Steiner Network

Teodora Dobos
Technical University of Munich
Department of Informatics

1 Introduction

Imagine a telecommunications network that contains nodes connected by links which are used to exchange information between the nodes. A message from a source to a destination node can be sent directly, if a link between these two exists, or passed along a path involving other nodes, which ultimately leads to the destination. But what if some links in the network fail? Will the message reach its destination? How can one ensure that a message will get to its destination even if a specific number of links fail in the initial network? This is a scenario that illustrates the Steiner network or the survivable network design problem (SNP).

SNP was first introduced by Krarup in 1987 [1] [2] and is a combinatorial optimization problem motivated by the telecommunications industry, where one wishes to create low-cost networks that can survive failures of the edges. Given a network and connectivity requirements for all pairs of vertices, the question is how to obtain a multigraph in which any two vertices are connected even after a specific number of edges fail.

This paper is structured as follows. Section 2 presents the configuration of the SNP and introduces its IP and the corresponding LP relaxation. Section 3 discusses what makes the problem hard to solve and describes an iterative rounding algorithm that achieves an approximation guarantee of 2. Section 4 summarizes the main ideas of the paper and briefly introduces another network-design problem that is related to SNP.

2 Problem formulation

In the SNP, an undirected graph $G = (V, E)$ is given, where V and E denote the sets of vertices and edges, respectively. For this graph, the following functions are defined: a cost function $c : E \to \mathbb{Q}^+$, a connectivity requirement function $r : V \times V \to \mathbb{Z}^+$ that maps pairs of distinct vertices to positive integers, and a function $u : E \to \mathbb{Z}^+ \cup \{\infty\}$ which gives an upper bound on the number of copies of an edge e that can be used in the final solution.

If $u(e) = \infty$, then there is no upper bound for e, which means that this edge can be used an arbitrary number of times.

The Steiner network problem is to find a minimum cost multigraph (V, H) which has $r(u, v)$ edge-disjoint paths for each pair of distinct vertices $u, v \in V$. The cost of the multigraph is defined with respect to the function c; each copy of edge e that is used in the resulting graph costs $c(e)$. Suppose there are $r(u, v) - 1$ edge failures in the network. The vertices u and v will still be connected in the modified graph, since $r(u, v) - 1$ edge failures can affect at most $r(u, v) - 1$ edge-disjoint paths that connect u and v. If certain pairs of nodes should be highly connected, i.e. survive a large number of edge failures, then one assigns higher connectivity requirements to these pairs.

Clearly, the generalized Steiner tree problem [3] is a special case of the Steiner network problem in which the connectivity requirement of any pair of vertices is either 0 or 1. Before formulating the integer program for the SNP, let us define the *cut requirement function.*

Definition 1 (Cut requirement function)**.**

$$f : 2^V \to \mathbb{Z}^+,\ f(S) = \max\{r(u, v) | u \in S, v \in \bar{S}\}.$$

The IP for the SNP is:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e && && (1) \\
\text{subject to} \quad & \sum_{e : e \in \delta(S)} x_e \geq f(S), && S \subseteq V \\
& x_e \in \mathbb{Z}^+, && e \in E, u_e = \infty \\
& x_e \in \{0, 1, ..., u_e\}, && e \in E, u_e \neq \infty
\end{aligned}$$

The objective of this optimization problem is to minimize the cost of the resulting multigraph, while satisfying two types of constraints. The former type relates to the required number of edge-disjoint paths between any pair of vertices, while the latter ensures that no edge e is used more than u_e times in the resulting graph. A feasible solution to IP(1) is a vector x^* containing $m = |E|$ entries, where the ith value specifies the number of copies of the edge e_i that are used in the computed graph. If we allow all x_i to take fractional values, we obtain the following LP-relaxation:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e x_e & & \\ \text{subject to} \quad & \sum_{e: e \in \delta(S)} x_e \geq f(S), & & S \subseteq V \\ & x_e \geq 0 & & e \in E, u_e = \infty \\ & u_e \geq x_e \geq 0 & & e \in E, u_e \neq \infty \end{aligned} \tag{2}$$

Since we are dealing with a minimization problem, an optimal solution to LP (2) will give a lower bound for the solution of IP(1).

