Parameterized complexity of length-bounded cuts and multi-cuts [⋆]

Pavel Dvořák and Dušan Knop[⋆⋆]

Department of Applied Mathematics, Charles University, Prague
{koblich,knop}@kam.mff.cuni.cz

Abstract. We show that the Minimal Length-Bounded L-But problem can be computed in linear time with respect to L and the tree-width of the input graph as parameters. In this problem the task is to find a set of edges of a graph such that after removal of this set, the shortest path between two prescribed vertices is at least L long. We derive an FPT algorithm for a more general multi-commodity length bounded cut problem when parameterized by the number of terminals also.
For the former problem we show a W[1]-hardness result when the parameterization is done by the path-width only (instead of the tree-width) and that this problem does not admit polynomial kernel when parameterized by tree-width and L. We also derive an FPT algorithm for the Minimal Length-Bounded Cut problem when parameterized by the tree-depth. Thus showing an interesting paradigm for this problem and parameters tree-depth and path-width.

Keywords: length bounded cuts, parameterized algorithms, W[1]-hardness

1 Introduction

The study of network flows and cuts begun in 1960s by the work of Ford and Fulkerson [10]. It has many generalizations and applications now. We are interested in a generalization of cuts related to the flows using only short paths.

Length bounded cuts Let $s, t \in V$ be two distinct vertices of a graph $G = (V, E)$ – we call them source and sink, respectively. We call a subset of edges $F \subseteq E$ of G an L-bounded cut (or L-cut for short), if the length of the shortest path between s and t in the graph $(V, E \setminus F)$ is at least $L + 1$. We measure the length of the path by the number of its edges. In particular, we do not require s and t to be in distinct connected components as in the standard cut, instead we do not allow s and t to be close to each other. We call the set F *a minimum L-cut* if it has the minimum size among all L-bounded cuts of the graph G.

We state the cut problem formally:

[⋆] Research was supported by the project SVV-2014-260103.
[⋆⋆] Author supported by the project Kontakt LH12095, project GAUK 1784214 and project CE-ITI P202/12/G061

PROBLEM: Minimum Length Bounded Cut (MLBC)
Instance: graph $G = (V, E)$, vertices s, t and integer $L \in \mathbb{N}$
Goal: find a minimum L-bounded s, t cut $F \subset E$

Length bounded flows were first considered by Adámek and Koubek [1]. They showed that the max-flow min-cut duality cannot hold and also that integral capacities do not imply integral flow. Finding a minimum length bounded cut is NP-hard on general graphs for $L \geq 4$ as was shown by Itai et al. [14]. They also found algorithms for finding L-bounded cut with $L = 1, 2, 3$ in polynomial time by reducing it to the usual network cut in an altered graph. The algorithm of Itai et al. [14] uses the fact that paths of length $1, 2$ and 3 are edge disjoint from longer paths, while this does not hold for length at least 4.

Baier et al. [2] studied linear programming relaxation and approximation of MLBC together with inapproximability results for length bounded cuts. They also showed instances of the MLBC having $O(L)$ integrality gap for their linear programming approach, which are series-parallel graphs and thus have constant bounded tree-width. The first parametrized complexity study of this and similar topics was made by Golovach and Thilikos [11] who studied parametrization by paths-length (that is in our setting the parameter L) and the size of the solution for cuts. They also proved hardness results – finding disjoint paths in graphs of bounded tree-width is a W[1]-hard problem.

The MLBC problem has its applications in the network design and in the telecommunications. Huygens et al. [13] use a MLBC as a subroutine in the design of 2-edge-connected networks with cycles at most L long. The MLBC problem is called *hop constrained* in the telecommunications and the number L is so called number of hops. The main interest is in the constant number of hops, see for example the article of Dahl and Gouveia [5].

Note that the standard use of the Courcelle theorem [3] gives for each fixed L a linear time algorithm for the decision version of the problem. But there is no apparent way of changing these algorithms into a single linear time algorithm. Moreover there is a nontrivial dependency between the formula (and thus the parameter L) and the running time of the algorithm given by Courcelle theorem.

Now we give a formal definition of a rather new graph parameter, for which we give one of our results:

Definition 1 (Tree depth [16]). *The closure $Clos(F)$ of a forest F is the graph obtained from F by making every vertex adjacent to all of its ancestors. The tree-depth $\mathrm{td}(G)$ of a graph G is one more than the minimum height of a rooted forest F such that $G \subseteq Clos(F)$.*

Our Contribution Our main contribution is an algorithm for the MLBC problem, its consequences and an algorithm for a more general multi-terminal version problem.

Theorem 1. *Let G be a graph of tree-width k. Let s and t be two distinct vertices of G. Then for any $L \in \mathbb{N}$ an minimum L-cut between s and t can be found in time $O((L^{k^2})^2 \cdot 2^{k^2} \cdot n)$.*

Corollary 1. *Let G be a graph, $k = \mathrm{td}(G)$ and s and t be two distinct vertices of G. Then for any $L \in \mathbb{N}$ an minimum L-cut between s and t can be found in time $f(k)n$, where f is a computable function.*

Proof As k is the tree depth of G it follows that the length of any path in G can be upper-bounded by $2k$. It is a folklore fact, that k is also a upper-bound on the tree-width of G. So we can use the Theorem 1 for $L < 2k$ and any other polynomial algorithm for minimum cut problem otherwise. $\square$

Corollary 2. *Let $G = (V, E)$ be a graph of tree-width k, $s \neq t \in V$ and $L \in \mathbb{N}$. There exists a computable function $f : \mathbb{N} \to \mathbb{N}$, such that a minimum L-cut between s and t can be found in time $O(n^{f(k)})$, where $n = |V|$.*

Theorem 2. Minimal Length Bounded Cut *parametrized by path-width is* W[1]*-hard.*

Tree-width versus tree-depth Admitting an FPT algorithm for a problem when parameterized by the tree-width implies an FPT algorithm for the problem when parameterized by the tree-depth, as parameter-theoretic observation easily shows. On the other hand, the FPT algorithm parameterized by the tree-width usually uses exponential (in the tree-width) space, while the tree-depth version uses only polynomial space (in the tree-depth).

