock-knee-one-layer model.

Via Minimization Traditionally via minimization is viewed as a separate problem after the routing has taken place [13]. Since we work with multi-layer models, via minimization is part of the routing. As can be seen in Table 2 we tried the "classical" instances with three different cost settings for the vias. First unit costs were used to minimize the total number of arcs, including vias. Next, the number of vias was minimized by setting the cost to 1,000, which dominates the total cost of all "regular" arcs, ensuring that a global minimum is reached. Finally, the cost of each via was set to 0.001, effectively minimizing the number of "regular" arcs. This results in solutions that have the same number of arcs as reported in [19] for the Manhattan one-layer model.

Interestingly, the number of vias is constant for *aug-dense-2*, *pedabox-2*, *modifieddense-3*, and *dense-3*. For the other instances, a minimization of the number of vias always results in detours, i.e., a higher total number of arcs used.

Table 2 Results for the node disjoint multi-aligned-layer model

Name	Size	N	$\|T\|$	B&B Nodes	Time [s]	Via-cost	LP relaxation	Vias	Arcs	Vias+Arcs
aug-dense-2	16×18	19	59	1	32	1	504.0	35	469	504
aug-dense-2	16×18	19	59	1	20	1000	35,469.0	35	469	504
aug-dense-2	16×18	19	59	1	67	0.001	469.035	35	469	504
difficult-2	23×15	24	66	1	23	1	526.0	56	470	526
difficult-2	23×15	24	66	7	181	1000	50,310.2727	51	484	535
difficult-2	23×15	24	66	1	50	0.001	468.8363	63	469	532
more-diff-2	22×15	24	65	5	113	1	518.60	61	461	522
more-diff-2	22×15	24	65	37	705	1000	50,276.8465	53	481	534
more-diff-2	22×15	24	65	1	38	0.001	460.9917	61	461	522
pedabox-2	15×16	22	56	1	10	1	390.0	47	343	390
pedabox-2	15×16	22	56	11	34	1000	45,885.1333	47	343	390
pedabox-2	15×16	22	56	14	78	0.001	341.4601	47	343	390
termintens-2	23×16	24	77	1	28	1	596.0	59	537	596
termintens-2	23×16	24	77	1	31	1000	55,562.0	55	562	617
termintens-2	23×16	24	77	1	28	0.001	537.059	59	537	596
dense-3	15×17	19	59	1	30	1	471.0	35	436	471
dense-3	15×17	19	59	1	21	1000	35,436.0	35	436	471
dense-3	15×17	19	59	1	44	0.001	436.035	35	436	471
mod-dense-3	16×17	19	59	1	24	1	485.0	35	450	485
mod-dense-3	16×17	19	59	1	26	1000	35,450.0	35	450	485
mod-dense-3	16×17	19	59	1	33	0.001	450.035	35	450	485

Table 3 STPP new instances

Name	Size	N	$\|T\|$	Variables	Constrains	Non-zeros
Node disjoint two-aligned-layer model						
sb80-80	81 × 81	60	158	6,168,636	5,150,535	19,965,324
sb99-99	100 × 100	70	183	10,906,800	9,052,440	35,250,720

