Parameterized Complexity of Spare Capacity Allocation and the Multicost Steiner Subgraph Problem

Tibor Jordán[1] and Ildikó Schlotter[2]

[1] Department of Operations Research and the MTA-ELTE Egerváry Research Group on Combinatorial Optimization
Eötvös Loránd University
1117 Budapest, Pázmány Péter sétány 1/C.
jordan@cs.elte.hu

[2] Department of Computer Science and Information Theory
Budapest University of Technology and Economics
1117 Budapest, Magyar Tudósok körútja 2.
ildi@cs.bme.hu

Abstract. We study the computational complexity of the Spare Capacity Allocation problem arising in optical networks that use a shared mesh restoration scheme. In this problem we are given a network with edge capacities and point-to-point demands, and the goal is to allocate two edge-disjoint paths for each demand (a working path and a so-called restoration path, which is activated only if the working path fails) so that the capacity constraints are satisfied and the total cost of the used and reserved bandwidth is minimized. We focus on the setting where we deal with a group of demands together, and select their restoration paths simultaneously in order to minimize the total cost. We investigate how the computational complexity of this problem is affected by certain parameters, such as the number of restoration paths to be selected, or the treewidth of the network graph. To analyze the complexity of the problem, we introduce a generalization of the Steiner Forest problem that we call Multicost Steiner Subgraph. We study its parameterized complexity, and identify computationally easy and hard cases by providing hardness proofs as well as efficient (fixed-parameter tractable) algorithms.

1 Introduction

In this paper, we give efficient combinatorial algorithms as well as hardness results for optimization problems arising in restoration planning strategies of optical networks. An important aspect of Generalized Multi-Protocol Label Switching (GMPLS) networks, which has been extensively studied in the last decade [1, 2], is fast restoration of service after a network failure. We focus on restoration path selection in the design of a shared mesh restoration scheme, which is a key component of such strategies, since it determines the spare bandwith needed and hence also contributes to the required network resources and its total cost.

A restorable connection (Label-Switched Path, or LSP) in a GMPLS network supporting shared mesh restoration has a working path as well as a protection path. During normal network operation, the connection is established along the working path, with resources reserved along the protection path, which is activated when some link on the working path fails. A subset of links in the network that share the risk of failure at the same time are said to belong to a Shared Risk Link Group (SRLG): a failure of an SRLG means the failure of all links in the group. SRLGs can be used to model several types of failures, including single-link or single-node failures. For a connection to be restorable, the working path and

the protection path have to be SRLG-disjoint, i.e., no SRLG can contain links of both the working path and the protection path of the connection.

To minimize the total bandwidth needed on the links of the network, shared restoration schemes allocate the bandwidth necessary for protection paths in a shared manner: a certain amount of bandwidth ensures protection for several demands at the same time. However, the bandwidth reserved along the protection paths must be sufficient to recover all affected restorable connections in the event of any single SRLG-failure. Hence, to realize shared restoration, bandwidth is reserved along the protection paths in such a way that two protection paths can be assigned the same bandwidth on a link only if the corresponding working paths are SRLG-disjoint, that is, they are not expected to fail simultaneously.

Most path selection algorithms first select the working path as the shortest path between the endpoints of the demand, with respect to appropriately defined edge-costs, and then select the protection path, trying to maximize bandwidth sharing and hence minimize the additional bandwidth needed. Several protection path selection algorithms have been developed for the situation when one protection path needs to be determined for a single additional demand [23, 25]. These solutions provide different performance guarantees—some of them may overestimate the bandwidth that needs to be reserved on some links.

The algorithm most relevant to our approach is the Full Information Restoration (FIR) algorithm of Li, Wang, Kalmanek, and Doverspike [23]. Their algorithm is able to find an optimal solution for the single demand situation, where all working paths have been fixed. It can also be used to improve an existing solution (i.e. a complete list of path pairs for all demands) in a local search type algorithm, which replaces protection paths by better ones, one by one, whenever possible.

Our goal is to analyze the more general scenario, when we need to select protection paths for k new demands simultaneously, given that all the working paths as well as the protection paths of the existing demands are fixed. This approach has the following advantages:

- First, this simultaneous allocation problem can be thought of as a local search task: given a complete realization of the network (that is, a working and a protection path for each demand), is it possible to change the protection paths for a *subset of the demands* in a way that the total cost decreases? As modifying the working paths is usually infeasible, re-allocating some of the protection paths is probably the most natural approach in this setting. By repeating this procedure and re-allocating the protection paths for groups of demands iteratively, we can expect a significant decrease in the total cost of the network.
  Solving a hard optimization problem step-by-step through a sequence of such local improvements is the central idea of local search, a heuristic that is extremely useful in many real-world routing problems. In particular, it has been successfully applied in different capacity allocation problems [12, 29]. To reduce costs using this method in our model as well, as a subtask we have to solve the above problem repeatedly.
- Second, this simultaneous allocation problem can also be considered as the core task of a spare capacity allocation procedure in networks where demands appear in an online fashion and, after fixing the new working paths, we may deal with the protection paths in groups of k without violating time constraints. Allocating spare capacity for the protection paths in larger groups may lead to solutions which are better than what we can achieve by doing it one by one.
- Third, this problem also arises in the case when some SRLG fails. In such a situation, the demands whose working paths failed activate their protection paths. Thus, these paths become unprotected, and we have to find new protection paths for them. Furthermore, the failure might effect some protection paths directly as well, leading again to simultaneous re-allocation.

We shall explore the complexity status of several versions of this simultaneous allocation problem from the *fixed-parameter tractability* point of view, focusing on the cases where the number k of new demands and/or the *treewidth* of the graph is considered to be constant. We provide hardness results wherever the problem remains intractable even if some parameter is fixed, and develop efficient algorithms in the remaining cases. For example, we give a linear-time algorithm in the case when k and the treewidth are both small.

To analyze the simultaneous allocation problem, we also introduce the Multicost Steiner Subgraph problem. This problem is an extension of the well-known Steiner Forest problem, and may be of independent interest. Its input is an undirected graph with a set of terminal pairs, and different edge costs defined for each terminal pair. The task is to connect each terminal pair by a path, minimizing the total cost under the following assumption: if an edge e is used by several paths connecting different terminal pairs, each having a different cost on the edge e, then the cost of e is defined as the maximum among these values. We show how this problem is related to the aforementioned local search variant of the Spare Capacity Allocation problem we investigate. We examine its computational complexity and give positive as well as negative results for it.

The organization of the paper is the following. Section 2 describes the notation and provides the necessary definitions. Section 3 deals with the simultaneous allocation problem and its connection to the Multicost Steiner Subgraph problem. Sections 4.1 and 4.2 contain our contribution regarding Multicost Steiner Subgraph; in Section 4.1 we present two FPT-algorithms, while Section 4.2 discusses some hardness results. We finish with some concluding remarks and some ideas for future research in Section 5.

2 Preliminaries and problem definitions

2.1 Basic notation

In this paper, graphs are undirected and simple. We denote by $V(G)$ and $E(G)$ the vertex and edge set of a graph G, respectively. For a set X of vertices (or edges), we let $G - X$ denote the graph obtained by removing the vertices (or edges, respectively) of X from G. For a set $X \subseteq V(G)$, the subgraph of G induced by X is $G[X] := G - (V(G) \setminus X)$. For a set $F \subseteq E(G)$, we let $G[F]$ be the subgraph of G consisting of the edges in F and their endpoints. For a path P in G and an edge $e \in E(G)$, we will write $e \in P$ to denote that e is an edge of P.

Given a set H, a *partition* of H is a tuple $(H_1, \ldots, H_n)$ such that $H_1, \ldots, H_n$ are pairwise disjoint subsets of H whose union is H. We call each H_i a *block* of the partition.

2.2 Parameterized complexity

A *parameterized problem* contains pairs of the form (I, k) where I is the input instance and k is the *parameter*, usually an integer or a tuple of integers. In case the parameter is a pair (k_1, k_2), we will usually simplify the terminology by saying that both k_1 and k_2 are parameters.

An algorithm is *fixed-parameter tractable* or FPT, if its running time on an instance (I, k) is at most $f(k)|I|^{O(1)}$ for some computable function f; note that the degree of the polynomial $|I|^{O(1)}$ does not depend on the parameter k. A parameterized problem is FPT, if there is an FPT algorithm that decides it.

We say that a parameterized problem is contained in the class XP, if for each fixed value of the parameter it admits a polynomial-time algorithm. Note that the degree of this polynomial may depend on the value of the parameter, for instance the running time can

be $|I|^k$; such an algorithm is not fixed-parameter tractable, but it still proves that the given problem is in XP. Observe also that FPT $\subseteq$ XP is trivial.

Analogously to classical complexity theory, the theory of W[1]-*hardness* can be used to prove that some problem is not FPT, unless the widely believed FPT $\subset$ W[1] conjecture fails. Given two parameterized problems Q and Q', a *parameterized reduction* from Q to Q' maps each instance (I, k) to an instance (I', k') in at most $f(k)|I|^{O(1)}$ time such that $(I, k) \in Q$ if and only if $(I', k') \in Q'$, and $k' \leq g(k)$ for some computable function g of k. In this paper, we will prove W[1]-hardness of a problem Q by giving a parameterized reduction from the W[1]-hard parameterized problem Clique to Q. In Clique, we are given a graph G and a parameter $k \in \mathbb{N}$, and the task is to decide whether there is a clique (that is, a complete subgraph) of size k in G.

For further details on parameterized complexity, we refer the reader to [14, 17, 27].

2.3 Tree-decomposition and treewidth

Treewidth is a common notion to measure how "tree-like" a graph is. Given a graph G, a *tree-decomposition* $\mathbb{T}$ for G consists of a tree T and a *bag* $B_t \subseteq V(G)$ for each $t \in V(T)$ such that the following three conditions hold:

- for each vertex $v \in V(G)$ there is a bag that contains v,
- for each edge $uv \in E(G)$ there is a bag that contains both v and u, and
- for each $v \in V(G)$, the node set $\{t \in V(T) : v \in B_t\}$ induces a connected subtree of T.

The *size* of the tree-decomposition $\mathbb{T}$ is the number of vertices in T. The *width* of $\mathbb{T}$ is the maximum cardinality of any bag minus one. The *treewidth* of G is the minimum width of any tree-decomposition for G. Graphs having treewidth at most 1 are forests, and graphs having treewidth at most 2 are generalized series-parallel graphs.

When performing dynamic programming on a tree-decomposition $\mathbb{T}$ for G, we will consider T to be rooted at a root r. Furthermore, we will use *nice tree-decompositions* [8, 20], where each bag B_t is one of the following types:

- a *leaf bag*: t is a leaf of T and $|B_t| = 1$;
- a *bag introducing a vertex* $v \in V(G)$: t has one child x, $v \notin B_x$, and $B_t = B_x \cup \{v\}$;
- a *bag forgetting a vertex* $v \in V(G)$: t has one child x, $v \in B_x$, and $B_t = B_x \setminus \{v\}$;
- a *join bag*: t has two children x and y, and $B_t = B_x = B_y$.

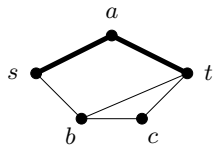
It is known that computing the treewidth of a graph is NP-complete [4]. However, for any constant w there is a linear-time algorithm by Bodlaender [6] that decides whether a graph G has treewidth at most w, and if so, constructs a tree-decomposition of width at most w. Furthermore, it is well-known that a linear-size tree-decomposition can be transformed into a nice tree-decomposition in linear time without changing its width [20]. For an introduction into treewidth see, e.g., [7].