From this point of view, it seems to be interesting to find problems that are "on the edge between path-width and tree-depth". That is problems that admit an FPT algorithm when parameterized by the tree-depth, but being W[1]-hard when parameterized by the path-width.

The only other result of this type, we a re aware of in the time of writing this article is by Gutin et al. [12]. The Minimum Length Bounded Cut problem is also a problem of this kind—as Theorems 2 and 1 demonstrate.

Theorem 1 gives us that the MLBC problem is fixed parameter tractable (FPT) when parametrized by the length of paths and the tree-width and that it belongs to XP when parametrized by the tree-width only (and thus solvable in polynomial time for graph classes with constant bounded tree-width).

Theorem 3. *There is no polynomial kernel for the* Minimum Length Bounded Cut *problem parameterized by the tree-width of the graph and the length L, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

We want to mention that our techniques apply also for more general version of the MLBC problem.

Length-bounded multi-cut We consider a generalized problem, where instead of only two terminals, we are given a set of terminals. For every pair of terminals, we are given a constraint—a lower bound on the length of the shortest path between these terminals. More formally:

Let $S = \{s_1, \ldots, s_k\} \subset V$ be a subset of vertices of the graph $G = (V, E)$ and let $a : S \times S \to \mathbb{N}$ be a mapping. We call a subset of edges $F \subseteq E$ of G

an **a**-*bounded multi-cut* if length of the shortest path between s_i and s_j in the graph $(V, E \setminus F)$ is at least $a(s_i, s_j)$ long for every $i \neq j$. Again if F has smallest possible size, we call it *minimum* **a**-*bounded* $\{s_1, \ldots, s_k\}$-*multi-cut*. We call the vertices $s_1, \ldots, s_k$ *terminals*. Finally, as there are only finitely many values of the mapping a we write $a_{s,t}$ instead of $a(s,t)$, we also write **a** instead of function a. Let $L \geq \max_{s,t \in S} a(s,t)$, we say that the problem is *L-limited*.

PROBLEM:	MINIMUM LENGTH BOUNDED MULTI-CUT (MLBMC)
Instance:	graph $G = (V, E)$, set $S \subset V$ and $a_{s,t} \in \mathbb{N}$ for all $s, t \in S$, satisfying the triangle inequalities
Goal:	find a minimum length bounded S multi-cut $F \subset E$

Theorem 4. *Let $G = (V, E)$ be a graph of tree-width k, $S \subseteq V$ with $|S| = t$ and let $p := t + k$. Then for any $L \in \mathbb{N}$ and any L-limited length-constraints* **a** *on S an minimum* **a**-*bounded multi-cut can be computed in time $O((L^p)^2 \cdot 2^{p^2} \cdot n)$.*

2 Preliminaries

In this section we recall some standard definitions from the graph theory and state what a tree decomposition is. After this we introduce changes of the tree decomposition specific for our algorithm. We proceed by the notion of auxiliary graphs used in proofs of our algorithm correctness. Finally, in Section 2.1 we summarize the results allowing us to prove that it is unlikely for a parameterized problem to admit a polynomial kernelization procedure.

We use the notion of tree decomposition of the graph:

Definition 2. *A* tree decomposition *of a graph $G = (V, E)$ is a pair $\mathcal{T} = (\{B_i : i \in I\}, T = (I, F))$, where T is a rooted tree and $\{B_i : i \in I\}$ is a family of subsets of V, such that*

1. *for each $v \in V$ there exists an $i \in I$ such that $v \in B_i$,*
2. *for each $e \in E$ there exists an $i \in I$ such that $e \subseteq B_i$,*
3. *for each $v \in V, I_v = \{i \in I : v \in B_i\}$ induces a subtree of T.*

We call the elements B_i the nodes, and the elements of the set F the decomposition edges.

We define a width of a tree decomposition $\mathcal{T} = (\{B_i : i \in I\}, T)$ as $\max_{i \in I} |B_i| - 1$ and the *tree-width* $\mathrm{tw}(G)$ of a graph G as the minimum width of a tree decomposition of the graph G. Moreover, if the decomposition is a path we speak about the *path-width* of G, which we denote as $\mathrm{pw}(G)$.

Nice tree decomposition [15] For algorithmic purposes it is common to define a *nice tree decomposition* of the graph. We naturally orient the decomposition edges towards the root and for an oriented decomposition edge (B_j, B_i) from B_j to B_i we call B_i the *parent* of B_j and B_j a *child* of B_i. If there is an oriented path from B_j to B_i we say that B_j is a *descendant* of B_i.

We also adjust a tree decomposition such that for each decomposition edge (B_i, B_j) it holds that $||B_i| - |B_j|| \leq 1$ (i.e. it joins nodes that differ in at most one vertex). The in-degree of each node is at most 2 and if the in-degree of the node B_k is 2 then for its children B_i, B_j holds that $B_i = B_j = B_k$ (i.e. they represent the same vertex set).