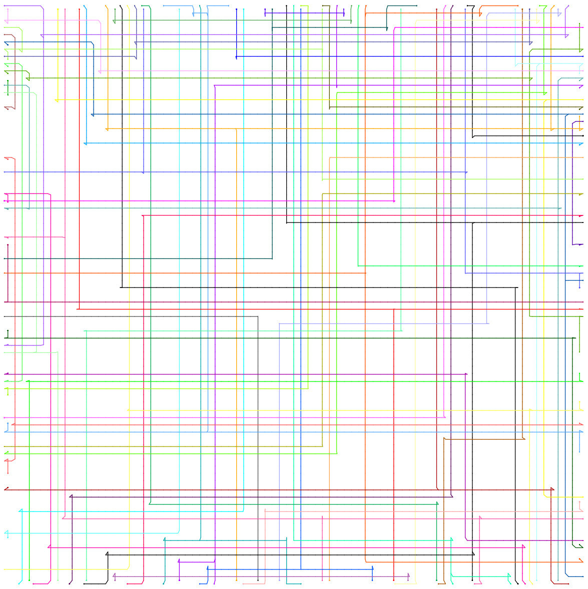
Fig. 10 sb80-80

New Instances All the instances presented so far are relatively old and can be solved in less than 12 minutes with CPLEX. To get an outlook on how far our approach will take us, we tried some new instances, see Table 3. *sb80-80* and *sb99-99* are randomly generated switchbox instances. *sb80-80* is about 35 times the size of the largest "classical" instance, and the resulting IP has more than six million variables and almost twenty million non-zero entries in the constraint matrix. *sb99-99* is again about 2 times larger than *sb80-80*. As discussed in [21] and [25], for several instances it was faster to solve the LP relaxations from scratch with the barrier algorithm than to reoptimize with the dual simplex algorithm. This setting is used for the new instances with a newer version of CPLEX, namely, CPLEX 12.4.

For sb80-80 the value of the LP relaxation turned out to be equal to the value of the integer optimal solution, namely 6533, and only one branch and bound node is needed. CPLEX takes 94,926 seconds to solve the root relaxation and finishes after 94,232 seconds, i.e., 26.18 hours, with the optimal solution, which has 226 vias and 6307 normal arcs, see Fig. 10.

The time for solving the root relaxation of sb99-99 is 356,664 seconds, i.e., 4.13 days. After more than 2 weeks CPLEX cannot either find a feasible solution or prove that the problem is infeasible. Gurobi 5.0 also experiences the same difficulty with this instance.

These computations show the bottleneck of our approach. The resulting IPs are large and solving their LP relaxations is already very hard. Any improvement in solving LP will turn out directly to an improvement in our approach.

3.3.3 Results for the Node Disjoint Multi-crossed-layer Model

Finally, we will have a look at the crossed-layer models. For the instances listed in Table 4, except for *dense-3* and *mod-dense-3*, the heuristics presented in Sect. 3.2 was used to provide CPLEX with a starting solution. For *dense-3* and *mod-dense-3* CPLEX quickly found solutions. Therefore, employing the heuristics provided no advantage. For all instances the provided solution turned out to be already optimal. The time needed to compute these initial solution is given under *Heur*.

To solve the instances we used 8 threads in opportunistic mode, the total times needed including the heuristics is reported in column *Total* as minutes of wall clock time. The *optimization emphasis* of CPLEX was set to "optimality", cut generation was set "aggressive" for Gomory-, 0–1/2-, and cover-cuts, all other cuts were set to "moderate". Furthermore, we explicitly added cuts (17)–(25) presented in Sect. 3.1 to the User-Cut pool of CPLEX.

While it is possible to find reasonable solutions with our heuristics, proving optimality is still a hard task. The table is ordered by an increasing difference between the LP relaxation and the optimum value. As can be seen this is reflected quite well in the number of B&B nodes needed to prove optimality. While the cuts we presented in this paper proved quite helpful, the main reason for the long running time is that solving the node LPs is very time consuming. For example, the number of B&B nodes that CPLEX needs to solve the instance *termintens-2* is merely 3,453. However, it takes 51.7 hours to solve the problem and 43.3 hours for the first 2,000 nodes.

Figure 11 shows the primal and dual bounds during the solving process of the instance *pedabox-2* with CPLEX using default setting and CPLEX using our heuristics, valid cuts, and variables elimination. Clearly, our approach improves both primal and dual bounds. For *pedabox-2* with unit via-cost, our heuristics found an optimal solution after 17.8 minutes, while CPLEX alone could not find the optimal value after more than 80 hours. For this problem our approach (using crossing cuts and special heuristics) gives a dual bound of value 358.2311 directly after the root node, while default CPLEX reaches this value only after 46.1 hours and 158,507 nodes. We stop solving with default CPLEX after 83.33 hours and 300,516 nodes, and obtain a dual bound of 358.7628. This value is already reached by our approach after 49.6 minutes and 471 nodes.