Let us analyse the correctness of LP(2). By the MaxFlow/MinCut theorem, we know that there are at least $r(u,v)$ edge-disjoint paths between two vertices u and v in the resulting graph (V,H) if and only if every cut $(S,\bar{S})$ that separates the two nodes contains at least $r(u,v)$ edges of H, i.e. $|\delta_H(S) \cap H| \geq r(u,v)$, where $\delta_H(S)$ denotes the edges in H that cross the cut $(S,\bar{S})$. Thus, if we want this statement to hold for any pair of nodes, we infer that the set H of edges is feasible if and only if:

$$|\delta_H(S) \cap H| \geq \max\{r(u,v) | u \in S, v \in \bar{S}\}, \forall S \subseteq V.$$

3 Solving the problem

Recall that a *basic feasible solution (BFS)* of an LP is a feasible solution which cannot be expressed as a convex combination of two feasible solutions. Thus, when solving LP(2), one searches for a BFS.

Clearly, the 'hardness‘ of LP(2) lies in the large number of possible cuts in the given graph. In a graph with n vertices, there are $2^{n-1}-1$ possible cuts $(S,\bar{S})$: we fix a node to be in the set S, and for each of the rest $n-1$ vertices we either assign it to S or to $\bar{S}$ (thus, 2^{n-1} possibilities). Since we do not want S or $\bar{S}$ to be empty, we have to subtract 1 from 2^{n-1}. Therefore, the total number of possible cuts is exponential in the number of vertices and, consequently, the LP has exponentially many constraints. Given this problem, it is not feasible to use the Simplex [4] algorithm (which, in practice, is an efficient method for solving linear programs), because the polytope P defined by the LP has unmanageable many extreme points which, in the worst case scenario, will be all considered in the algorithm's execution.

However, the LP can be solved in polynomial time with the *ellipsoid method* [5], which does not need an explicit description of the facets of the polytope, unlike the Simplex algorithm. Instead, the ellipsoid method solves a *separation problem*: given a point $x \in \mathbb{R}^m$, the method decides whether $x \in P$ or, if this is not the case, it finds a hyperplane (e.g. a violated constraint in the LP) that separates x from the feasible region. To be able to solve the separation problem, the method needs a *separation oracle* that it can solve in polynomial time.

For the Steiner network problem, we consider the following separation oracle which is based on a max flow routine [6]. Given a solution $x \in \mathbb{R}^m$, we construct a graph on the vertex set V having capacity x_e for each edge $e \in E$. Then, for each pair of distinct vertices u and v, we compute the max flow from u to v (since the graph is undirected, we can choose either vertex to be the source and 'direct' the graph). If the maximum flow value sent from u to v is at least $r(u, v)$, then all constraints are satisfied. Otherwise, if for some pair of vertices u, v the maximum flow value is less than $r(u, v)$, then there exists a cut $(S, \bar{S})$ with $u \in S$ and $v \in \bar{S}$, such that $\sum_{e \in \delta(S)} x_e < r(u, v)$, which gives a violated constraint. We can compute a max flow instance in $O(V^3)$ and there are $O(V^2)$ pairs of vertices. Hence, the separation oracle can be solved in $O(V^5)$, which is polynomial in the number of vertices. In the following subsection we present an iterated rounding algorithm that solves the SNP.

3.1 An iterated rounding algorithm

Before introducing a factor 2 algorithm that solves the SNP, let us first define the *residual cut requirement function*, which will be important in the update step of the method. Let H be the set of edges that are picked by the algorithm at some point. Recall that $f(S)$ denotes the cut requirement function and $\delta_H(S)$ represents the edges in H that cross the cut $(S, \bar{S})$.

Definition 2 (Residual cut requirement function)**.**

$$f' : 2^V \to \mathbb{Z}^+,\ f'(S) = f(S) - |\delta_H(S)|.$$

Algorithm 1 is an iterated rounding procedure for solving a SNP instance. The algorithm outputs the set of edges H of the resulting multigraph (V, H). It starts by initializing H to the empty set and the residual cut requirement f' to the cut requirement f. Then, as long as H is not a feasible solution, i.e. the graph (V, H) does not fulfill all constraints related to the required edge-disjoint paths and to the upper bounds on the edges, three main steps are performed. At first (2.1), the current LP is solved on the edges that are not yet included in H, obtaining a BFS, x. In the subsequent step (2.2), the elements of the vector $x \in \mathbb{R}^m$ are considered: if an edge e has a value $x_e \geq 1/2$, then we round up x_e and add $\lceil x_e \rceil$ copies of e in H. Since the upper bounds on all edges e must be fulfilled in the final solution, u_e is decremented by $\lceil x_e \rceil$. In the third step (2.3), the residual cut requirement is updated.