We classify the nodes of a nice decomposition into four classes—namely *introduce nodes*, *forget nodes*, *join nodes* and *leaf nodes*. We call the node B_i an introduce node of the vertex v, if it has a single child B_j and $B_i \setminus B_j = \{v\}$. We call the node B_i a forget node of the vertex v, if it has a single child B_j and $B_j \setminus B_i = \{v\}$. If the node B_k has two children, we call it a join node (of nodes B_i and B_j). Finally we call a node B_i a leaf node, if it has no child.

Proposition 1. *[15] Given a tree decomposition of a graph G with n vertices that has width k and $O(n)$ nodes, we can find a nice tree decomposition of G that also has width k and $O(n)$ nodes in time $O(n)$.*

So far we have described a standard nice tree decomposition. Now we change the introduce nodes. Let B_j be an introduce node and B_i its parent. We add another two copies B_j^p, B_j^s of B_j to the decomposition. We remove decomposition edge (B_j, B_i) and add three decomposition edges $(B_j, B_J^p), (B_j^s, B_j^p)$ and (B_j^p, B_i). Note that after this operation, B_j^s is a leaf of the decomposition, B_j remains an introduce node and B_j^p is a join node. We call B_j^s a sibling of B_j. Note that by these further modifications we preserve linear number of nodes in the decomposition.

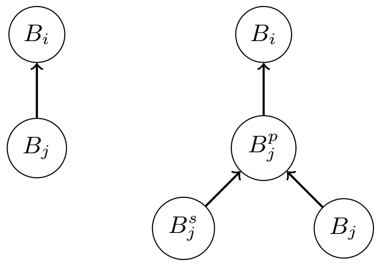

Fig. 1. Change of Introduction nodes in the nice tree decomposition

Auxiliary subgraphs Recall that for each edge there is at least one node containing that particular edge. Note that after our modification of the decomposition for each edge e there is at least one leaf B_i of the decomposition satisfying $e \subseteq B_i$. To see this, suppose this is not true and that some edge, say e, must be placed into a non-leaf node B_j. We may suppose that B_j is an introduce node (for join or forget node choose its descendant). However, in our construction any introduce node B_j has a sibling B_j^s such that is a leaf in the decomposition tree and their bags are equal.

Thus, for every edge $e \in E(G)$ we choose an arbitrary leaf node B_i such that $e \in B_i$ and say that the edge e *belongs* to the leaf node B_i. By this process we have chosen set $E_i \subset E(G)$ for each leaf node B_i. We further use the notion of *auxiliary graph* G_{B_i}, or G_i for short. For a leaf node B_i we set a graph $G_i = (V_i = B_i, E_i)$. For a non-leaf node B_i we set a graph $G_i = (V, E)$, where $V = B_i \cup \bigcup_{B_j \text{ child of } B_i} V_j$ and $E = \bigcup_{B_j \text{ child of } B_i} E_j$.

2.1 Preliminaries on refuting polynomial kernels

Here we present simplified review of a framework used to refute existence of polynomial kernel for a parameterized problem from Chapter 15 of a monograph by Cygan et al. [4].

In the following we denote by Σ a final alphabet, by Σ^* we denote the set of all words over Σ and by $\Sigma^{\leq n}$ we denote the set of all words over Σ and length at most n.

Definition 3 (Polynomial equivalence relation). *An equivalence relation $\mathcal{R}$ on the set Σ^* is called* polynomial equivalence relation *if the following conditions are satisfied:*

1. *There exists an algorithm such that, given strings $x, y \in \Sigma^*$, resolves whether $x \equiv_{\mathcal{R}} y$ in time polynomial in $|x| + |y|$.*
2. *Relation $\mathcal{R}$ restricted to the set $\Sigma^{\leq n}$ has at most $p(n)$ equivalence classes for some polynomial $p(\cdot)$.*

Definition 4 (Cross-composition). *Let $L \subseteq \Sigma^*$ be an unparameterized language and $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parametrized language. We say that L* cross-composes *into Q if there exists a polynomial equivalence relation $\mathcal{R}$ and an algorithm $\mathcal{A}$, called the cross-composition, satisfying the following conditions. The algorithm $\mathcal{A}$ takes on input a sequence of strings $x_1, x_2, \ldots, x_t \in \Sigma^*$ that are equivalent with respect to $\mathcal{R}$, runs in polynomial time in $\sum_{i=1}^{t} |x_i|$, and outputs one instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that:*

1. *$k \leq p(max_{i=1}^{t}|x_i|, \log t)$ for some polynomial $p(\cdot, \cdot)$, and*
2. *$(y, k) \in Q$ if and only if $x_i \in L$ for all i.*

With this framework, it is possible to refute even stronger reduction techniques—namely polynomial compression—according to the following definition:

Definition 5 (Polynomial compression). *A* polynomial compression *of a parameterized language $Q \subseteq \Sigma^* \times \mathbb{N}$ into an unparameterized language $R \subseteq \Sigma^*$ is an algorithm that takes as an input an instance $(y, k) \in \Sigma^* \times \mathbb{N}$, works in polynomial time in $|x| + k$, and returns a string y such that:*

1. *$|y| \leq p(k)$ for some polynomial $p(\cdot)$, and*
2. *$y \in R$ if and only if $(x, k) \in Q$.*

It is possible to refute existence of polynomial kernel using Definitions 3,4 and 5 with the help of use of the following theorems and a complexity assumption that is unlikely to hold—namely $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Theorem 5 ([7]). *Let $L, R \subseteq \Sigma^*$ be two languages. Assume that there exists an AND-distillation of L into R. Then $L \in \mathsf{coNP/poly}$.*

Theorem 6. *Assume that an NP-hard language L AND-cross-composes to a parameterized language Q. Then Q does not admit a polynomial compression, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

3 Minimal Length Bounded Multi-cuts

In this section we give a more detailed study of the length constraints for the length-bounded multi-cut and the triangle inequalities. From this we derive Lemma 1 for merging solutions for edge-disjoint graphs.