2.4 Multiple demand networks

An *SRLG-network* is described by a quadruple $N = (G, g, c, \mathcal{R})$ where $G = (V, E)$ is an undirected graph with vertex set V and edge set E, the functions $g : E \to \mathbb{R}_0^+$ and $c : E \to \mathbb{R}_0^+$ represent non-negative edge *capacities* and edge *costs*, respectively, and the set $\mathcal{R} = \{R_1, \ldots, R_t\}$ contains so-called *SRLGs*, each being a subset of E. Each SRLG represents a set of edges that can fail simultaneously in the network.

A *demand* d in an SRLG-network is described by a triple (s, t, b) where $s, t \in V(G)$ and $b \in \mathbb{R}_0^+$. Here, s is the *source*, t is the *target*, and b is the required *bandwidth* of the

demand. Two paths P_1 and P_2 are said to be *SRLG-disjoint*, if they are edge-disjoint and no SRLG contains edges both from P_1 and P_2. A *realization of a demand* (s, t, b) consists of two SRLG-disjoint paths from s to t in G; one of them is called the *working path* (or service path), and the other one is the *protection path* (or restoration path). See Figure 1. An *SRLG-network with demands* is a pair (N, D), where N is an SRLG-network and D is a set of demands in N. A *realization of an SRLG-network with demands* (N, D), is the union of the realizations for all demands in D. We denote by $P(d)$ and $Q(d)$ the working and the protection paths assigned to some demand $d \in D$. We say that a demand d is *affected* by an SRLG R_i, if the working path $P(d)$ contains some edge of R_i.



SRLGs: $R_1 = \{at, bt\}$
$R_2 = \{sb, bt, bc\}$
$R_3 = \{bc, ct\}$

Fig. 1. The figure illustrates the concept of SRLG-disjoint paths in a simple network. Let $P = (s, a, t)$ be the working path (shown in bold) for the terminal pair (s, t). Notice that both (s, b, t) and (s, b, c, t) are edge-disjoint paths from P. However, since $R_1 = \{at, bt\}$ is an SRLG in the network, path (s, b, t) is not SRLG-disjoint from P. By contrast, path (s, b, c, t) is SRLG-disjoint from P, yielding the only possible protection path for P.

For each edge $e \in E$, we associate certain values with a given realization as follows. The *service bandwidth* of e, denoted by $p(e)$ is the value obtained by summing up the bandwidth values over those demands whose working path contains e. This value describes the bandwidth actively used by the demands routed through the edge e. To determine the additional bandwidth needed for the protection paths of the demands, we define the *spare bandwidth* $q_i(e)$ of an edge e *with respect to some SRLG* R_i as the sum of bandwidths over those demands that (i) are affected by R_i and (ii) have their protection path routed through e. This value describes the additional capacity necessary at edge e for the restoration of the network in case the i-th SRLG R_i fails.

We let the *spare bandwidth vector* of e be the vector $q(e)$ of length t, whose i-th component is $q_i(e)$. Now, the *total spare bandwidth* of an edge in the given realization, denoted by $q^{max}(e)$, is the maximal component of the vector $q(e)$. Thus, the total spare bandwidth of e describes the maximal bandwidth needed (in addition to the service bandwidth) in case any of the SRLGs fails. We define the *total bandwidth* $b(e)$ used by e as $p(e) + q^{max}(e)$.

Now, we are ready to define the feasibility and the cost of a realization Γ of an SRLG-network with demands. We say that the realization is *feasible*, if $b(e) \leq g(e)$ for each edge $e \in E$, that is, each edge can accommodate the working paths routed through it and has additional bandwidth sufficient for the restoration of the network in case any of the SRLGs fails. The *cost of a feasible realization* Γ is $c(\Gamma) = \sum_{e \in E} c(e)b(e)$. The cost of any non-feasible realization is defined to be $+\infty$.

2.5 Problem definitions

First, let us formally describe the Spare Capacity Allocation (SCA) problem. The input of this optimization problem is an SRLG-network with demands, (N, D), where $N = (G, g, c, \mathcal{R})$ and $D = \{d_i \mid 1 \leq i \leq \ell\}$, together with paths $P_1, \ldots, P_\ell$ in G, where for each $d_i = (s_i, t_i, b_i) \in D$, the path P_i leads from s_i to t_i. The task of the SCA problem is to find

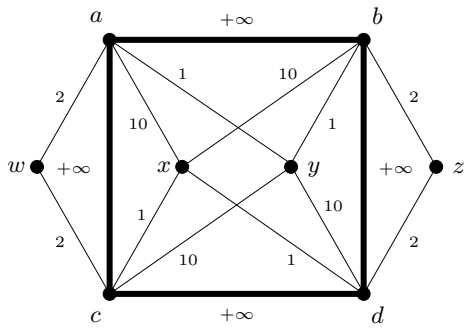
a minimum-cost realization for (N, D) in which the working path for the demand $d_i \in D$ is P_i, for each $1 \leq i \leq \ell$.

Motivated by the NP-hardness of the SCA problem (see e.g. [24]), we consider the following variant of SCA that we call k-IMPROVE SCA. We are given an SRLG-network with demands and its *partial realization* where each demand has a fixed working path, but the protection paths are only given for a subset of the demands. We call the demands for which the protection paths are given *protected*, and we refer to the remaining ones as *unprotected*. The number of unprotected demands, denoted by k, is assumed to be small compared to the total number of demands. The task is to find protection paths for the unprotected demands that yield a feasible realization while minimizing the cost.

Formally, the input of the k-IMPROVE-SCA problem consists of an SRLG-network $N = (G, g, c, \mathcal{R})$ with demand set $D = \{d_i \mid 1 \leq i \leq k\} \cup D^f$, paths $P_1, \ldots, P_k$ in G, and a realization for each demand in D^f. We refer to the demands in D^f as *fixed demands*, and to the demands in $D^u := D \setminus D^f$ as *unprotected demands*. The task of k-IMPROVE-SCA is to find a minimum-cost realization for (N, D) that uses the realizations given for the fixed demands in D^f, and for each unprotected demand $d_i \in D^u$ uses the path P_i as the working path. If Γ is such a realization, then we say that the protection paths $Q(d_i)$ for the demands $d_i \in D^u$ *induce* Γ.

An instance I of this problem is illustrated in Figures 2 and 3. In this example, there are two unprotected demands, d_1 and d_2, and there are six fixed demands. There are three SRLGs: $R_1 = \{ax, cx\}$, $R_2 = \{by, dy\}$, and $R_3 = \{aw, bz\}$. Note that d_1 is only affected by R_1, and d_2 is only affected by R_2. We define edge costs to be uniform. Furthermore, we set the capacities of the edges in such a way that only the edges ab, bd, cd, and ac can be used by the protection paths of demands d_1 and d_2. Thus, both of these paths (that is, $Q(d_1)$ and $Q(d_2)$) can only be routed in two different ways: either using the direct link, or using the remaining three edges (e.g., either (a, b) or (a, c, d, b) for $Q(d_1)$).

Figure 3(b) shows the solution we obtain for I by first minimizing the cost for d_1 and subsequently for d_2, or vice versa. In this case, $Q(d_1) = (a, b)$ and $Q(d_2) = (c, d)$; the additional cost of allocating capacity for these paths is 10. Figure 3(c) shows the optimum solution for the instance. Here, $Q(d_1) = (a, c, d, b)$ and $Q(d_2) = (c, a, b, d)$, and the additional cost of allocating the capacities for these paths is 8. Hence, this example shows that finding protection paths one by one can be suboptimal and thus leads to inefficient capacity allocation.



demand	working path	protection path	affecting SRLGs
$d_1 = (a, b, 5)$	(a, x, b)	?	R_1
$d_2 = (c, d, 5)$	(c, y, d)	?	R_2
$(a, x, 5)$	(a, x)	(a, b, x)	R_1
$(b, y, 1)$	(b, y)	(b, a, y)	R_2
$(c, x, 1)$	(c, x)	(c, d, x)	R_1
$(d, y, 5)$	(d, y)	(d, c, y)	R_2
$(a, c, 2)$	(a, w, c)	(a, c)	R_3
$(b, d, 2)$	(b, z, d)	(b, d)	R_3

Fig. 2. Illustrating the instance I of 2-Improve-SCA. The labels indicate capacities. The edges through which the protection paths of d_1 and d_2 can be routed are shown in bold.

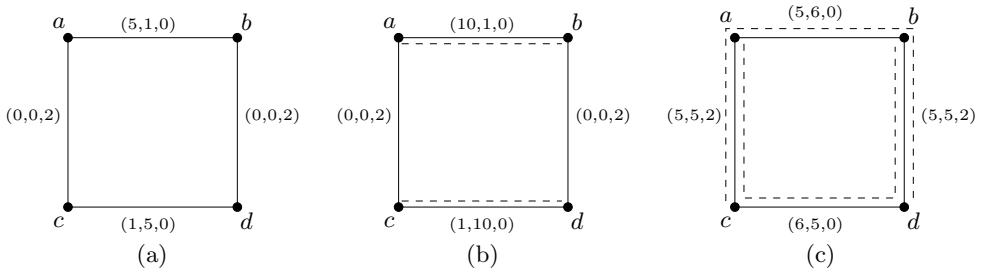


Fig. 3. (a) shows the relevant edges of the instance I, together with their spare bandwidth vector before finding the protection paths for d_1 and d_2. The spare capacity allocated in total for these edges is 14. (b) depicts the solution found by minimizing the cost separately for d_1 and for d_2; the spare bandwidth allocated here is 24 in total. (c) shows the optimal solution; the spare bandwidth allocated here is 22 in total.

We shall consider an important special case of k-Improve-SCA, in which the working paths $P_1, \ldots, P_k$ of the unprotected demands are pairwise SRLG-disjoint. In the next section we shall see that this special case turns out to be equivalent to another optimization problem that we call Multicost Steiner Subgraph and define as follows. The input of this problem is an undirected graph $G = (V, E)$ and k triples $\{(s_i, t_i, c_i) \mid 1 \leq i \leq k\}$ where s_i and t_i are vertices in G, and $c_i : E \to \mathbb{R}_0^+ \cup \{+\infty\}$ is a non-negative cost function on the edges for each i. We call s_i and t_i *terminals*, and we refer to c_i as the *cost function* corresponding to the terminal pair (s_i, t_i). The task of the Multicost Steiner Subgraph problem is to find a path Q_i from s_i to t_i for each $1 \leq i \leq k$ such that $\sum_{e \in E} \max_{i:e \in Q_i} c_i(e)$ is minimized. In other words, the cost of an edge e is defined by the cost function having maximum value on e among those cost functions c_i that correspond to terminal pairs (s_i, t_i) whose path Q_i contains e; our aim is to minimize the total cost of the edges. Figure 4 depicts a simple example. We will sometimes refer to edges e having $c_i(e) = +\infty$ as *forbidden edges* for the i-th terminal pair (s_i, t_i); note that including such an edge in Q_i yields a solution with cost $+\infty$.