Algorithm 1: Steiner network

```
1. $H \leftarrow \emptyset, f' \leftarrow f$
2. While $H$ is not a feasible solution do:
     2.1 Solve LP(2) on edge set $E - H$ with cut requirements $f'$
         to obtain BFS $x$.
     2.2 For each edge $e$ such that $x_e \geq 1/2$, include $\lceil x_e \rceil$ copies of $e$
         in $H$, and decrement $u_e$ by this amount.
     2.3 Update $f'$: for $S \subseteq V, f'(S) \leftarrow f(S) - |\delta_H(S)|$.
3. Return $H$.
```

Because the algorithm iteratively rounds up the LP solution to obtain the final feasible solution, the technique on which it relies is called *iterated rounding.* Thus, in each iteration, a set of variables x_e are made integral and the rest of the edges are 'ignored‘, leading to a residual problem that considers the edges that were initially dismissed. Then, the LP relaxation of the residual problem is solved.

Generally, if the drop in the LP-value is at least $\frac{1}{\rho}$ times the cost of the rounded variables, the algorithm will give, in the end, a ρ-approximation for the problem it tries to solve. In our setting, the variables are always rounded by at most a factor of 2, which means that we are dealing with a 2-approximation algorithm. In the next subsection we show that Algorithm 1 achieves an approximation guarantee of 2.

3.2 Approximation guarantee of 2

Showing that Algorithm 1 gives a factor 2 approximation for the SNP requires characterizing the original and the residual cut requirement functions. Furthermore, Theorem 4, which will be discussed in this section, reveals a key observation regarding the edge values of a BFS and will be used in our approximation-guarantee proof. In the following, we introduce weakly supermodular and submodular functions and discuss their connection to the original and residual cut requirement functions.

Definition 3. *A function $f : 2^V \rightarrow \mathbb{Z}^+$ is weakly supermodular if $f(\emptyset) = f(V) = 0$, and for any two sets $A, B \subseteq V$ at least one of the following conditions holds:*

$$f(A) + f(B) \leq f(A \cap B) + f(A \cup B) \tag{3}$$

$$f(A) + f(B) \leq f(A - B) + f(B - A). \tag{4}$$

Lemma 1. *The function $f(S) = max\{r(u,v) | u \in S, v \in \bar{S}\}$ is weakly supermodular.*

The proof of this lemma can be found in [7].

Definition 4. *A function $f : 2^V \to \mathbb{Z}^+$ is submodular if $f(V) = 0$, and $\forall A, B \subseteq V$ the following inequalities hold:*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B) \tag{5}$$

$$f(A) + f(B) \geq f(A - B) + f(B - A). \tag{6}$$

We say that two sets of V, A and B, *cross* if the sets $A-B$, $B-A$, and $A \cup B$ are not empty. The function $|\delta_G(.)|$ is characterized by the following lemma.

Lemma 2. *For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.*

Proof. We consider two cases:

Case 1: A and B do not cross. This implies that $A \cap B = \emptyset$ or $A \subseteq B$ or $B \subseteq A$. Assume $A \cap B = \emptyset$ (the other two cases can be proven similarly). Then, it holds $|\delta_G(A \cap B)| = 0$, $|\delta_G(A - B)| = |\delta_G(A)|$ and $|\delta_G(B - A)| = |\delta_G(B)|$. Consequently, $|\delta_G(A)| + |\delta_G(B)| = |\delta_G(A \cup B)| = |\delta_G(A)| + |\delta_G(B)| \geq |\delta_G(A \cap B)| + |\delta_G(A \cup B)|$ and $|\delta_G(A)| + |\delta_G(B)| \geq |\delta_G(A - B)| + |\delta_G(B - A)|$, which leads to the proof of the statement for this case.