The triangle inequalities Note that the solution for MLBMC problem has to satisfy the triangle inequalities with respect to its instance. This means that for any three terminals $s, t, u \in S$ and the distance function *dist* it holds that $dist(s, u) + dist(u, t) \geq dist(s, t) \geq a_{s,t}$. Thus, we can restrict instances of MLBMC problem only to those satisfying these triangle inequalities.

Definition 6 (Length constraints). *Let $G = (V, E)$ be a graph, $S \subset V$ and let $k = |S|$. We call a vector $\mathbf{a} = (a_{s_1,s_2}, \ldots, a_{s_{k-1},s_k})$ a length constraint if for every $s, t, u \in S$ it holds that $a_{s,u} + a_{u,t} \geq a_{s,t}$.*

For our approach it is important to see the structure of the solution on a graph composed from two edge disjoint graphs.

Lemma 1. *Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ be edge disjoint graphs. Then for the graph $G = G_1 \cup G_2$ and $S = V_1 \cap V_2$ and an arbitrary length constraints $\mathbf{a} \in \mathbb{N}^{\binom{|S|}{2}}$ it holds that the minimum length bounded $(S, \mathbf{a})$ multi-cut F for G is a disjoint union of the $(S, \mathbf{a})$ multi-cuts F_1 and F_2 for G_1 and G_2.*

Proof First we prove that there cannot be smaller solution than $F_1 \cup F_2$. To see this observe that for every $(S, \mathbf{a})$ cut F' on G it holds that $F' \cap E_1$ is an $(S, \mathbf{a})$ cut on G_1 (and vice versa for G_2). Hence if F' would be a cut of smaller size then F, we would get a contradiction with the minimality of choice of F_1 and F_2, because we would have $|F'| < |F| = |F_1| + |F_2|$.

Now we prove that $F_1 \cup F_2$ is a valid solution. To see this we prove that every path between two terminals is not short. We prove that there cannot be such P by an induction on number of $h := |V(P) \cap S|$. If $h = 2$ then because G_1 and G_2 are edge disjoint, we may (by symmetry) assume that $P \subset G_1$, a contradiction with the choice of F_1. If $h > 2$ then there is a vertex $u \in S \setminus \{s, t\}$ such that the path P is composed from two segments P_1 and P_2, where P_1 is a path between s and u and P_2 is a path between u and t. And so we have $|P| = |P_1| + |P_2| \geq a_{s,u} + a_{u,t} \geq a_{s,t}$, what was to be demonstrated. □

4 Restricted bounded multi-cut

In this section we present our approach to the L-bounded cut for the graphs of bounded tree-width. We present our algorithm together with some remarks on our results.

Recall that we use dynamic programming techniques on a tree decomposition of an input graph. First we want to root the decomposition in a node containing both source and sink of the L-cut problem. This can be achieved by adding the source to all nodes on the unique path in the decomposition tree between any node containing the source and any node containing the sink. Note that this may add at most 1 to the width of the decomposition.

Length vectors and tables As it was mentioned in the previous section, we solve the L-cut by reducing it to simple instances of generalized MLBMC problem. We begin with a mapping $a : S \times S \to \mathbb{N}$ with meaning $a(s,t) = l_{s,t}$. For simplicity we represent the mapping a by a vector, calling it a *length vector* $\mathbf{a}$ and relax it for a node X $\mathbf{a} = (a_{x_1,x_2}, \ldots, a_{x_{k-1},x_k}) \in \mathbb{N}^{\binom{k}{2}}$, where $k = |X|$ and $X = \{x_1, \ldots, x_k\}$. We reduce the problem to the $\mathbf{a}$-bounded multi-cut for k terminals, where $k = tw(G)+2$ (the additional two is for changing the decomposition). Let us introduce a relation on length vectors $\mathbf{a}, \mathbf{b} \in \mathbb{N}^{\binom{k}{2}}$ on X of the same size. We write $\mathbf{a} \preceq \mathbf{b}$, if $a_{x_i,x_j} \leq b_{x_i,x_j}$ for all $1 \leq i < j \leq |X|$.

Let the set of vertices $X = \{x_1, \ldots, x_k\}$, $\mathbf{a}$ be a length vector, let $I \subset [k]$ and let $Y = \{x_i \in X : i \in I\}$. By $\mathbf{a}|_Y$ we denote the length vector $\mathbf{a}$ containing a_{x_i,x_j} if and only if both $i \in I$ and $j \in I$ (in an appropriate order) – in this case we say $\mathbf{a}|_Y$ is $\mathbf{a}$ *contracted* on the set Y.

Recall that for each node X we have defined the auxiliary graph G_X (see Section 2 for the definition). With a node X we associate the table Tab_X. The table entry for constraints $\mathbf{a} = (a_{1,2}, \ldots, a_{k-1,k})$ of Tab_X (denoted by $Tab_X[\mathbf{a}]$) for the node $X = \{x_1, \ldots, x_k\}$ contains the size of the $\mathbf{a}$-bounded multi-cut for the set X in the graph G_X. Note that for two length vectors $\mathbf{a} \preceq \mathbf{b}$ it holds that $Tab_X[\mathbf{a}] \leq Tab_X[\mathbf{b}]$.