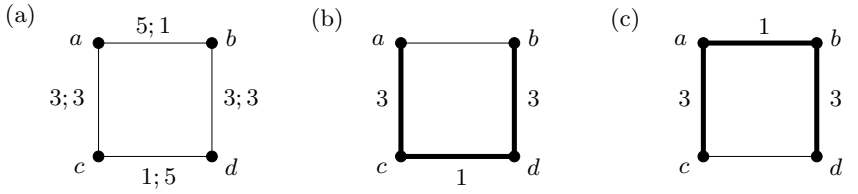


Fig. 4. (a) depicts an instance of Multicost Steiner Subgraph. Let the given terminal pairs be (a, b) and (c, d). The values of the corresponding cost functions are written on the edges, e.g., the cost of the edge ab is 5 w.r.t. to the terminal pair (a, b) and 1 w.r.t. the terminal pair (c, d). An optimal solution has total cost 8; the corresponding paths are indicated in bold in (b) and (c). Note that the union of the solution paths forms a 4-cycle.

3 The k-Improve-SCA problem

Let us consider an instance of the k-Improve-SCA problem, where we are given an SRLG-network $(G, g, c, \mathcal{R})$ and demand set $D = \{d_i \mid 1 \leq i \leq k\} \cup D^f$, together with paths

$P_1, \ldots, P_k$ in G and a realization for each demand in D^f. It is not hard to observe that this problem is NP-hard in general, even in a very restricted case.

Theorem 1. *The decision version of the k-*Improve-SCA *problem is NP-complete even in the special case when $k = 2$, $D^f = \emptyset$, $\mathcal{R} = \{R_1\}$, the cost function is arbitrarily fixed and*

(a) either $R_1 = \{r_1, r_2\}$ for some $r_1, r_2 \in E$, and each edge has capacity 1,
(b) or $R_1 = \{r\}$ for some $r \in E$ having capacity 2, and each edge in $E \setminus \{r\}$ has capacity 1.

Proof. Containment in NP is obvious. We shall show a polynomial-time reduction from the undirected version of the 2-Commodity Flow problem with unit capacities, which is NP-hard [16]. The input of the 2-Commodity Flow problem consists of an undirected graph G and two pairs (s_1, t_1) and (s_2, t_2) of vertices of G. The task in this problem is to find two edge-disjoint paths, one leading from s_1 to t_1 and the other one from s_2 to t_2.

Given an instance of 2-Commodity Flow as above, we construct two instances I_a and I_b of 2-Improve-SCA as follows. In both inputs, we fix an arbitrary cost function and we define the demand set to contain the two unprotected demands $d_1 = (s_1, t_1, 1)$ and $d_2 = (s_2, t_2, 1)$ (with $D^f = \emptyset$).

The graph G_a of I_a is obtained from G by adding newly introduced edges $r_1 = s_1t_1$ and $r_2 = s_2t_2$, and we let $R_1 = \{r_1, r_2\}$. Furthermore, we set each capacity in G_a to 1. We set the working paths P_1 and P_2 of d_1 and d_2 to be the paths consisting only of the edges r_1 and r_2, respectively. The graph G_b of I_b is obtained from G by adding two new vertices x, y and five new edges s_1x, s_2x, $r = xy$, yt_1, yt_2 to the graph G. We let $R_1 = \{r\}$, we set $g(r) = 2$, and we set each remaining capacity in G_b to 1. We let P_1 be the path s_1, x, y, t_1 and P_2 be the path s_2, x, y, t_2. We set $\mathcal{R} = \{R_1\}$ both for I_a and I_b. This completes the definitions of instances I_a and I_b.

It is easy to see that in both instances the protection paths $Q(d_1)$ and $Q(d_2)$ give rise to two edge-disjoint paths in the original graph G, one from s_1 to t_1 and one from s_2 to t_2, and vice versa. Hence, deciding whether the minimum cost of a solution for I_a (I_b, respectively) is $+\infty$ or less is equivalent with solving the given instance of the 2-Commodity Flow problem. This proves the theorem. □

As the general k-Improve-SCA problem is intractable even in a very restricted case, we pose an additional requirement on the input: we assume that the working paths given for the k unprotected demands are pairwise SRLG-disjoint. We show that in this case the problem becomes equivalent with the Multicost Steiner Subgraph problem with k terminal pairs.

Theorem 2. *The restriction of the k-*Improve-SCA *problem where the working paths given for the k unprotected demands are pairwise SRLG-disjoint is polynomially equivalent with the* Multicost Steiner Subgraph *problem with k terminal pairs.*

Proof. "k-SCA $\prec$ k-MCSS": For the first direction, let I_{SCA} be an input of the k-Improve-SCA problem with the claimed property. Let $(G = (V, E), g, c, \mathcal{R})$ be the SRLG-network, and let D^f and $D^u = \{d_1, \ldots, d_k\}$ be the set of fixed demands and the set of unprotected demands in I_{SCA}. Let $P(d)$ denote the given working path for each demand $d \in D^f \cup D^u$, and let $Q(d)$ be the given protection path for each fixed demand $d \in D^f$. Let $p^f(e)$ and $q^f(e) = (q^f_1(e), \ldots, q^f_{|\mathcal{R}|}(e))$ be the service bandwidth and the spare bandwidth vector on some edge e, respectively, corresponding to the realization of the fixed demands. We will also use $q^{f,max}(e) = \max_{R_i \in \mathcal{R}} q^f_i(e)$ to denote the total spare bandwidth on e. In addition, let $p^u(e)$ be the service bandwidth on e corresponding to the working paths of the unprotected demands.

We construct an instance I_{MCSS} of Multicost Steiner Subgraph consisting of the graph G and a triple (s_i, t_i, c_i) defined for each unprotected demand $d_i = (s_i, t_i, b_i)$ as follows. Let us fix an i between 1 and k. Let $\mathcal{R}(i) \subseteq \mathcal{R}$ denote the set of those SRLG-groups that affect the unprotected demand d_i. Furthermore, we define the value

$$b_i^{\Delta}(e) = \max(0, b_i - q^{f,max}(e) + \max_{j \in \mathcal{R}(i)} \{q_j^f(e)\})$$

for each edge e. Informally speaking, $b_i^{\Delta}(e)$ describes the additional capacity needed at edge e if we route the protection path of d_i through e.

Next, we define a set F_i of forbidden edges. To this end, let F_i^{SRLG} be the set of edges that are contained by an SRLG-group that affects d_i, and let F_i^{cap} be the set of those edges e for which $p^u(e) + p^f(e) + q^{f,max}(e) + b_i^{\Delta}(e) > g(e)$ holds. We set F_i to be the union of F_i^{SRLG} and F_i^{cap}, and define the cost $c_i(e) = +\infty$ for each forbidden edge $e \in F_i$. Finally, we let $c_i(e) = c(e)b_i^{\Delta}(e)$ for each remaining edge $e \in E \setminus F_i$, finishing the definition of I_{MCSS}.

Suppose that we have k paths $Q_1, \ldots, Q_k$, with Q_i leading from s_i to t_i for each $1 \leq i \leq k$. We claim that these paths form a minimum-cost solution for I_{SCA} if and only if they form a minimum-cost solution for I_{MCSS} as well.

First, we argue that these paths induce a feasible realization of all the demands in I_{SCA} if and only if Q_i avoids all edges in F_i, for each $1 \leq i \leq k$. Recall that Q_i yields a realization for d_i (together with the working path $P(d_i)$) if and only if Q_i and $P(d_i)$ are SRLG-disjoint. By the definition of F_i^{SRLG}, this means exactly that Q_i avoids all edges of F_i^{SRLG}. Thus, we may assume that this holds, and the paths $Q_1, \ldots, Q_k$ induce a realization Γ for all the demands in I_{SCA} (with the realization of the fixed demands as given in the input I_{SCA}).

It remains to show that Γ is feasible if and only if Q_i avoids F_i^{cap} as well. Let us fix an edge e. Let $q^{\Gamma}(e)$ be the spare bandwidth vector on e in Γ, with its j-th component denoted by $q_j^{\Gamma}(e)$. Since the working paths of the unprotected demands are pairwise SRLG-disjoint, for each SRLG-group R_j there is at most one unprotected demand affected by R_j. Now, if d_i is a demand affected by R_j and Q_i goes through e, then $q_j^{\Gamma}(e) = q_j^f(e) + b_i$, i.e., the j-th component of the spare bandwidth vector increases by b_i; otherwise (if no such demand exists) $q_j^{\Gamma}(e) = q_j^f(e)$. Therefore, the total spare bandwidth on e with respect to Γ is

$$\max_{1 \leq j \leq |\mathcal{R}|} q_j^{\Gamma}(e) = \max(q^{f,max}(e), \max_{i,j: e \in Q_i, j \in \mathcal{R}(i)} \{q_j^f(e) + b_i\}) = q^{f,max}(e) + \max_{i: e \in Q_i} b_i^{\Delta}(e).$$

This shows that Γ is feasible if and only if $p^u(e) + p^f(e) + q^{f,max}(e) + b_i^{\Delta}(e) \leq g(e)$ for each edge e and each i with $1 \leq i \leq k$. This is exactly the condition that Q_i avoids all edges from F_i^{cap} for any i, proving our claim on the feasibility of Γ.

Next, we show that a feasible realization Γ induced by paths $Q_1, \ldots, Q_k$ has minimum cost in I_{SCA} if and only if these paths form a minimum-cost solution in the instance I_{MCSS}. Clearly, the cost of Γ is

$$c(\Gamma) = \sum_{e \in E} c(e)(p^u(e) + p^f(e) + \max_{1 \leq j \leq |\mathcal{R}|} q_j^{\Gamma}(e)) = \sum_{e \in E} c(e)(p^u(e) + p^f(e) + q^{f,max}(e) + \max_{i: e \in Q_i} b_i^{\Delta}(e)).$$

As $p^u(e) + p^f(e) + q^{f,max}(e)$ does not depend on $Q_1, \ldots, Q_k$, such a cost is minimal if and only if $\sum_{e \in E} \max_{i: e \in Q_i} c(e) b_i^{\Delta}(e) = \sum_{e \in E} \max_{i: e \in Q_i} c_i(e)$ is minimal, which is exactly the cost of the solution $Q_1, \ldots, Q_k$ in I_{MCSS}. This proves our claim.

"k-MCSS $\prec$ k-SCA": For the other direction of the theorem, assume that we are given an instance I_{MCSS} of Multicost Steiner Subgraph with its input consisting of the graph $G = (V, E)$ and k triples $\{(s_i, t_i, c_i) \mid 1 \leq i \leq k\}$. We are going to construct an equivalent instance I_{SCA} of k-Improve-SCA, with the working paths for the unprotected demands being pairwise SRLG-disjoint. To begin, we define a graph $G' = (V, E')$ which is

obtained as follows. To the graph G we first add $k+1$ additional copies of the edge set E by introducing copies $e^1, \ldots, e^{k+1}$ for each edge $e \in E$ (connecting the same vertices as e, also present in G'), and then we add k more edges $r_1 = s_1t_1, \ldots, r_k = s_kt_k$. (In case we want to avoid parallel edges, we can introduce paths of length 2 instead of these edges; for simplicity we do not care about this issue.) For each $1 \leq i \leq k+1$, we write E^i for the edge set $\{e^i \mid e \in E\}$. In addition, for each $1 \leq i \leq k$ we define a set F_i that contains those edges $e \in E$ for which $c_i(e) = +\infty$, and we let $b_i = \max_{e \in E \setminus F_i} c_i(e)$. That is, b_i is the maximum value of c_i on any edge where c_i is finite.