Case 2: A and B cross. Analyzing figure 1, we consider three types of edges depending on the places where their endpoints are. Edges such as e_1, having one endpoint in $A \cap B$ and the other in $\overline{A \cup B}$, contribute to $\delta(A)$, $\delta(B)$, $\delta(A \cap B)$, $\delta(A \cup B)$, but not to $\delta(A - B)$, $\delta(B - A)$. Hence, inequality (6) strictly holds and inequality (5) holds with equality. Edges such as e_2, having the endpoints in $A - B$ and $B - A$, contribute to $\delta(A)$, $\delta(B)$, $\delta(A - B)$, $\delta(B - A)$, but not to $\delta(A \cup B)$ and $\delta(A \cap B)$, and, thus, (5) strictly holds and (6) holds with equality. Edges which are not of type e_1 or e_2 contribute equally to both sides of inequalities (5) and (6). $\square$

By lemma 1, the original cut requirement function f is weakly supermodular. Recall that, in each step of the Algorithm 1, the function f' is implicitly updated. Thus, an immediate question is whether all residual cut requirement functions f' are also weakly supermodular. The answer to this question is given by the following lemma.

Lemma 3. *Let H be a subgraph of G. If $f : 2^{V(G)} \to \mathbb{Z}^+$ is weakly supermodular, then so is the residual cut requirement function f'.*

Proof. We prove the first inequality. Suppose $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$. By lemma 2, $|\delta_H(.)|$ is submodular, therefore: $|\delta_H(A)| + |\delta_H(B)| \geq |\delta_H(A \cap B)| + |\delta_H(A \cap B)|$. By definition, $f'(A) = f(A) - |\delta_H(A)|$ and $f'(B) = f(B) - |\delta_H(B)|$. Adding these two

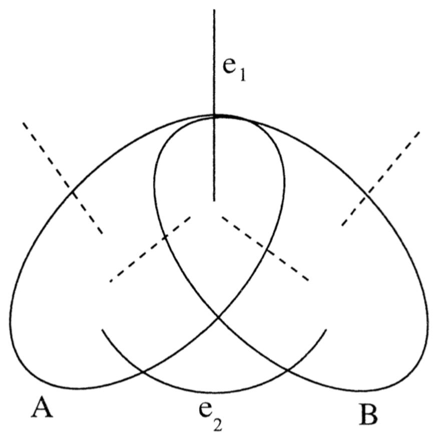

Figure 1: Sets A and B (Source: [8])

terms we get: $f'(A) + f'(B) = (f(A) - |\delta_H(A)|) + (f(B) - |\delta_H(B)|) \leq (f(A \cup B) - |\delta_H(A \cup B)|) + (f(A \cap B) - |\delta_H(A \cap B)|) = f'(A \cup B) + f'(A \cap B)$. The second inequality follows analogously. $\square$

We now state a central theorem that will help us prove that Algorithm 1 achieves an approximation guarantee of 2.

Theorem 4. *For any BFS x to our LP such that f is a weakly supermodular function, there exists some edge $e \in E$ such that $x_e \geq 1/2$.*

Before proving this theorem, we need to define some concepts. A feasible solution for a set of linear inequalities in $\mathbb{R}^m$ is a BFS iff it satisfies m linearly independent inequalities with equality. Clearly, in the Steiner network problem, $m = |E|$. Given a solution x to LP(2), we say that an inequality is *tight* if it holds with equality. Further, if this inequality relates to the cut requirement of a set S, we say that set S *is tight.* For each set S we define its *incident vector* $\chi_{\delta(s)}$ as follows: if for $e_i \in E$ it holds that $e_i \in \delta(S)$, then the ith entry of $\chi_{\delta(s)}$ is 1, otherwise it is 0.

Definition 5 (Laminar collection)**.** *A collection $\{S_1, S_2, ...\}$ is called laminar if for every i, j the intersection of the sets S_i and S_j is either empty, or equals S_i, or equals S_j.*

We formulate the following theorem, whose proof can be found in [7].

Theorem 5. *For any BFS x to the LP(2) with f a weakly supermodular function, there is a collection $\mathcal{L}$ of subsets of vertices with the following properties:*

1. *For all $S \in \mathcal{L}$, S is tight.*

2. *The vectors* $\chi_{\delta(S)}$ *for* $S \in \mathcal{L}$ *are linearly independent.*

3. $|\mathcal{L}| = |E|$.