4.1 Node Lemmas

The leaf nodes are the only nodes bearing some edges. We use an exhaustive search procedure for building tables for these nodes. For this we need to compute the lengths of the shortest paths between all the vertices of the leaf node, for which we use the well known procedure due to Floyd and Warshall [8,17]:

Proposition 2 ([8,17]). *Let G be a graph with nonnegative length $f : G(E) \to \mathbb{N}$. It is possible to compute the table of lengths of the shortest paths between any pair $u, v \in V(G)$ with respect to f in time $O(|V(G)|^3)$.*

Lemma 2 (Leaf Nodes). *For all L-limited length vectors and a leaf node X the table Tab_X of sizes of minimum length-bounded multi-cuts can be computed in time $O(L^{k^2} \cdot 2^{k^2} \cdot k^3)$, where $k = |X|$.*

Proof Fix one L-limited length vector $\mathbf{a}$. We run the Floyd-Warshall algorithm (stated as Proposition 2) for every possible subgraph of G_X. As $|E(G_X)| \leq \binom{k}{2}$ there are $O(2^{k^2})$ such subgraphs. This gives us a running time $O(2^{k^2} \cdot k^3)$ for a single length vector $\mathbf{a}$ – we check all $O(k^2)$ length constraints if each of them is satisfied and set

$$Tab_X[\mathbf{a}] := \min_{F \subseteq E(G_X):\, F \text{ satisfies } \mathbf{a}} (|E(G_X) \setminus F|).$$

Finally there are $O(L^{k^2})$ L-bounded length vectors, this gives our result. □

We now use Lemma 1 to prove time complexity of finding a dynamic programming table for join nodes from the table of its children.

Lemma 3 (Join Nodes). *Let X be a join node with children Y and Z, let L be the limit on length vectors components and let $k = |X|$. Then the table Tab_X can be computed in time $O(L^{k^2})$ from the table Tab_Y and Tab_Z.*

Proof Recall that graphs G_Y and G_Z are edge disjoint and that we store sizes of $\mathbf{a}$-bounded multi-cuts. Note also that $X = V(G_Y) \cap V(G_Z)$ and so we can apply Lemma 1 and set $Tab_X[\mathbf{a}] := Tab_Y[\mathbf{a}] + Tab_Z[\mathbf{a}]$, for each $\mathbf{a}$ satisfying the triangle inequalities.

As there are $O(L^{k^2})$ entries in the table Tab_X we have the complexity we wanted to prove. □

As the forget node expects only forgetting a vertex and thus forgetting part of the table of the child. This is the optimizing part of our algorithm.

Lemma 4 (Forget Nodes). *Let X be a forget node, Y its child, let L be the limit on length vectors components and let $k = |X|$. Then the table Tab_X can be computed in time $O((L^{k^2})^2)$ from the table Tab_Y.*

Proof Fix one length vector $\mathbf{a}$ and compute the set $\mathcal{A}(\mathbf{a})$ of all Y-augmented length vectors. Formally $\mathbf{b} \in \mathcal{A}(\mathbf{a})$ if $\mathbf{b}$ is a length vector satisfying the triangle inequalities for Y and $\mathbf{b}|_X = \mathbf{a}$. After this we set

$$Tab_X[\mathbf{a}] := \min_{\mathbf{b} \in \mathcal{A}(\mathbf{a})} Tab_Y[\mathbf{b}].$$

There are at most L^{k^2} of Y-augmented length vectors for each $\mathbf{a}$ and this gives the claimed time. □

Also the introduce node (as the counter part for the forget node) only adds coordinates to the table of its child. It does no computation as there are no edges it can decide about – these nodes now only add isolated vertex in the graph.

Lemma 5 (Introduce Nodes). *Let X be an introduce node, Y its child, let L be the limit on length vectors components and let $k = |X|$. Then the table Tab_X can be computed in time $O(L^k)$ from the table Tab_Y.*

Proof Let x be the vertex with the property $x \in X \setminus Y$. The key property is that the vertex x is an isolated vertex in G_X and thus we can set $Tab_X[\mathbf{a}] := Tab_Y[\mathbf{a}|_Y]$, because x is arbitrarily far from any vertex in G_Y, especially from the set Y. □

4.2 Proofs of Theorems

We use Lemmas 2, 3, 4 and 5 to prove the theorem about computing L-bounded (s,t)-cut in graph of bounded tree-width. For this, note that we can use $k = O(tw(G))$ and put it into all the Lemmas as it is an upper bound on the size of any node in the decomposition.

We compute all the L-bounded length-constraint that satisfy the triangle inequalities in advance. This takes additional time $O(L^{k^2} \cdot k^3)$ which can be upper-bounded by $O((L^{k^2})^2 \cdot 2^k)$ for $k \geq 2$ and $L \geq 2$ and so this does not make the overall time complexity worse.

Proof of Theorem 1 As there are $O(n)$ nodes in nice tree decomposition (by Proposition 1) and as we can upper-bound time needed to compute any type of node by $O((L^{k^2})^2 \cdot 2^{k^2})$, we have complexity proposed in state of the Theorem 1. □

Let us now point out that the value of the parameter L can be upper-bounded by the number of vertices n of the input graph G (in fact by $n^{1-\varepsilon}$ as it is proved in [2]).