Now, for each edge $e = xy \in E$ we introduce $k+1$ demands as follows. First, we define demands $d(e,i) = (x, y, M + c_i(e) - b_i)$ for each $1 \leq i \leq k$, for which $c_i(e)$ is finite; here M is a large enough integer such that each of these demands has non-negative bandwidth. We put $d(e,i) = (x, y, M)$, for $1 \leq i \leq k$, in case $c_i(e) = +\infty$. Second, we define the demand $d(e, k+1) = (x, y, M)$. These demands define the set of fixed demands $D^f = \{d(e,i) \mid e \in E, 1 \leq i \leq k+1\}$. For each fixed demand $d(e,i) \in D^f$, $1 \leq i \leq k+1$, we set e^i and e as the working and protection path (of length 1), respectively. Next, we define the set of unprotected demands as $D^u = \{(s_i, t_i, b_i) \mid 1 \leq i \leq k\}$, and we let the working path of the demand $d_i := (s_i, t_i, b_i)$ be r_i. We define the SRLG-network of I_{SCA} to be $(G', g, c \equiv 1, \mathcal{R} = \{R_1, \ldots, R_{k+1}\})$ where the capacity function g is defined to be b_i on the edges of r_i for each i, M on each edge $e^{k+1} \in E^{k+1}$, $M + c_i(e) - b_i$ on each edge $e^i \in E^i$, $1 \leq i \leq k$, for which $c_i(e)$ is finite, M on the edges e^i with $c_i(e) = +\infty$, and $+\infty$ on all the remaining edges (i.e. on the edges in E). Finally, the i-th SRLG-group is defined as $R_i = r_i \cup F_i \cup E^i$ if $i \leq k$, and we let $R_{k+1} = E^{k+1}$.

By definition, paths $Q_1, \ldots, Q_k$ with Q_i leading from s_i to t_i induce a feasible realization with finite cost for the demands $D = D^f \cup D^u$ in I_{SCA} if and only if each Q_i is SRLG-disjoint from r_i and does not exceed the given capacities. As the working paths defined for the demands in D already use up the total capacities on the edges in $E' \setminus E$, this latter condition means that all paths must only use edges from E. Note that the definition of the SRLG groups implies that in a finite cost solution a protection path Q_i must avoid all edges e with $c_i(e) = +\infty$. Hence, the given paths induce a feasible realization with finite cost if and only if, for each i, the path Q_i only contains edges from $E \setminus F_i$, meaning that $Q_1, \ldots, Q_k$ is a solution for I_{MCSS} having finite cost.

So let us consider paths $Q_1, \ldots, Q_k$ such that each edge in Q_i is from $E \setminus F_i$. The cost of these paths as a solution of the instance I_{MCSS} is $\sum_{e \in E} \max_{i:1 \leq i \leq k, e \in Q_i} c_i(e)$. Let us now consider the cost of the realization induced by the paths $Q_1, \ldots, Q_k$. First, as the working paths are fixed, this cost is minimal if and only if $\sum_{e \in E'} q^{max}(e)$ is minimal, where $q^{max}(e)$ is the total spare bandwidth on some edge $e \in E'$. Note that $q^{max}(e) = 0$ for each edge $e \in E' \setminus E$. For an edge $e \in E$ and some $1 \leq i \leq k+1$, the i-th component $q_i(e)$ of the spare bandwidth vector can be calculated as follows:

$$q_i(e) = \begin{cases} M & \text{if } i = k+1, \text{ or } c_i(e) = +\infty \\ M + c_i(e) - b_i & \text{if } 1 \leq i \leq k,\ c_i(e) \text{ is finite, and } Q_i \text{ does not go through } e \\ M + c_i(e) & \text{if } 1 \leq i \leq k \text{ and } Q_i \text{ goes through } e \end{cases}$$

Note also that for all $e \in E \setminus F_i$ we have $M + c_i(e) - b_i \leq M$ by the definition of b_i. Hence, we obtain that the cost of the realization induced by the paths $Q_1, \ldots, Q_k$ is minimal if the following expression is minimal:

$$\sum_{e \in E} q^{max}(e) = \sum_{e \in E} \max(M, \max_{i:1 \leq i \leq k, e \in Q_i} \{M + c_i(e)\}) = |E| \cdot M + \sum_{e \in E} \max_{i:e \in Q_i} c_i(e).$$

Hence, the realization induced by the paths $Q_1, \ldots, Q_k$ has minimum cost if and only if these paths form a minimum-cost solution for the I_{MCSS} instance. This finishes the proof of the theorem. □

Let us now remark that from the proof of Theorem 2 it follows that the restriction of k-Improve-SCA where the unprotected demands have pairwise SRLG-disjoint working paths can be reduced to an instance of Multicost Steiner Subgraph with the same underlying graph as in the SRLG-network given for k-Improve-SCA. Moreover, the presented reduction only changes the cost of a solution by an additive constant term.

4 The Multicost Steiner Subgraph problem

In this section we investigate the complexity of the Multicost Steiner Subgraph problem. Although various similar problems appear in the literature (see e.g. [3, 26, 28, 30]), the version relevant to us has not been studied before. To begin, we examine its strong connections to an important problem in combinatorial optimization, the Steiner Forest problem.

The input of Steiner Forest is an undirected graph G, possibly with positive edge costs, and a set of k *demands* $(s_1, t_1), \ldots, (s_k, t_k)$ where each demand is a pair of vertices (called *terminals*) of G. The task is to obtain a minimum-cost subgraph of G which contains a path from s_i to t_i for each $i \in \{1, \ldots, k\}$ and has minimum cost (or, in the unweighted case, minimum size). The optimal solution is always a forest, a so-called *Steiner forest* of the terminal pairs. Observe that Steiner Forest is exactly the special case of Multicost Steiner Subgraph where the cost functions belonging to the terminal pairs are the same.

If all demands contain a common terminal, then Steiner Forest becomes the classical Steiner Tree problem. Garey and Johnson [18] proved that unweighted Steiner Tree is NP-complete even for planar graphs. This implies that Multicost Steiner Subgraph is NP-complete even if $c_i \equiv 1$ for all $i \in \{1, \ldots, k\}$, and the input graph is planar.

The Steiner Forest problem turns out to be considerably harder than the Steiner Tree problem when considering its complexity on bounded-treewidth graphs. Namely, while Steiner Tree can be solved in linear time on bounded-treewidth graphs [10, 21], Steiner Forest remains NP-hard [19] even on graphs with treewidth 3.

Let us summarize the consequences of these facts in Proposition 1.

Proposition 1. *The* Multicost Steiner Subgraph *problem is NP-complete, even in the following cases:*
(a) the input graph is planar, and each cost function is the unit cost function;
(b) the input graph has treewidth 3*.*

Motivated by this intractability, we use the parameterized complexity approach to investigate the effect of several properties of the input on the computational complexity of Multicost Steiner Subgraph. Let $G = (V, E)$ and $\{(s_i, t_i, c_i) \mid 1 \leq i \leq k\}$ be our input instance. We will focus on the interplay between the following parameters:

- k, the number of terminal pairs;
- w, the treewidth of the input graph G;
- e_s, the number of so-called irregular edges: an edge $e \in E$ is *irregular*, if there are indices $1 \leq i < j \leq k$ such that $c_i(e) \neq c_j(e)$. In some sense, this parameter measures the pairwise distance of the given cost functions.

The parameter k, describing the number of terminal pairs, is probably the most natural parameterization of the problem. However, in Theorem 5 we are going to show that Multicost Steiner Subgraph is already NP-hard if $k = 2$. From the parameterized viewpoint, this means that Multicost Steiner Subgraph is not in XP when parameterized by k (unless P=NP). In fact, Theorem 5 contains a W[1]-hardness result for the $k = 2$ case, where

the parameter is the cost of the solution that we aim for. This means that the problem remains intractable even if there are only two terminal pairs, and we are looking for paths with small cost, and hence, of small length. Hence, Theorem 5 sharply contrasts the result that Steiner Forest is polynomial-time solvable for two (in fact, for any fixed number k of) terminal pairs; see e.g. [22].

Regarding the treewidth of the input graph, in Theorem 6 we prove that Multicost Steiner Subgraph is NP-hard on series-parallel graphs, that is, on graphs with treewidth 2. In some sense, this generalizes the result in [19], saying that Steiner Forest is NP-hard for graphs of treewidth 3. Our theorem yields a strict distinction between easy and hard cases when the treewidth w of the input graph is considered, since Multicost Steiner Subgraph is trivially linear-time solvable on forests, that is, on graphs with treewidth 1.

On the positive side, in Theorem 4 we propose an FPT algorithm for the case where both k and the treewidth w are regarded as parameters.

Looking into the hardness proofs in Theorems 5 and 6, we can observe that the hardness of the problem strongly relies on the fact that different terminal pairs have different cost functions. Therefore, it is interesting to examine how the difference of the cost functions influences the tractability of Multicost Steiner Subgraph. This motivates the study of our third possible parameter, the number e_s of irregular edges. In contrast to the intractability results mentioned above, in Theorem 3 we present a fixed-parameter tractable algorithm for the case where we regard both k and e_s as parameters.

Note that these results are strongest possible in the sense that the parameterization where only e_s, k, or w is considered as a parameter yields a parameterized problem that is not even in XP (unless P=NP), by the facts summarized in Proposition 1 and Theorem 5.

4.1 Fixed-parameter tractable algorithms for Multicost Steiner Subgraph

In this section, we give fixed-parameter tractable algorithms for solving the Multicost Steiner Subgraph problem. First, in Theorem 3 we propose an FPT algorithm for the case where the parameters are the number k of terminal pairs and the number e_s of irregular edges. Second, Theorem 4 provides an FPT algorithm for the problem when parameterized by k and the treewidth w of the input graph.

In the rest of this subsection, let I be our input instance of Multicost Steiner Subgraph, consisting of a graph $G = (V, E)$ and k triples $\{(s_i, t_i, c_i) \mid 1 \leq i \leq k\}$. We write $n = |V|$ and $m = |E|$. As before, e_s denotes the number of irregular edges in G, and w denotes the treewidth of G.

Theorem 3. Multicost Steiner Subgraph *can be solved in time*

$$\tilde{O}(3^{2k+2e_s}n + 2^{2k+2e_s}n^2 + nm) + 2^{O(ke_s + k\log k + e_s \log e_s)}.$$

Proof. Let E_s denote the set of irregular edges in our input, let X denote the set of terminals, and W the union of X and the end-vertices of all irregular edges. Note that a non-irregular edge can be assumed to have finite cost with respect to each cost function, as we can safely remove edges whose cost is infinite with respect to all cost functions.

Suppose that paths $Q_1, \ldots, Q_k$ form a minimum-cost solution. Let Q be the k-tuple $(Q_1, \ldots, Q_k)$, and let G_Q be the subgraph of G defined as the union of these paths. The main idea of the algorithm relies on the observation that after removing the irregular edges from G, the remainder of the solution, that is, $G_Q - E_s$ can be thought of as a collection of Steiner trees that connect vertices of W. Of course, we do not know which vertices of W belong to the same component in $G_Q - E_s$, neither do we know which of the irregular edges should be used by which paths; the algorithm tries all possibilities.

To capture the main structure of the solution Q, we introduce the following notation. For an irregular edge e let $c^*_Q(e)$ be the cost of e in Q, i.e., the maximum cost $c_i(e)$ taken over all indices i for which Q_i is routed through e. Also, we define the partition π_Q of W determined by the connected components of the graph $G_Q - E_s$. We call the pair (c^*_Q, π_Q) the *structure* of the solution Q.