4. *The collection* $\mathcal{L}$ *is laminar.*

The first three properties of Theorem 5 follow trivially by analysing the polytope defined by LP(2) and a BFS x. Clearly, x lies at the intersection of m linearly independent tight constraints, where $m = |E|$.

A set $C \in \mathcal{L}$ is the *child* of $S \in \mathcal{L}$ if C is strictly contained in S and there exists no other set $C' \in \mathcal{L}$ containing C which is also strictly contained in S. In other words, S is the smallest set that strictly contains C. In the following, we give a proof by contradiction for Theorem (4).

Proof. Suppose that for all edges $e \in E$, $0 < x_e < 1/2$. The proof is divided into two parts. At first, we formulate a charging scheme in which we distribute charge to the sets $S \in \mathcal{L}$ and we show that the total charge will be strictly less than $|E|$. In the second part, we show that each $S \in \mathcal{L}$ receives a charge of at least one and, since by Theorem (5) $|\mathcal{L}| = |E|$, the total charge will be at least $|E|$. The statements inferred in these two parts will eventually lead to a contradiction.

Let us start with the first part of the proof. We formulate the following charging scheme. For each $e \in E$, we distribute a charge of $1-2x_e > 0$ to the smallest set $S \in \mathcal{L}$ that contains both endpoints of e, if such a set exists, and for each endpoint of e we give a charge of x_e to the smallest set $S \in \mathcal{L}$ that contains this endpoint, if such a set exists. Since we assumed that $0 < x_e < 1/2$, both $1 - 2x_e$ and x_e are positive. Further, for a given edge, the total charge that we distribute is at most $1 - 2x_e + x_e + x_e = 1$. By Theorem (5), the incidence vectors $\chi_{\delta(S)}$ are linearly independent. Thus, for any set $S \in \mathcal{L}$ which is not contained in any other set of the laminar collection, there must exist edges e that cross the cut $(S, \bar{S})$, i.e. edges $e \in \delta(S)$. Clearly, such edges do not have both endpoints contained in the same set $S \in \mathcal{L}$. Therefore, by our charging scheme, the charge $1 - 2x_e$ will not be distributed to any set. Since $0 < x_e < 1/2$, the charge given for e will be strictly less that 1, since we distribute at most $2x_e < 1$ units. It follows that the total charge distributed is strictly less than $|E|$.

In the second part of the proof we show that each set $S \in \mathcal{L}$ receives a charge of at least 1. Let $C_1, ..., C_k \in \mathcal{L}$ be the children of S, if any exist. By Theorem (5), S and C_i are all tight sets, i.e. $x(\delta(S)) = f(S)$ and $x(\delta(C_i)) = f(C_i)$ for all i. Let $C = \bigcup_i C_i$. We analyse the edges $e = \{u, v\}$ in $E_S = \delta(S) \cup \bigcup_i \delta(C_i)$ related to a set S, and divide them

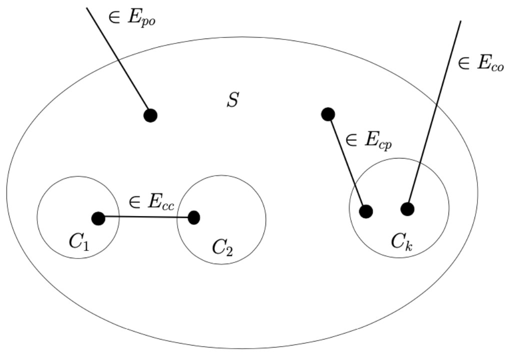

Figure 2: Illustration of the edge sets E_{cc}, E_{cp}, E_{co}, E_{po} (Source: [7])

into four disjunctive sets, which are illustrated in Figure 2. Following the charging scheme, we mention the contributed charge to S for each of the four sets:

1. E_{cc} ('child-child edges'): contains edges $e \in E_S$ for which $u \in C_i$ and $v \in C_j$, $i \neq j$. Since S is the smallest set containing both endpoints u and v, an edge $e \in E_{cc}$ contributes a charge of $1 - 2x_e$ to S.

2. E_{cp} ('child-parent edges'): contains edges $e \in E_S$ for which $u \in C_i$ and $v \in S - C$. Such edges give a charge of x_e to S for $v \in S - C$ and $1 - 2x_e$, because S is the smallest set containing both endpoints. Thus, for an edge $e \in E_{cp}$, the total charge distributed to S is $1 - x_e$.