We now want to sum-up the key ideas leading to Theorem 1. First, it is the use of dynamic program for computing all options of cuts for bounded number of possible choices and for second it is the idea of creating a node that includes both the source and the sink while not harming the tree-width too much. On the other hand, we can use this idea to solve also the generalized version of the problem – length-bounded multi-cut – with the additional parameter the number of terminals. It is easy to see that in this setting that again it is possible to achieve node containing every terminal and thus this yields following Theorem 7.

Theorem 7. *Let $G = (V, E)$ be a graph of tree-width k, $S \subseteq V$ with $|S| = t$ and let $p := t + k$. Then for any $L \in \mathbb{N}$ and any L-limited length-constraints $\mathbf{b}$ satisfying the triangle inequalities on S an minimum $\mathbf{b}$-bounded multi-cut can be computed in time $O((L^p)^2 \cdot 2^{p^2} \cdot n)$.*

Proof To see that Theorem 7 implies Theorem 4 note, that given any L-limited length-constraints $\mathbf{a}$ the minimum $\mathbf{a}$-bounded multi-cut must satisfy triangle inequalities. Thus, we find the minimum $\mathbf{b}$-bounded multi-cut among all $\mathbf{b} \succeq \mathbf{a}$. Note that we can do this in additional time $O(L^{p^2})$, but this does not make the total running time worse. □

5 Hardness of the *L*-bounded cut

In this section we prove MLBC parametrized by path-width is W[1]-hard [9] by FPT-reduction from k-Multicolor Clique.

PROBLEM: k-Multicolor Clique

Instance:	k-partite graph $G = (V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k, E)$, where V_i is independent set for every i and they are pairwise disjoint
Parameter:	k
Goal:	find a clique of the size k

Denoting In this section, sets $V_1, \ldots, V_k$ are always partites of the k-partite graph G. We denote edges between V_i and V_j by E_{ij}. The problem is W[1]-hard even if every independent set V_i has the same size and the number of edges between every V_i and V_j is the same. In whole Section 5 we denote the size of an arbitrary V_i by N and the size of an arbitrary E_{ij} by M. For an FPT-reduction from k-Multicolor Clique to MLBC we need:

1. Create an MLBC instance $G' = (V', E'), s, t, L$ from the k-Multicolor Clique instance $G = (V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k, E)$ where the size of G' is polynomial in the size of G.
2. Prove that G contains a k-clique if and only if G' contains an L-bounded cut of the size $f(k, N, M)$ where f is a polynomial.
3. Prove the path-width of H is smaller than $g(k)$ where g is a computable function.

Our ideas were inspired by work of Michael Dom et al. [6]. They proved W[1] hardness of Capacitated Vertex Cover and Capacitated Dominating Set parametrized by the tree width of the input graph. We remarked that their reduction also proves W[1] hardness of these problems parametrized by path-width.

5.1 Basic gadget

In the k-Multicolor Clique problem we need to select exactly one vertex from each independent set V_i and exactly one edge from each E_{ij}. Moreover, we have to make certain that if $e \in E_{ij}$ is the selected edge and $u \in V_i, v \in V_j$ are the selected vertices then $e = \{u, v\}$. The idea of the reduction is to have a basic gadget for every vertex and edge. We connect gadgets g_v for every v in V_i into a path P_i. The path P_i is cut in the gadget g_v if and only if the vertex $v \in V_i$ is selected into the clique. The same idea will be used for selecting the edges.

Definition 7. *Let $h, Q \in \mathbb{N}$. Butte $B(s', t', h, Q)$ is a graph which contains h paths of length 2 and Q paths of length $h + 2$ between the vertices s' and t'. The short paths (of length 2) are called shortcuts, the long paths are called ridgeways and the parameter h is called height.*

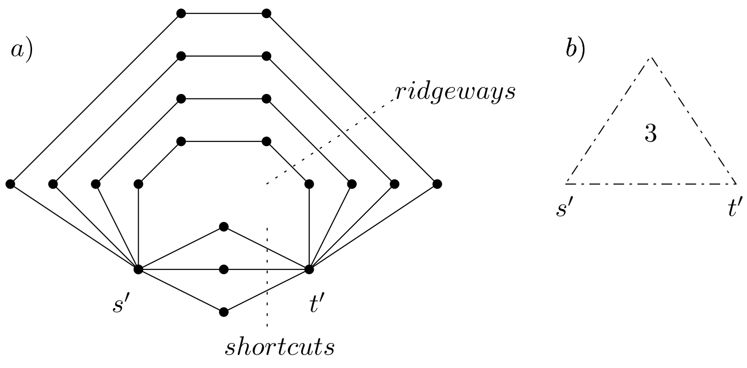

Fig. 2. a) Example of a butte for $h = 3$ and $Q = 4$. b) Simply diagram for a butte of height 3.

A butte for $h = 3, Q = 4$ is shown in Figure 2 part a. In our reduction all buttes will have the same parameter Q (it will be computed later). For simplicity we depict buttes as a dash dotted line triangles with its height h inside (see Figure 2 part b), or only as triangles without the height if it is not important.