From a high-level perspective, our strategy is the following. First, for each non-empty subset $Y \subseteq W$, we compute the cheapest Steiner tree in $G - E_s$ that connects the vertices of Y. Note that all cost functions coincide on the edges of $G - E_s$, so this is a well-defined task. Second, for each possible structure (c^*_Q, π_Q) of the solution, we check if a solution Q can be obtained by taking the union of the minimum-cost Steiner trees connecting the vertices of each block in the partition π_Q in $G - E_s$, and connecting them with irregular edges in a way that each irregular edge e can only be used by a path Q_i if $c_i(e) \leq c^*_Q(e)$. Finally, we take the solution having minimum cost.

Let us now describe the algorithm in detail.

Computing the Steiner trees. For a non-empty subset $Y \subseteq W$, let $T(Y)$ denote a minimum-cost Steiner tree (a tree containing each vertex in Y) in $G - E_s$, where the cost $c(e)$ of an edge $e \in E \setminus E_s$ is $c(e) = c_1(e) = \cdots = c_k(e)$. Let $c(Y)$ be the cost of such a tree.

Using the well-known Dreyfus–Wagner algorithm [15], $T(Y)$ and $c(Y)$ can be computed for each $Y \subseteq W$ in $\tilde{O}(3^{|W|}n + 2^{|W|}n^2 + nm)$ time. Here, n is the number of vertices, $m = |E \setminus E_s|$; the notation $\tilde{O}$ suppresses polylogarithmic factors. (We remark that if the edge costs are from the set $\{1, 2, \ldots, K\}$ for some integer K, this task can be done even in $\tilde{O}(2^{|W|}n^2 + nm)$ time applying recently developed techniques by Björklund et al. [5].)

Trying every possible structure. Next, the algorithm tries every possible structure (c^*, π) in order to find a solution Q where $c^*_Q = c^*$ and $\pi_Q = \pi$; such a solution is said to be *compatible* with (c^*, π). There are exactly k^{e_s} possible functions for choosing c^*, as we can choose the cost $c^*(e)$ in k different ways for each irregular edge e. There are at most $|W|^{|W|}$ different partitions for choosing π. Assuming that we are given the structure (c^*, π), we check whether there exists a solution Q compatible with (c^*, π), and if so, we find such a solution with minimum cost.

Checking the validity of a structure. To find out if there exists a solution compatible with (c^*, π), the algorithm proceeds as follows. For each i with $1 \leq i \leq k$, it computes a graph $H_i^{(c^*,\pi)}$. The blocks of the partition π form the vertex set of $H_i^{(c^*,\pi)}$; an edge connects two blocks A and B of π if and only if there are vertices $a \in A$ and $b \in B$ such that $e = ab$ is an irregular edge with $c_i(e) \leq c^*(e)$. It should be clear that if Q is a solution compatible with (c^*, π), then for each i, there must exist a path in $H_i^{(c^*,\pi)}$ connecting the vertices corresponding to the two blocks of the partition π that contain s_i and t_i. If this condition holds for each i, then we say that (c^*, π) is *valid*. Note that the validity of a solution structure can be checked in $O(k(|W| + |E_s|))$ time.

Finding a minimum-cost solution. Using the concept of validity, we claim that the cost c_{OPT} of a minimum-cost solution can be computed using the following formula:

$$c_{OPT} = \min_{(\pi,c^*) \text{ is valid}} \left\{ \sum_{P \in \pi} c(E(P)) + \sum_{e \in E_s} c^*(e) \right\}.$$

Running time analysis. With the above formula, the algorithm can compute the cost of an optimal solution in at most $k^{e_s}|W|^{|W|}O(k(|W|+|E_s|))$ time, once the Steiner trees are already computed. By $|W| \leq 2k + 2e_s$, this can be upper-bounded by $2^{O((k+e_s)(\log k + \log e_s))}$. Thus, the total running time of the algorithm is

$$\tilde{O}(3^{2k+2e_s}n + 2^{2k+2e_s}n^2 + nm) + 2^{O((k+e_s)(\log k + \log e_s))}.$$

It is straightforward to verify that the algorithm can also compute a minimum-cost solution itself in such a running time.

Note that the running time of our algorithm can be seen as the time used by the algorithm of [5] (for computing a minimum-cost Steiner tree), plus an additive term that is independent of the input size, and depends only on the parameters k and e_s.

Correctness of the formula. It remains to prove the correctness of the above formula. To do so, let us first observe that if Q is a minimum-cost solution, then (c^*_Q, π_Q) is valid. Also, the cost of Q is the cost of the irregular edges, that is, exactly $\sum_{e \in E_s} \max_{i:e \in Q_i} \{c_i(e)\} = \sum_{e \in E_s} c^*_Q(e)$, plus the cost of the remaining edges. The latter is the sum of the costs $c(e)$ for each edge e in $G_Q - E_s$. As the connected components of $G_Q - E_s$ form a collection of Steiner trees, each connecting the vertices of a block in the partition π, the total cost of these trees is at most the total cost of the corresponding minimum-cost Steiner trees, i.e. $\sum_{P \in \pi} c(E(P))$. This shows that c_{OPT} is at most the value defined by the right-hand side of the formula above.

As for the other direction, it suffices to observe that if (c^*, π) is valid, then we can construct a path Q_i in G from s_i to t_i that lies within the subgraph consisting of the union of the Steiner trees plus those irregular edges e for which $c_i(e) \leq c^*(e)$. This directly follows from our definition of validity and the definition of Steiner trees. Hence, the union of these paths forms a solution Q for the Multicost Steiner Subgraph instance given in the input; also, it is not hard to see that its cost is indeed at most $\sum_{P \in \pi} c(E(P)) + \sum_{e \in E_s} c^*(e)$. Hence, c_{OPT} is at most the right-hand side of the formula, proving the correctness of our algorithm. □

Theorem 4 shows that Multicost Steiner Subgraph becomes FPT, if we regard both the number k of terminal pairs and the treewidth w of the input graph as parameters.

Theorem 4. *There is an algorithm that solves* Multicost Steiner Subgraph *in linear FPT time, where the parameters are w and k. If a tree-decomposition for G is given together with the input, then the algorithms runs in $(I_{w+3})^{2k}O(n)$ time. Here, $I_{w+3} = O((w+3)^{(w+3)/2}e^{\sqrt{w+3}})$ is the number of matchings on $w+3$ vertices.*

Proof. We are going to present an algorithm using the standard dynamic programming approach on bounded treewidth graphs (see e.g. [9]).

To begin, our algorithm obtains a nice tree-decomposition $\mathbb{T}$ for the input graph G, such that $\mathbb{T}$ has width at most w and size $O(n)$. First, if there is no tree-decomposition for G given a priori, then we use the algorithm by Bodlaender [6] that decides whether a given graph has treewidth at most w, and if so, produces a tree-decomposition of width at most w. For any fixed w, this algorithm runs in linear time. Second, we can transform the obtained tree-decomposition into a nice tree-decomposition in $O(n)$ time without increasing its width [20].

To define partial solutions that we are looking for during the dynamic programming, we first need some additional notation. For some node $t \in V(T)$, we define $V_t = \{v \mid v \in B_x$ for some descendant x of t in $T\}$, and we associate the subgraph $G_t = G[V_t]$ with t. Given a set $\mathcal{P}$ of vertex-disjoint paths in a graph, their *shadow* is a matching which for each path $P \in \mathcal{P}$ contains an edge connecting the endpoints of P. Given two edge sets in a graph, they are *compatible* with each other, if they are disjoint, and their union induces a set of vertex-disjoint paths. By *joining* two compatible edge sets F and F', we mean taking the shadow of the paths induced by their union; the resulting matching is denoted by $F \oplus F'$.

Partial solutions and shadow patterns. We define a *partial solution* for t as a k-tuple $(P_1, \ldots, P_k)$ where each P_i is a collection of vertex-disjoint paths in G_t such that

- each path $P \in P_i$ has its endpoints in $B_t^{+i} := B_t \cup (\{s_i, t_i\} \cap V_t)$;

– if $v \in \{s_i, t_i\} \cap (V_t \setminus B_t)$, then there is a path $P \in P_i$ which has v as an endpoint;
– if both s_i and t_i lie on some path $P \in P_i$, then they are the endpoints of P and $P_i = \{P\}$.

Note that if Q_i is a path connecting s_i with t_i in G, then $G_t[Q_i]$ fulfills the above conditions. The *cost of a partial solution* $(P_1, \ldots, P_k)$ is defined as $\sum_{i=1}^{k} \sum_{e \in E(P_i)} c_i(e)$ where $E(P_i)$ is the set of those edges in E which are contained in some path of P_i.

The *shadow pattern* of a partial solution $(P_1, \ldots, P_k)$ is now defined as the k-tuple $(M_1, \ldots, M_k)$, where M_i is the shadow of the paths in P_i. Thus, for each $i \in \{1, \ldots, k\}$ by definition we have that

(A) M_i is a matching on the vertices of B_t^{+i};
(B) if $v \in \{s_i, t_i\} \cap (V_t \setminus B_t)$, then M_i must contain an edge incident to v;
(C) $s_i t_i \in M_i$ implies $M_i = \{s_i t_i\}$.

We let $\mathcal{S}_t$ be the set of all possible shadow patterns at t, i.e., the set of all k-tuples $(M_1, \ldots, M_k)$ where each M_i satisfies the properties (A)–(C). The *cost of a shadow pattern* $S \in \mathcal{S}_t$ is the minimum cost of a partial solution for t whose shadow pattern is S; we denote this value by $f_t(S)$. Clearly, the minimum cost of a solution for I is exactly $f_r(\{s_1 t_1\}, \ldots, \{s_k t_k\})$ where r is the root of T.

Now we are ready to describe the details of our algorithm. For each node t, we are going to compute a set $\hat{\mathcal{S}}_t$ of shadow patterns and a value $\hat{f}_t(S)$ for each $S \in \hat{\mathcal{S}}_t$. Later we will show $\hat{\mathcal{S}}_t = \mathcal{S}_t$ and $\hat{f}_t \equiv f_t$. We compute $\hat{\mathcal{S}}_t$ and $\hat{f}_t$ in a bottom-up manner, starting from the leaves of T, and ending at the root r. When creating a shadow pattern S with cost c at some node t, we mean adding it to $\hat{\mathcal{S}}_t$ and setting $\hat{f}_t(S) := \min(\hat{f}_t(S), c)$; initially, $\hat{f}_t(S)$ has value $+\infty$.

Let us describe our computation at some node $t \in V(T)$ depending on the type of t.

Leaf node. If t is a leaf, then $\hat{\mathcal{S}}_t$ only contains the "empty" shadow pattern $(\emptyset, \ldots, \emptyset)$ having cost zero.

The correctness of this step is trivial, so $\hat{\mathcal{S}}_t = \mathcal{S}_t$, and $\hat{f}_t \equiv f_t$ hold for each leaf node $t \in V(T)$. We are going to prove these facts for each node of T by induction. Hence, in the following we assume that they hold for the descendants of t.