Table 4 Results for the node disjoint multi-crossed-layer model

Name	Size	N	$\|T\|$	Heur [m]	Total [m]	B&B Nodes	LP relaxation	Vias	Arcs	Vias+Arcs
dense-3	15×17	19	59	–	55	77	461.8062	30	434	464
mod-dense-3	16×17	19	59	–	88	121	476.9078	28	451	479
aug-dense-2	16×18	19	59	108	269	227	492.6260	29	469	498
pedabox-2	15×16	22	56	18	112	3,027	353.4275	26	336	362
difficult-2	23×15	24	66	28	822	2,214	492.5417	39	464	503
termintens-2	23×16	24	77	97	3,103	3,453	573.1981	46	538	584
more-diff-2	22×15	24	65	60	30,727	24,713	481.1991	38	455	493

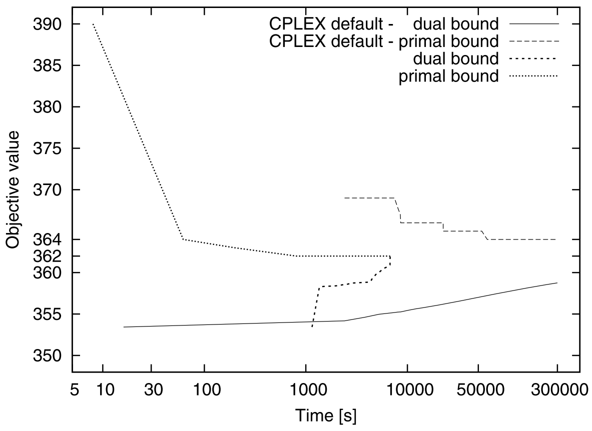

Fig. 11 pedabox-2 with unit via cost—primal and dual bounds

4 Conclusion and Outlook

The Steiner connectivity and the Steiner tree packing problem are just two examples of challenging and important questions to find the best possible connection of a set of terminals in a graph. For such problems chances are good that we can derive theoretical insights and come up with quite powerful solution algorithms, guided by our knowledge of the basic case. We are admittedly still working on individual problem variants, far from anything like a universal solution engine, and, in fact, going one step further into the direction of industrial models, e.g., in line planning or chip design, immediately makes things much more difficult, and even more, since problem sizes of real-world problems are growing fast. But such is life, would Martin Grötschel say. And he could still come, 20 years after Alexander Martin finished his thesis, into the office of one of his Ph.D. students and talk enthusiastically about an interesting and open Steiner problem.

Acknowledgements We thank an anonymous referee and the editors for helpful comments and suggestions that improved the presentation of this paper. The work of Marika Karbstein was supported by the DFG Research Center MATHEON "Mathematics for key technologies".

References

1. Achterberg, T., Raack, C.: The MCF-separator—detecting and exploiting multi-commodity flows in MIPs. Math. Program. Comput. **2**, 125–165 (2010)
2. Boit, C.: Personal communication (2004)
3. Borndörfer, R., Karbstein, M.: A direct connection approach to integrated line planning and passenger routing. In: Delling, D., Liberti, L. (eds.) 12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems. OpenAccess Series in Informatics (OASIcs), vol. 25, pp. 47–57. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Wadern (2012)
4. Borndörfer, R., Karbstein, M., Pfetsch, M.E.: The Steiner connectivity problem. Math. Program., Ser. A (2012). doi:10.1007/s10107-012-0564-5