Let $B(s', t', h, Q)$ be a butte. We denote by $s(B), t(B), h(B), Q(B)$ the parameters of butte B s', t', h and Q, respectively. We state easy but important observation about the butte path-width:

Observation 8 *Path-width of an arbitrary butte B is at most 3.*

Proof If we remove vertices $s(B)$ and $t(B)$ from B we get $Q(B)$ paths from ridgeways and $h(B)$ isolated vertices from shortcuts. This graph has certainly path-width 1. If we add $s(B)$ and $t(B)$ to every node of the path decomposition we get a proper path decomposition of B with width 3. □

Let butte $B(s', t', h, Q)$ be a subgraph of a graph G. Let u, v be vertices of G and all paths between u and v goes through B such that they enter into B in s' and leave it in t' (see Figure 3). The important properties of the butte B are:

1. By removing one edge from all h shortcuts of B, we extend the the distance between u and v by h. If we cut all shortcuts of butte B we say the butte B is ridged.
2. Let size of the cut is bounded by $K \in \mathbb{N}$ and we can remove edges only from B. If we increase Q to be bigger than K then any path between u and v cannot be cut by removing edges from B (only extended by ridging the butte B).

5.2 Butte path

In this section we define how we connect buttes into a path, which we call highland. The main idea is to have highland for every pair $(i, j), i \neq j \in [k]$. In

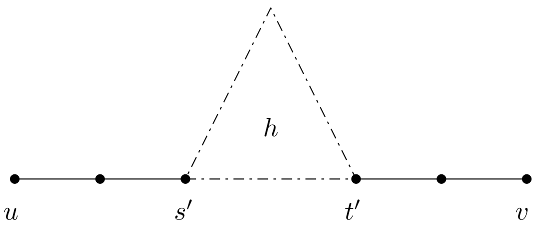

Fig. 3. Example of a path going through a butte.

the highland for (i, j), there are buttes for every vertex $v \in V_i$ and every edge $e \in E_{i,j}$. We connect vertex buttes and edge buttes into a path. Then we set the butte heights and limit the size of the cut in such way that:

1. Exactly one vertex butte and exactly one edge butte have to be ridged.
2. If a butte for a vertex v is ridged, then only buttes for edges incident with v can be ridged.

Formal description of the highland is in the following definition.

Definition 8. *The highland $H(X, Y, s, t)$ is a graph containing 2 vertices s and t and $Z = X + Y$ buttes $B_1, \ldots, B_Z$ where:*

1. *$s = s(B_1), t = t(B_Z)$ and $t(B_i) = s(B_{i+1})$ for every $1 \leq i < Z$.*
2. *$h(B_i) = X^2 + i$ for $1 \leq i \leq X$.*
3. *$h(B_i) \in \{X^4, \ldots, X^4 + X - 1\}$ for $X + 1 \leq i \leq Z$.*
4. *$Q(B_i) = X^4 + X^2$ for every i.*

Let $H(X, Y, s, t)$ be a highland. We call buttes $B_1, \ldots, B_X$ from H low and buttes $B_{X+1}, \ldots, B_{X+Y}$ high (low buttes will be used for the vertices and high buttes for the edges). The vertex $t(B_X) = s(B_{X+1})$, where low and high buttes meet, is called the center of highland H. Note that there can be more buttes with the same height among high buttes and they are not ordered by height as the low buttes. An example of a highland is shown in Figure 4.

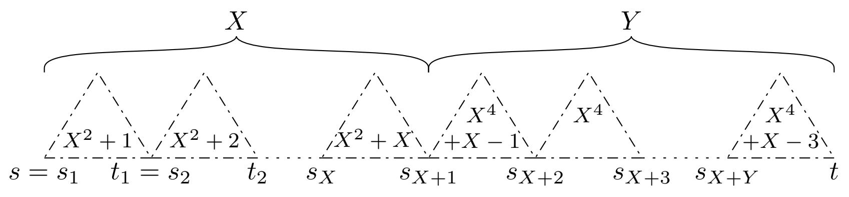

Fig. 4. Example of a highland $H(X, Y, s, t)$.

Proposition 3. *Let $H(X, Y, s, t)$ be a highland. Let $L = 2(X + Y) + X^4 + X^2 + X - 1$. Let C be the L-cut of size $X^4 + X^2 + X$, which cut all paths of length L and shorter between s and t then:*

1. *The cut C contains only edges obtained by ridging the exactly two buttes B_i, B_j, such that B_i is low and B_j is high.*
2. *Let B_i be the ridged low butte and B_j be the ridged high butte. Then, $h(B_j) = X^4 + X - i$.*

Proof of Proposition 3 Every butte has at least $X + 1$ shortcuts and $X^4 + X^2$ ridgeways. Therefore, C can not cut all paths in H between s and t and it is useless to add edges from ridgeways to the cut C. Note that the shortest st-path in H has the length $2(X + Y)$.

1. If we ridge every low butte we extend the shortest st-path by $X^3 + \frac{X^2}{2} + \frac{X}{2}$. However, it is not enough and at least one high butte has to be ridged. Two high buttes cannot be ridged otherwise the cut would be bigger then the bound. No high butte can extend the shortest st-path enough, therefore at least one low butte has to be ridged. However, two low buttes and one high butte cannot be ridged because the cut C would be bigger then the bound.
2. Let F be the set of removed edges from ridged buttes B_i and B_j. The height of B_i is $X^2 + i$. Therefore, the length of the shortest st-path after ridging B_i and B_j and the size of F is $2(X+Y)+X^2+i+h(B_j)$. If $h(B_j) < X^4+X-i$ then shortest st-path is strictly shorter then $2(X+Y)+X^4+X^2+X$. Thus, F is not L-cut. If $h(B_j) > X^4 + X - i$ then and $|F| > X^4 + X^2 + X$ thus F is bigger than C.