3. E_{po} ('parent-out edges'): contains edges $e \in E_S$ for which $u \in S - C$ and v is outside the set S. An edge $e \in E_{po}$ gives a charge of x_e to S for u.

4. E_{co} ('child-out edges'): contains edges $e \in E_S$ for which $u \in C_i$ and v is outside the set S. An edge $e \in E_{co}$ contributes no charge to S.

Assume that all edges of E_S are in E_{co}. This means that an edge is in $\delta(S)$ if and only if it is in $\delta(C_i)$ for some i. But this implies that $\chi_{\delta(S)} = \sum_{i=1}^{k} \chi_{\delta(C_i)}$, which contradicts Theorem (5) that states that the incidence vectors $\chi_{\delta(T)}$ for a set $T \in \mathcal{L}$ are linearly independent. Notice that the sum is defined using modulo 2 addition.

Thus, it cannot be the case that all edges of E_S are in E_{co}. At least one edge must be in E_{cc}, E_{cp}, or E_{po}. By the rules defined above, each edge in one of these three sets contributes a positive charge to S. Thus, the total charge received by S is:

$$|E_{cc}| - 2x(E_{cc}) + |E_{cp}| - x(E_{cp}) + x(E_{po}) > 0.$$

Further, it holds that

$$x(\delta(S)) - \sum_{i=1}^{k} x(\delta(C_i)) = x(E_{po}) - x(E_{cp}) - 2x(E_{cc}).$$

Then, the total charge distributed to S is

$$|E_{cc}| + |E_{cp}| + \Big(x(\delta(S)) - \sum_{i=1}^{k} x(\delta(C_i))\Big) = |E_{cc}| + |E_{cp}| + \Big(f(S) - \sum_{i=1}^{k} f(C_i)\Big),$$

since all sets are tight. We notice that the last expression is a sum of integers and we already showed that the total charge is positive. Consequently, the total charge is at least one. It follows that each set $S \in \mathcal{L}$ receives a charge of at least 1, leading to a total charge of at least $|\mathcal{L}| = |E|$ for all sets in the laminar collection. However, in the first part of the proof, we showed that we distributed a total charge strictly less than $|E|$, which gives the contradiction.

□

Lemma 6. *For any $H \subseteq E$, we can solve the LP(2) in polynomial time with edge set $E - H$ and function $g(S) = f(S) - |\delta_H(S)|$ when $f(S) = max\{r(u,v) | u \in S, v \in \bar{S}\}$.*

Proof. Clearly, $g = f'$ is the residual cut requirement function. As already mentioned, we can solve LP(2) in polynomial time using the ellipsoid method with the separation oracle based on a max flow routine which was introduced in section 1. For the first iteration of the algorithm, given a solution x, we construct a graph on vertex set V, with capacity x_e for edge $e \in E$. For a subsequent iteration, given a solution x' to LP(2), we define x as $x_e = x'_e + e_H$ for all edges e, where e_H is the number of copies of e in the current set H.

Then, for all pairs of distinct vertices $u, v \in V$, we check whether the flow from u to v in this graph has value at least $r(u,v)$. If it has, then for any $u, v \in V$ and any cut $(S, \bar{S})$ with $u \in S$, $v \in \bar{S}$ the capacity is at least $r(u,v)$, which implies that $\delta_x(S) + |\delta_H(S)| \geq r(u,v)$, where $\delta_x(S) = \sum_{e: e \in \delta(S)} x_e$. Hence, $\delta_x(S) \geq r(u,v) - |\delta_H(S)|$. It follows that $\delta_x(S) \geq g(S)$, since the observation holds for all pairs of vertices. Otherwise, if the maximum flow between u and v has value less than $r(u,v)$, then, by MaxFlow/MinCut theorem, the capacity of the minimum cut $(S, \bar{S})$ is less than $r(u,v)$, which implies that $\delta_x(S) + |\delta_H(S)| < r(u,v)$ and, thus, $\delta_x(S) < r(u,v) - |\delta_H(S)|$. This is a violated constraint. □

Lemma 7. *A cut $(S, \bar{S})$ is violated by solution x' under cut requirement function f' iff it is violated by solution x under cut requirement function f.*

Proof. We know $|\delta_x(S)| = \delta'_x(S) + |\delta_H(S)|$. Since $f(S) = f'(S) + |\delta_H(S)|$, $\delta_x(S) \geq f(S)$ iff $\delta'_x(S) \geq f'(S)$. This implies that solution x' is feasible for the cut requirement function f' iff solution x is feasible for f. Moreover, the lemma implies that there is no need to update f' explicitly after each iteration. □

We can now formulate and prove Theorem 8, which characterizes the solution computed by Algorithm 1.