Introduce node. If t is a node introducing some vertex v, then we perform the following operation for each shadow pattern $S' = (M'_1, \ldots, M'_k) \in \mathcal{S}_x$, where x is the unique child of t in T. For each $i \in \{1, \ldots, k\}$, we iterate over all possible choices for choosing a set A_i containing at most two edges from the set $\{vu \mid u \in B_t, vu \in E(G_t)\}$. Suppose we are processing the case when we pick the sets $A_1, \ldots, A_k$. For each $i \in \{1, \ldots, k\}$, we first check whether M'_i is compatible with A_i, and if so, then we compute the shadow $M_i = M'_i \oplus A_i$ and check if property (C) holds for it. If these steps are performed successfully for each i, then we create the shadow pattern $S = (M_1, \ldots, M_k)$. We define $c_A = \sum_{e \in A} \max_{i : e \in A_i} c_i(e)$ where $A = \bigcup_{i=1}^{k} A_i$, and we set the cost of S as $\hat{f}_x(S') + c_A$.

To prove correctness, first observe that M_i trivially satisfies property (A), and by induction we get (B) as well. Property (C) is ensured by the algorithm. This proves that $S \in \mathcal{S}_t$, and consequently, $\hat{\mathcal{S}}_t \subseteq \mathcal{S}_t$. Also, it is easy to see that given a minimum-cost partial solution $P' = (P'_1, \ldots, P'_k)$ for x whose shadow is S', we can construct a partial solution P for t with shadow S by adding the edges in A_i to P'_i for each $i \in \{1, \ldots, k\}$.[3] Observe that the cost of P is exactly $c_A + f_x(S')$. This implies that for each $S \in \hat{\mathcal{S}}_t$ there is a partial solution with shadow S whose cost is $\hat{f}_t(S)$, proving $\hat{f}_t(S) \geq f_t(S)$.

It remains to show $\hat{\mathcal{S}}_t \supseteq \mathcal{S}_t$ and $\hat{f}_t \leq f_t$.

[3] To be precise, P'_i contains the maximal paths of the subgraph of G obtained by taking the union of all paths in P_i together with the edges in A_i.

Suppose that $S = (M_1, \ldots, M_k) \in \mathcal{S}_t$ and $P = (P_1, \ldots, P_k)$ is a minimum-cost partial solution for t whose shadow is S. For each $i \in \{1, \ldots, k\}$, let A_i be the set containing those edges incident to v which lie on some path in P_i. Note that $0 \leq |A_i| \leq 2$. Furthermore, let P'_i be the set of paths obtained by deleting all edges in A_i from the paths in P_i; observe that $(P'_1, \ldots, P'_k)$ is a partial solution for x. Let M'_i be the shadow of P'_i for each $i \in \{1, \ldots, k\}$; then $M'_i \oplus A_i = M_i$ by the definitions. Hence, when the algorithm examines the shadow pattern $S' = (M'_1, \ldots, M'_k) \in \mathcal{S}_x$ and the choice of the sets $A_1, \ldots, A_k$ as described above, then it will indeed create the shadow pattern S. This proves $\hat{\mathcal{S}}_t \supseteq \mathcal{S}_t$, implying $\hat{\mathcal{S}}_t = \mathcal{S}_t$.

The cost of the edges in the partial solution P can be obtained as the cost of the edges in A, which is c_A by definition, plus the cost $c_{P'}$ of the remaining edges. As $c_{P'}$ is exactly the cost of the partial solution $(P'_1, \ldots, P'_k)$ for x, by induction we know $c_{P'} \geq f_x(S') = \hat{f}_x(S')$. Hence, $f_t(S) = c_A + c_{P'} \geq c_A + \hat{f}_x(S') \geq \hat{f}_t(S)$. This implies $f_t \equiv \hat{f}_t$.

Forget node. If t is a node forgetting some vertex v, then we proceed as follows. Let x be the unique child of t. For each $S \in \hat{\mathcal{S}}_x$, we either put S into $\hat{\mathcal{S}}_t$ with $\hat{f}_t(S) = \hat{f}_x(S)$, or ignore it. We keep some $S = (M_1, \ldots, M_k) \in \hat{\mathcal{S}}_x$, if the following condition holds for each $i \in \{1, \ldots, k\}$:

$$(\star) \quad v \in \{s_i, t_i\} \text{ if and only if } M_i \text{ contains an edge incident to } v.$$

In this case, it should be clear that $\mathcal{S}_t \subseteq \mathcal{S}_x$ and $f_t(S) = f_x(S)$ for each $S \in \mathcal{S}_t$. Thus, to prove the correctness of this step it suffices to show that we put a shadow pattern $(M_1, \ldots, M_k) \in \hat{\mathcal{S}}_x$ into $\hat{\mathcal{S}}_t$ exactly if M_i satisfies properties (A)–(C) with respect to the node t for each $i \in \{1, \ldots, k\}$. Property (C) holds by induction. Property (A) can only be violated if M_i contains an edge incident to v and $v \notin \{s_i, t_i\}$, and property (B) is violated only if $v \in \{s_i, t_i\}$ but no edge in M_i is incident to v. Therefore, the filtering condition $(\star)$ used by the algorithm indeed ensures $\hat{\mathcal{S}}_t = \mathcal{S}_t$, proving the correctness for this case.

Join node. If t is a join node, then let x and y be its two children. For each $S_x = (M^x_1, \ldots, M^x_k) \in \hat{\mathcal{S}}_x$ and $S_y = (M^y_1, \ldots, M^y_k) \in \hat{\mathcal{S}}_y$ we proceed by first checking if M^x_i is compatible with M^y_i for each $i \in \{1, \ldots, k\}$, and if so, we create the shadow pattern $S = (M_1, \ldots, M_k)$ where $M_i = M^x_i \oplus M^y_i$ for each i. If M_i violates property (C) for some $i \in \{1, \ldots, k\}$, then we ignore S. Otherwise, we put S into $\hat{\mathcal{S}}_t$ with cost $\hat{f}_x(S_x) + \hat{f}_y(S_y)$. See Figure 5 for an illustration of this case.

It is straightforward to prove $\hat{\mathcal{S}}_t \subseteq \mathcal{S}_t$ and $\hat{f}_t \geq f_t$ along the same lines as for introduce nodes: a minimum-cost partial solution for x and for y with shadow S_x and S_y, respectively, can be joined in order to obtain a partial solution for t with shadow S, having cost at most $f_x(S_x) + f_y(S_y)$; note that properties (A) and (B) hold for each M_i, and the algorithm ensures (C) as well.

So let us prove $\hat{\mathcal{S}}_t \supseteq \mathcal{S}_t$ and $\hat{f}_t \leq f_t$ now. Let $S \in \mathcal{S}_t$ be a shadow pattern and let $(P_1, \ldots, P_k)$ be a partial solution for t having shadow S and cost $f_t(S)$. For each $i \in \{1, \ldots, k\}$ we construct two sets of paths, P^x_i and P^y_i, as follows. We partition each path $P \in P_i$ as follows: we put the subpaths of P induced by V_x into P^x_i, and we put the remaining subpaths of P into P^y_i. Note that each path in P^y_i runs in G_y but avoids edges with both endpoints in B_t. In particular, no path in P^x_i shares an edge with a path in P^y_i.

It should be clear that $P^x = (P^x_1, \ldots, P^x_k)$ is a partial solution for x, and similarly, $P^y = (P^y_1, \ldots, P^y_k)$ is a partial solution for y. Furthermore, by joining the shadow M^x_i of P^x_i and the shadow M^y_i of P^y_i we obtain the shadow of P_i. Therefore, when the algorithm considers the shadow patterns $S_x = (M^x_1, \ldots, M^x_k) \in \mathcal{S}_x$ and $S_y = (M^y_1, \ldots, M^y_k) \in \mathcal{S}_y$, then it will produce our shadow pattern S. Hence we have $\hat{\mathcal{S}}_t \supseteq \mathcal{S}_t$, and consequently, $\hat{\mathcal{S}}_t = \mathcal{S}_t$. Note that $f_t(S)$ equals the cost of P^x plus the cost of P^y, which is at least $f_x(S_x) + f_y(S_y) \geq \hat{f}_t(S)$. This proves $f_t(S) \geq \hat{f}_t(S)$, implying $f_t \equiv \hat{f}_t$ as well. This finishes the proof of correctness for our algorithm.

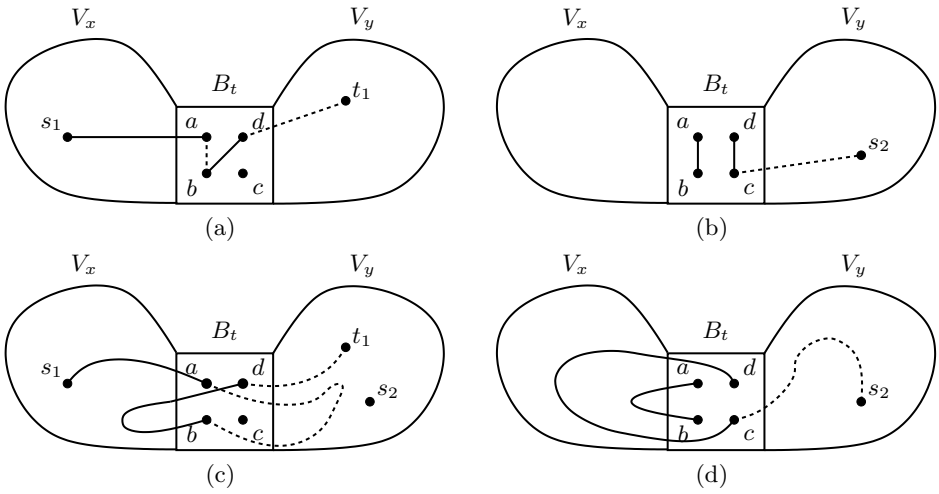


Fig. 5. Illustration of the case when t is a join node and $k = 2$. The example shows how the algorithm combines shadow patterns $(M_1^x, M_2^x) = (\{s_1a, bd\}, \{ab, cd\}) \in \mathcal{S}_x$ and $(M_1^y, M_2^y) = (\{ab, dt_1\}, \{cs_2\}) \in \mathcal{S}_y$ into a shadow pattern $(\{s_1t_1\}, \{ab, ds_2\})$. Panel (a) shows the combination of M_1^x and M_1^y, while panel (b) depicts M_2^x and M_2^y. Panels (c) and (d) illustrate the corresponding partial solutions. Solid (dashed) lines indicate the partial solution for x (for y, respectively) and the corresponding shadow pattern.

Running time. Obtaining the tree-decomposition for G takes linear FPT time, transforming it into a nice tree-decomposition is performed in $O(n)$ time.

To analyze the time spent at some node of the tree T, let I_ℓ denote the number of matchings in the complete graph on ℓ vertices, or equivalently, the number of *involutions*, that is, self-inverse permutations on ℓ elements; the asymptotic value of I_ℓ is described by $I_\ell \simeq c\left(\frac{\ell}{e}\right)^{\ell/2} e^{\sqrt{\ell}}$ where $c = \sqrt{2}e^{-1/4}$, see e.g. [11]. As a shadow pattern is a k-tuple of matchings, where each matching is defined on at most $w+3$ elements, we get $|\mathcal{S}_t| \leq (I_{w+3})^k$. The time spent at node t is at most $O(|\mathcal{S}_t|^2) = O((I_{w+3})^{2k})$. Thus the running time of our algorithm (after obtaining the tree-decomposition) is

$$(I_{w+3})^{2k} O(n) = O((w+3)^{(w+3)k} e^{2\sqrt{w+3}k} n),$$

as there are $O(n)$ nodes in the tree T. □

4.2 Hardness of Multicost Steiner Subgraph

In this section, we prove our hardness results for Multicost Steiner Subgraph.