5. Brady, M.L., Brown, D.J.: VLSI routing: four layers suffice. In: Preparata, F.P. (ed.) Advances in Computing Research: VLSI Theory, vol. 2, pp. 245–258. Jai Press, London (1984)
6. Burstein, M., Pelavin, R.: Hierarchical wire routing. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **2**, 223–234 (1983)
7. Chopra, S.: Comparison of formulations and a heuristic for packing Steiner trees in a graph. Ann. Oper. Res. **50**, 143–171 (1994)
8. Chvátal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. **4**(3), 233–235 (1979)
9. Coohoon, J.P., Heck, P.L.: BEAVER: a computational-geometry-based tool for switchbox routing. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **7**, 684–697 (1988)
10. Feige, U.: A threshold of $\ln n$ for approximating set-cover. In: Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 314–318 (1996)
11. Frank, A.: Connections in Combinatorial Optimization. Oxford University Press, Oxford (2011)
12. Fulkerson, D.R.: Blocking and anti-blocking pairs of polyhedra. Math. Program. **1**, 168–194 (1971)
13. Grötschel, M., Jünger, M., Reinelt, G.: Via minimization with pin preassignments and layer preference. Z. Angew. Math. Mech. **69**(11), 393–399 (1989)
14. Grötschel, M., Martin, A., Weismantel, R.: Optimum path packing on wheels: the consecutive case. Comput. Math. Appl. **31**, 23–35 (1996)
15. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: a cutting plane algorithm and computational results. Math. Program. **72**, 125–145 (1996)
16. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: further facets. Eur. J. Comb. **17**, 39–52 (1996)
17. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: polyhedral investigations. Math. Program. **72**, 101–123 (1996)
18. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: separation algorithms. SIAM J. Discrete Math. **9**, 233–257 (1996)
19. Grötschel, M., Martin, A., Weismantel, R.: The Steiner tree packing problem in VLSI design. Math. Program. **78**(2), 265–281 (1997)
20. Held, S., Korte, B., Rautenbach, D., Vygen, J.: Combinatorial optimization in VLSI design. In: Chvátal, V. (ed.) Combinatorial Optimization—Methods and Applications. NATO Science for Peace and Security Series—D: Information and Communication Security, vol. 31, pp. 33–96 (2011)
21. Hoàng, N.D., Koch, T.: Steiner tree packing revisited. Math. Methods Oper. Res. **76**(1), 95–123 (2012)
22. Jørgensen, D.G., Meyling, M.: Application of column generation techniques in VLSI design. Master's thesis, Department of Computer Science, University of Copenhagen (2000)
23. Karbstein, M.: Line planning and connectivity. Ph.D. thesis, TU Berlin (2013)
24. Koch, T.:. ZIMPL. zimpl.zib.de
25. Koch, T.: Rapid mathematical programming. Ph.D. thesis, Technische Universität Berlin (2004)
26. Korte, B., Prömel, H.J., Steger, A.: Steiner trees in VLSI-layout. In: Korte, B., Lovász, L., Prömel, H.J., Schrijver, A. (eds.) Paths, Flows, and VLSI-Layout. Springer, Berlin (1990)
27. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. Wiley, New York (1990)
28. Lipski, W.: On the structure of three-layer wireable layouts. In: Preparata, F.P. (ed.) Advances in Computing Research: VLSI Theory, vol. 2, pp. 231–244. Jai Press, London (1984)
29. Luk, W.K.: A greedy switch-box router. Integration **3**, 129–149 (1985)
30. Martin, A.: Packen von Steinerbäumen: Polyedrische Studien und Anwendungen. Ph.D. thesis, Technische Universität Berlin (1992)
31. Oxley, J.G.: Matroid Theory. Oxford University Press, Oxford (1992)
32. Polzin, T.: Algorithms for the Steiner problem in networks. Ph.D. thesis, Universität des Saarlandes (2003)

33. Prömel, H., Steger, A.: The Steiner Tree Problem. Vieweg, Wiesbaden (2002)
34. Raack, C., Koster, A.M.C.A., Orlowski, S., Wessäly, R.: On cut-based inequalities for capacitated network design polyhedra. Networks **57**(2), 141–156 (2011)
35. Raghavan, S., Magnanti, T.: Network connectivity. In: Dell'Amico, M., Maffioli, F., Martello, S. (eds.) Annotated Bibliographies in Combinatorial Optimization, pp. 335–354. Wiley, Chichester (1997)
36. Robacker, J.T.: Min-Max theorems on shortest chains and disjunct cuts of a network. Research Memorandum RM-1660, The RAND Corporation, Santa Monica, CA (1956)
37. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica **2**(4), 385–393 (1982)
38. Wong, R.T.: A dual ascent approach for Steiner tree problems on a directed graph. Math. Program. **28**, 271–287 (1984)