Theorem 8. *Algorithm 1 achieves an approximation guarantee of 2 for the Steiner network problem.*

Proof. Formally, the theorem can be proven using induction on the number of iterations executed by the algorithm when run with weakly supermodular residual functions f'. Such a proof can be found in [8]. In the following, we prove the theorem in a more intuitive way. Recall that a solution for the LP relaxation (2) gives a lower bound for the actual cost obtained by rounding up the edge-values computed by the algorithm in each iteration. The idea is to show that the cost of the integral solution is within a factor of 2 of the cost of the optimal fractional solution.

We denote with x_i^* the rounded solution computed in the ith iteration and with H_i the set of edges that are chosen by the algorithm in the ith iteration. The cost of the edges in H_i is defined as $\text{cost}(H_i) = \sum_{e \in H_i} c_e {x_i}_e^*$. Suppose that the algorithm needs k iterations to compute a feasible solution. This means that $\text{cost}(H_{k+1}) = 0$.

Since we round up variables by at most a factor of 2, for the first iteration it holds that $\text{cost}(x_1^*) \leq 2OPT$, where OPT is the cost of an optimal solution. In the second iteration, the algorithm solves a residual problem in which the edges and their cost included in the first iteration are not considered. Thus, $\text{cost}(x_2^*) \leq \text{cost}(x_1^*) - \text{cost}(H_1)$, which implies $\text{cost}(H_1)$ $\leq \text{cost}(x_1^*) - \text{cost}(x_2^*)$. If we generalize this statement, we get for the i-th iteration (where $i \in \{2, 3, ..., k\}$) that $\text{cost}(x_i^*) \leq \text{cost}(x_{i-1}^*) - \text{cost}(H_{i-1})$, which implies $\text{cost}(H_i) \leq \text{cost}(x_i^*)$ $- \text{cost}(x_{i+1}^*)$. Since the algorithm stops after k iterations, it holds that $\text{cost}(x_{k+1}^*) = 0$ and $\text{cost}(H_k) \leq \text{cost}(x_{k-1}^*)$. Now, for the cost of the final edge set $H = \bigcup_{i=1}^{k} H_i$ we obtain:

$$\text{cost}(H) \leq \text{cost}(H_1) + \text{cost}(H_2) + ... + \text{cost}(H_k) \leq \text{cost}(x_1^*) \leq 2OPT,$$

which concludes the proof.

□

4 Conclusion

In this paper, we presented the Steiner network problem and an iterated rounding algorithm that solves it achieving an approximation guarantee of 2. We also discussed that SNP addresses the questions that arise from designing low-cost telecommunications networks.

We think that SNP is a powerful classical network problem, since it can be easily extended to address other network-design questions. One of these is related to media broadcasting as in streaming services, where one wishes to connect a global server to a single local server in each community, which will then broadcast the stream to all clients in its community (Group Steiner Tree Problem). In the fault-tolerant version of this problem (Rooted Group SNP) [9], one wishes to design a low-cost network, such that there exist a specific number of edge-disjoint paths that connect the root node to a node in each community. Clearly, approximating the solution for such a problem involves more complex approaches [9] than the iterated rounding technique which solves SNP. However, SNP remains a classical problem that represents the foundation of numerous network-design problems.

References

[1] J. Krarup. The generalized steiner problem (unpublished note, 1978).

[2] P. Winter. Generalized steiner problem in outerplanar networks. *BIT 25*, 1985.

[3] R. Courant and Herbert Robbins. What is mathematics? An elementary approach to ideas and methods. 1941.

[4] George B. Dantzig and Mukund N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag, Berlin, Heidelberg, 1997.

[5] Steffen Rebennack. Ellipsoid method. *Encyclopedia of Optimization, Second Edition*, 2008.

[6] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21:39–60.

[7] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[8] Vijay V. Vazirani. *Approximation Algorithms*. Springer Publishing Company, Incorporated, 2010.

[9] Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.