Theorem 5. *The decision version of* Multicost Steiner Subgraph *where there are two terminal pairs and each cost is in* $\{1, +\infty\}$ *is*
(a) NP-complete, and
(b) W[1]-hard, if the parameter is the cost of an optimum solution.

Proof. To prove claim (b), we present a parameterized reduction from the W[1]-hard Clique problem to the special case of Multicost Steiner Subgraph with two terminal pairs as required by the theorem. Since the given reduction will be computable in polynomial time, this will also prove the NP-completeness result stated in (a).

Let $G = (V, E)$ be the input graph and k the parameter given as an instance of the parameterized Clique problem; we assume that G is simple. We construct an instance of

Multicost Steiner Subgraph as follows. The two terminal pairs are (s_1, t_1) and (s_2, t_2), and we denote the cost functions corresponding to the two terminal pairs by c_1 and c_2, respectively. To define the graph H underlying the instance, we construct k^2 *gadgets*; see Figure 6 for an illustration.

Each gadget is a subgraph $G^{i,j}$ of H for some $1 \leq i, j \leq k$. There will be only two types of gadgets: all gadgets $G^{i,j}$ with $i \neq j$ will be pairwise isomorphic and called *incidency-gadgets*, and similarly, all gadgets $G^{i,j}$ with $i = j$ will be pairwise isomorphic and called *identity-gadgets*. The vertex set and the edge set of these gadgets are as shown below.

$$
\begin{aligned}
V(G^{i,j}) &= \{a_x^{i,j}, b_x^{i,j}, c_x^{i,j}, d_x^{i,j} \mid x \in V\} \cup \{e_{x,y}^{i,j}, f_{x,y}^{i,j} \mid xy \in E\}, && \text{if } i \neq j,\\
E(G^{i,j}) &= \{e_{x,y}^{i,j} f_{x,y}^{i,j}, a_x^{i,j} e_{x,y}^{i,j}, f_{x,y}^{i,j} b_x^{i,j}, c_y^{i,j} e_{x,y}^{i,j}, f_{x,y}^{i,j} d_y^{i,j} \mid xy \in E\}, && \text{if } i \neq j,\\
V(G^{i,i}) &= \{a_x^{i,i}, b_x^{i,i}, c_x^{i,i}, d_x^{i,i}, e_{x,x}^{i,i}, f_{x,x}^{i,i} \mid x \in V\},\\
E(G^{i,i}) &= \{e_{x,x}^{i,i} f_{x,x}^{i,i}, a_x^{i,i} e_{x,x}^{i,i}, f_{x,x}^{i,i} b_x^{i,i}, c_x^{i,i} e_{x,x}^{i,i}, f_{x,x}^{i,i} d_x^{i,i} \mid x \in V\}.
\end{aligned}
$$

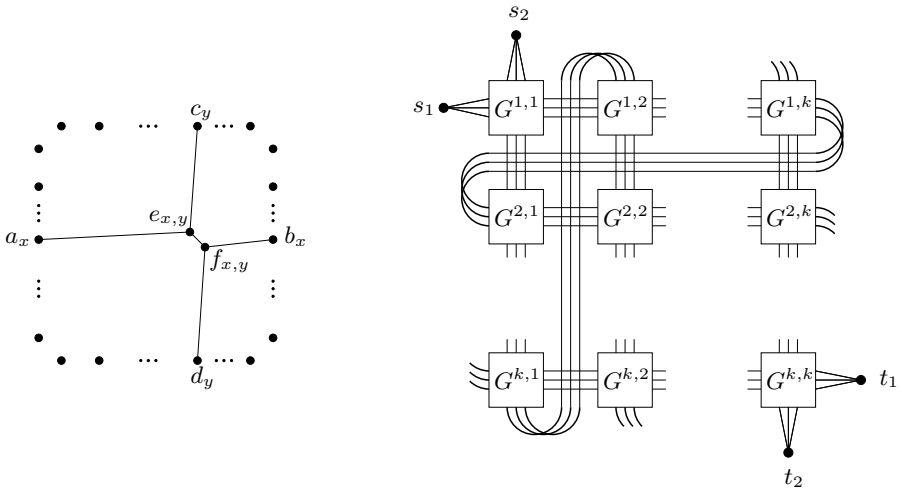


Fig. 6. An illustration of the graph H. Panel (a) depicts an incidency-gadget $G^{i,j}$ for some $i \neq j$. We assume $xy \in E$, and for simplicity, superscripts are omitted. Panel (b) shows how the k^2 gadgets are connected to each other and the four terminal vertices s_1, s_2, t_1, and t_2.

We arrange these gadgets into a $k \times k$ grid, with $G^{i,j}$ being placed at the intersection of the i-th row and the j-th column. We will call the following ordering of the gadgets their *horizontal ordering*: $G^{1,1}, G^{1,2}, \ldots, G^{1,k}, G^{2,1}, G^{2,2}, \ldots, G^{2,k}, \ldots, G^{k,1}, G^{k,2}, \ldots, G^{k,k}$. Similarly, we define their *vertical ordering* to be $G^{1,1}, G^{2,1}, \ldots, G^{k,1}, G^{1,2}, G^{2,2}, \ldots, G^{k,2}, \ldots, G^{1,k}, G^{2,k}, \ldots, G^{k,k}$.

The vertex set of the graph H consists of the vertices of the k^2 gadgets as defined above, plus the terminal vertices s_1, t_1, s_2, t_2. The edge set of H consists of the edges of the k^2 gadgets, together with the following three set of edges, defined below. The set $E_{st} = E_{st}^1 \cup E_{st}^2$ connects the source vertices with the gadget $G^{1,1}$ and the target vertices with the gadget $G^{k,k}$. The sets E_h and E_v connect the gadgets one by one, according to their horizontal and vertical ordering, respectively. The precise definition of these edge sets is the following.

$$E^1_{st} = \{s_1 a^{1,1}_x, b^{k,k}_x t_1 \mid x \in V\}$$
$$E^2_{st} = \{s_2 c^{1,1}_x, d^{k,k}_x t_2 \mid x \in V\}$$
$$E_h = \{b^{i,j}_x a^{i,j+1}_x \mid x \in V, 1 \leq i \leq k, 1 \leq j < k\} \cup \{b^{i,k}_x a^{i+1,1}_y \mid x, y \in V, 1 \leq i < k\}$$
$$E_v = \{d^{i,j}_x c^{i+1,j}_x \mid x \in V, 1 \leq i < k, 1 \leq j \leq k\} \cup \{d^{k,j}_x c^{1,j+1}_y \mid x, y \in V, 1 \leq j < k\}$$

It remains to define the cost functions c_1 and c_2 for each edge in H. First, we set $c_1(e) = 1$ and $c_2(e) = +\infty$ for each $e \in E_h \cup E^1_{st}$ and conversely, we set $c_1(e) = +\infty$ and $c_2(e) = 1$ for each $e \in E_v \cup E^2_{st}$. For each edge e inside some gadget, we set $c_1(e) = c_2(e) = 1$. This completes the definition of our instance of Multicost Steiner Subgraph which we call I_{MCSS}.

Before going into details, let us give some intuition about the reduction. Our construction will ensure that any solution for I_{MCSS} consists of two paths with the following properties. First, each gadget is used by both paths, but one of them traverses the gadgets in their horizontal ordering, while the other path follows the vertical ordering. Second, in order to fit into a certain budget, these two paths have to use a common edge in each of the gadgets. The way how the "horizontal path" enters the k gadgets in the i-th row corresponds to picking the i-th vertex of the clique. Analogously, the way how the "vertical path" enters the k gadgets in the j-th column, encodes the j-th vertex of the clique. The identity gadget $G^{i,i}$ ensures that the i-th vertex encoded both by the horizontal and the vertical path is well-defined. Finally, the incidency-gadget $G^{i,j}$ checks that the i-th and j-th vertices of the desired clique are indeed adjacent.

To proceed more formally, observe first that the instance I_{MCSS} indeed does not contain any forbidden edges, and each cost is an integer. Moreover, it can clearly be constructed in polynomial time. Thus, to show that the reduction is correct, it suffices to show the following.

Claim. The instance I_{MCSS} has a solution with cost at most $7k^2 + 2$ if and only if the graph G has a clique of size k.

Suppose first that a solution exists for I_{MCSS} with cost at most $7k^2 + 2$. Let Q_1 and Q_2 denote the two paths in the solution connecting the given terminal pairs. By the definition of the cost functions and our budget, we immediately have that Q_1 cannot contain edges from $E_v \cup E^2_{st}$, and Q_2 cannot contain edges from $E_h \cup E^1_{st}$. Thus, looking at the structure of the graph $H - (E_v \cup E^2_{st})$, we can see that in order to reach t_1 starting from s_1, the path Q_1 must go through each gadget one by one, traversing all the gadgets according to their horizontal ordering. Similarly, path Q_2 lies entirely in $H - (E_h \cup E^1_{st})$, and so it must traverse all the gadgets according to their vertical ordering. Thus, each of the paths Q_1 and Q_2 must contain at least $k^2 + 1$ edges from $E_h \cup E_v \cup E_{st}$, and we get that the cost of those edges of the solution that are contained inside some gadget can be at most $7k^2 + 2 - 2(k^2 + 1) = 5k^2$.

However, it is not hard to see that Q_1 and Q_2 must each contain at least three edges from any gadget, and morerover, they cannot share their first and last edges in the gadget. This follows from the definition of the gadget, and the fact that in $G^{i,j}$, the path Q_1 must connect some vertex $a^{i,j}_x$ with some vertex $b^{i,j}_y$, while the path Q_2 must connect some vertex $c^{i,j}_v$ with some vertex $d^{i,j}_z$. As this requires at least five edges in any gadget, this means that any solution with total cost at most $7k^2 + 2$ must contain exactly five edges from each gadget. This immediately implies that given a gadget $G^{i,j}$, the subpath of Q_1 in $G^{i,j}$ must be $a^{i,j}_x, e^{i,j}_{x,y}, f^{i,j}_{x,y}, b^{i,j}_x$ and the subpath of Q_2 in $G^{i,j}$ must be $c^{i,j}_y, e^{i,j}_{x,y}, f^{i,j}_{x,y}, d^{i,j}_y$ for some x and y. Let us define $\sigma(i, j)$ to be the pair (x, y) for this x and y.

Observe that by the definition of E_h, we get that $\sigma(i, j) = (x, y)$ implies $\sigma(i, j + 1) = (x, y')$ for some y' if $j < k$; similarly, the edges in E_v show that $\sigma(i, j) = (x, y)$ also implies

$\sigma(i+1,j) = (x', y)$ for some x' if $i < k$. Hence, we can define $\sigma_h(i)$ to be the unique vertex x if each of $\sigma(i,1), \sigma(i,2), \ldots, \sigma(i,k)$ has x as the first component, and similarly, we can define $\sigma_v(j)$ to be the unique vertex y if each of $\sigma(1,j), \sigma(2,j), \ldots, \sigma(k,j)$ has y as the second component. This way we get $\sigma(i,j) = (\sigma_h(i), \sigma_v(j))$.

Now, by the definition of the gadgets, in case of an incidency-gadget (when $i \neq j$) we know that $\sigma_h(i)$ and $\sigma_v(j)$ must be adjacent vertices in the graph G (the simplicity of G implying also that they cannot coincide), and in case of an identity-gadget (when $i = j$) we must have $\sigma_h(i) = \sigma_v(j)$. This means that $\sigma_h(1) = \sigma_v(1)$, $\sigma_h(2) = \sigma_v(2)$, $\ldots$, $\sigma_h(k) = \sigma_v(k)$, and these k vertices must form a clique in G.

For the other direction, suppose that $v_1, v_2, \ldots, v_k$ form a clique in G. It is straightforward to verify that taking Q_1 to be the path induced by the edges

$$\{s_1 a^{1,1}_{v_1}, b^{k,k}_{v_k} t_1\} \cup \{a^{i,j}_{v_i} e^{i,j}_{v_i v_j}, e^{i,j}_{v_i v_j} f^{i,j}_{v_i v_j}, f^{i,j}_{v_i v_j} b^{i,j}_{v_i} \mid 1 \leq i, j \leq k\}$$

$$\cup \{b^{i,j}_{v_i} a^{i,j+1}_{v_i} \mid 1 \leq i \leq k, 1 \leq j < k\} \cup \{b^{i,k}_{v_i} a^{i+1,1}_{v_{i+1}} \mid 1 \leq i < k\},$$

and taking Q_2 to be the path induced by the edges

$$\{s_2 c^{1,1}_{v_1}, d^{k,k}_{v_k} t_2\} \cup \{c^{i,j}_{v_j} e^{i,j}_{v_i v_j}, e^{i,j}_{v_i v_j} f^{i,j}_{v_i v_j}, f^{i,j}_{v_i v_j} d^{i,j}_{v_j} \mid 1 \leq i, j \leq k\}$$

$$\cup \{d^{i,j}_{v_j} c^{i+1,j}_{v_j} \mid 1 \leq i < k, 1 \leq j \leq k\} \cup \{d^{k,j}_{v_j} c^{1,j+1}_{v_{j+1}} \mid 1 \leq j < k\},$$

we obtain a solution for I_{MCSS} with cost exactly $7k^2 + 2$. □

Theorem 6. *The decision version of* Multicost Steiner Subgraph *is NP-complete if the underlying graph is a series-parallel graph, i.e., it has treewidth* 2*. In fact, NP-hardness holds even for the case where the input graph is the subdivision of four parallel edges.*

Proof. We present a polynomial-time reduction from the Multiway Cut problem with three terminals and unit costs. In this problem, we are given an undirected graph $G = (V, E)$ with three terminal vertices v_1, v_2, v_3, and an integer b; the task is to delete at most b edges from G to separate each of the terminals from the other two. This problem is NP-complete [13].

Given the input I_{MC} of Multiway Cut as above, let $V = \{v_1, \ldots, v_n\}$ and $|E| = m$. We construct an instance I_{MCSS} of Multicost Steiner Subgraph with $k = n + m$ terminal pairs as follows. The graph H underlying I_{MCSS} is the series-parallel graph obtained by taking two vertices s and t, and connecting them with four innerly disjoint paths. Three of these paths, P^1, P^2, and P^3 have length n; the fourth one, denoted by R, has length m. We refer to the i-th edge on P^j as e^j_i for any $i \in \{1, \ldots, n\}$ and $j \in \{1, 2, 3\}$. Also, for each $v_h v_j \in E$ we assign a unique edge $r_{h,j}$ on R; the order of these edges on R does not matter.

The instance I_{MCSS} will consist of the graph H together with $n + m$ triples: each vertex v_i of V defines a triple (s, t, c_i), and each edge $v_h v_j$ of E defines a triple $(s, t, c_{h,j})$. We define the cost function to be

$$c_i(e) = \begin{cases} b+1, & \text{if } e = e^i_i, \\ 0, & \text{if } e \in P^i \setminus \{e^i_i\}, \\ +\infty, & \text{otherwise,} \end{cases} \qquad \text{for a terminal } v_i,\ i \in \{1,2,3\};$$

$$c_j(e) = \begin{cases} b+1, & \text{if } e = e^i_j \text{ for some } i \in \{1,2,3\}, \\ +\infty, & \text{if } e \in R, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for a non-terminal vertex } v_j;$$

$$c_{h,j}(e) = \begin{cases} 1, & \text{if } e = r_{h,j}, \\ b+1, & \text{if } e \in \{e^i_h, e^i_j\} \text{ for some } i \in \{1,2,3\}, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for an edge } v_h v_j \in E.$$

Finally, we set our budget to $B = n(b+1) + b$. This finishes the construction of I_{MCSS}. Clearly, the reduction is polynomial-time computable, it remains to argue that it is correct.

For the first direction, let Q be a solution for I_{MCSS} having cost at most B. Notice that for each of the n triples in I_{MCSS} corresponding to a vertex in V, the solution Q has to pick a path connecting s with t via one of the paths P^1, P^2, or P^3. Thus, Q naturally partitions the vertices of G into three groups: for $j \in \{1,2,3\}$, we denote by V^j the set of vertices in G whose corresponding triple is assigned the path P^j. Clearly, (V^1, V^2, V^3) indeed is a partition of V. Moreover, we must have $v_1 \in V^1$, $v_2 \in V^2$, and $v_3 \in V^3$ by the definition of the cost functions c_1, c_2, and c_3.

We claim that the edges running between different blocks of the partition (V^1, V^2, V^3) yields a solution for the Multiway Cut instance I_{MC}. To show this, first observe that no matter how the solution Q routes the i-th triple (s, t, c_i), its path assigned by Q must indicate a cost of $b+1$ at exactly one of the edges $\{e_i^1, e_i^2, e_i^3\}$. Thus, routing all triples corresponding to vertices of G already implies a cost of $n(b+1)$.

Now, let us consider a triple $(s, t, c_{h,j})$ corresponding to the edge $v_h v_j$ of G. Notice that if the path assigned by Q to this triple is P^i for some $i \in \{1,2,3\}$, then the edges e_h^i and e_j^i will have cost $b+1$ in Q. As $(n+1)(b+1) > B$, these edges must already be used by the paths assigned to the triples (s, t, c_h) and (s, t, c_j), since otherwise Q would exceed the budget because of using an additional edge with cost $b+1$. Therefore, in this case we know that $\{v_h, v_j\} \in V^i$. That is, if a triple corresponding to an edge $v_h v_j$ is not routed through R, then we know that its two endpoints must be in the same block of the partition (V^1, V^2, V^3). Since routing a triple $(s, t, c_{h,j})$ through R raises a cost of 1 on the edge $r_{h,j}$, there can be at most b such triples routed through R. Hence, there are at most b edges of G whose endpoints are not in the same block of the partition (V^1, V^2, V^3), yielding a solution of cost at most b for I_{MC}.

For the other direction of the proof, let F be a subset of at most b edges in G whose removal separates each terminal from the others. We can define a partition (V^1, V^2, V^3) of V such that $v_i \in V^i$ for each $i \in \{1,2,3\}$, and each connected component of $G - F$ is entirely contained in some block of the partition. To finish the proof of the theorem, it suffices to check that the following solution for I_{MMC} has cost at most B.

- For each vertex v_j, we route the path assigned to the triple (s, t, c_j) through P^i, where V^i is the block containing v_j. These n paths have a total cost of $n(b+1)$.
- For each $v_h v_j \notin F$ such that $\{v_h, v_j\} \subseteq V^i$, we route the path assigned to the triple $(s, t, c_{h,j})$ through P^i. These paths do not induce any additional cost (when the paths routed in the previous step had already been considered).
- For each $v_h v_j \in F$, we route the path assigned to the triple $(s, t, c_{h,j})$ through R. These $|F|$ paths imply an additional cost of $|F| \leq b$.

It is easy to see that this solution for I_{MCSS} has total cost at most B, as required. □

5 Concluding remarks

We examined the computational complexity of a variant of the Spare Capacity Allocation problem where we want to find protection paths for a group of demands simultaneously, minimizing the total cost of these paths. We investigated its close relation to a natural generalization of the classical Steiner Forest problem which we called Multicost Steiner Subgraph, and applied the framework of fixed-parameter tractability to deal with the computational intractability of this problem. We proved strong hardness results, and proposed efficient FPT algorithms for the remaining cases. In particular, we gave a linear-time algorithm for the case where both the number of protection paths to be found and the treewidth of the network graph is a fixed constant.

There are several possibilities for future research. An interesting question is whether one can find further parameters in the Spare Capacity Allocation problem that have small values in real-world instances, but yield fixed-parameter tractability. Identifying such parameters would lead to efficient algorithms in practice.

Another possible direction is to examine the Multicost Steiner Subgraph from a different angle, and develop approximation algorithms, or exact exponential-time algorithms for it.

Acknowledgements

We thank the anonymous referees for carefully reading our manuscript and for several useful suggestions. This work was supported by the Hungarian Scientific Research Fund grants nos. K81472, K109240, K108383, and K108947.

References

1. Generalized multi-protocol label switching (GMPLS) signalling functional description, 2003. IETF RFC 3471.
2. Generalized multi-protocol label switching (GMPLS) architecture, October 2004. IETF RFC 3945.
3. A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.
4. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8(2):277–284, 1987.
5. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: Fast subset convolution. In *STOC 2007: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 67–74. ACM, 2007.
6. H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
7. H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS 1997*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997.
8. H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21:358–402, 1996.
9. H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.
10. M. Chimani, P. Mutzel, and B. Zey. Improved Steiner tree algorithms for bounded treewidth. In *IWOCA 2011*, pages 374–386. Springer, 2011.
11. S. Chowla, I. N. Herstein, and W. K. Moore. On recursions connected with symmetric groups I. *Canad. J. Math.*, 3:328–334, 1951.
12. O. Crochat and J.-Y. Le Boudec. Design protection for WDM optical networks. *IEEE Journal on Selected Areas in Communications*, 16(7):1158–1165, 2006.
13. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.
14. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999.
15. S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971.
16. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.
17. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, New York, 2006.
18. M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32:826–834, 1977.
19. E. Gassner. The Steiner Forest Problem revisited. *Journal of Discrete Algorithms*, 8(2):154–163, 2010.

20. T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
21. E. Korach and N. Solel. Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth. Technical Report 632, Israel Institute of Technology, 1990.
22. K. J. Lai, C. P. Gomes, M. K. Schwartz, K. S. McKelvey, D. E. Calkin, and C. A. Montgomery. The Steiner multigraph problem: Wildlife corridor design for multiple species. In *AAAI 2011*, pages 1357–1364. AAAI Press, 2011.
23. G. Li, D. Wang, C. Kalmanek, and R. Doverspike. Efficient distributed restoration path selection for shared mesh restoration. *IEEE/ACM Transactions on Networking*, 11:761–771, 2003.
24. Y. Liu. *Spare Capacity Allocation: Model, Analysis and Algorithm*. PhD thesis, Sch. Information Sciences, Univ. Pittsburgh, PA, 2001.
25. Y. Liu, D. Tipper, and P. Siripongwutikorn. Approximating optimal spare capacity allocation by successive survivable routing. *IEEE/ACM Transactions on Networking*, 13:198–211, 2001.
26. I. Murthy and S.-S. Her. Solving min-max shortest-path problems on a network. *Naval Research Logistics (NRL)*, 39(5):669–683, 1992.
27. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
28. M. B. Richey and R. G. Parker. On multiple Steiner subgraph problems. *Networks*, 16(4):423–438, 1986.
29. C.-C. Shyur, T.-C. Lu, and U.-P. Wen. Applying tabu search to spare capacity planning for network restoration. *Computers & Operations Research*, 26(12):1175 – 1194, 1999.
30. P. Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.