□

5.3 Reduction

In this section we present our reduction. Let $G = (V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k, E)$ be the input for k-Multicolor Clique. As we stated in the last section, the main idea is to have a low butte B_v for every vertex $v \in V(G)$ and a high butte B_e for every edge $e \in E(G)$. Vertex v and edge e is selected into the k-clique if and only if the butte B_v and the butte B_e are ridged. From G we construct MLBC input G', s, t, L (the construction is quite technical, for better understanding see Figure 5):

1. For every $1 \leq i, j \leq k, i \neq j$ we create highland $H^{i,j}(N, M, s, t)$ of buttes $B^{i,j}_1, \ldots, B^{i,j}_{N+M}$.
2. Let $V_i = \{v_1, \ldots, v_N\}$. The vertex $v_\ell \in V_i$ is represented by the low butte $B^{i,j}_\ell$ of the highland $H^{i,j}$ for every $j \neq i$. Thus, we have $k - 1$ copies of buttes (in different highlands) for every vertex. Hence, we need to be certain that only buttes representing the same vertex are ridged. Note that buttes representing the same vertex have the same height and the same distance from the vertex s.

3. Let $E_{ij} = \{e_1, \dots, e_M\}, i < j$. Edge $e_\ell = \{u, v\} \in E_{ij} (u \in V_i, v \in V_j)$ is represented by the high butte $B^{i,j}_{N+\ell}$ of the highland $H^{i,j}$ and by the high butte $B^{j,i}_{N+\ell}$ of the highland $H^{j,i}$. Note that two buttes represented the same edge has same distance from the vertex s. Let h_i, h_j be the heights of buttes representing the vertices u and v, respectively. We set the buttes heights:
   (a) $h(B^{i,j}_{N+\ell}) = N^4 + N - h_i$
   (b) $h(B^{j,i}_{N+\ell}) = N^4 + N - h_j$
4. We add edge $\{t(B^{i,j}_\ell), t(B^{i,j+1}_\ell)\}$ for every $1 \le i \le k, 1 \le j < k, i \ne j$ and $1 \le \ell < N$.
5. We add paths of length $N - 1$ connected $t(B^{i,j}_\ell)$ and $t(B^{j,i}_\ell)$ for every $1 \le i, j \le k, i \ne j$ and $1 \le \ell < M$.
6. We set L to $2(N + M) + N^4 + N^2 + N - 1$.

We call paths between highlands in Items 4 and 5 the valley paths.

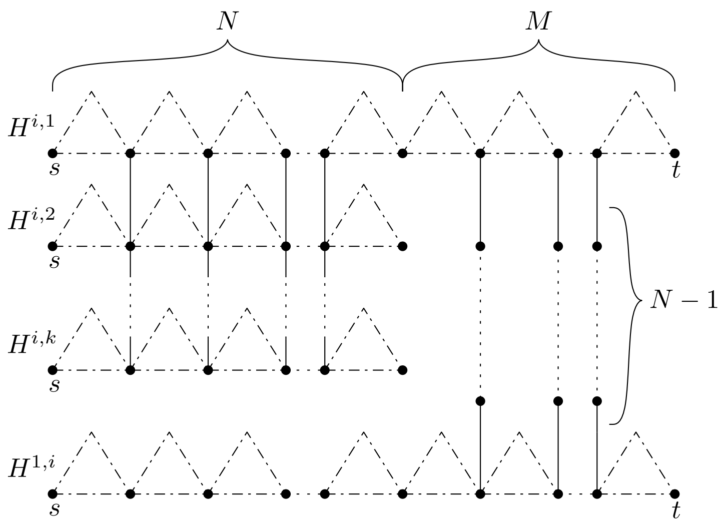

Fig. 5. Some part of the graph G'. All vertices labeled s and t are actually two vertices s and t in the graph G'. We divided them for better illustration. Highlands $H^{i,2}$ and $H^{i,k}$ have also high buttes, but we omitted them.

Observation 9 *Graph G' has a polynomial size in the size of the graph G.*

Theorem 10. *If graph G has a clique of size k then (G', s, t) has an L-cut of size $k(k-1)(N^4 + N^2 + N)$.*

Proof Let G has a k-clique $\{v_1, \dots, v_k\}$ where $v_i \in V_i$ for every i and $e_{ij} = \{v_i, v_j\} \in E_{ij}$. For every i we ridge all $k - 1$ buttes representing the vertex v_i in G'. And for every $i < j$ we ridge both buttes representing the edge e_{ij}.

We claim that set of removed edges from ridged buttes forms the L-cut. Let $H^{i,j}$ be an arbitrary highland. There is no st-path shorter than L in $H^{i,j}$. Let $h(B_v) = N^2 + \ell$ where B_v is arbitrary butte representing the vertex v_i. By construction of G', the high butte representing the edge e_{ij} in $H^{i,j}$ has height $N^4 + N - \ell$. Thus, ridged buttes in $H^{i,j}$ extend the shortest st-path by $N^4 + N^2 + N$ and it has length $2(M+N) + N^4 + N^2 + N$. Buttes representing the vertex v_i have same height. Thus, a path through the low buttes of highlands using some valley path is always longer than path going through low buttes of only one highland. Therefore, it is useless to use valley paths among low buttes for the shortest st-path.

Other situation is among high buttes because buttes representing the same edge have different heights. The butte B_v representing the vertex v_i extend the shortest path at least by $N^2 + 1$. The butte B_e representing the edge $e_{i,j}$ extend the shortest at least by N^4. However, if $h(B_v) + h(B_e) < N^4 + N^2 + N$ then B_v and B_e have to be in different highlands. Therefore, the st-path going through B_v and B_e has to use a valley path between high buttes, which has length $N - 1$. Hence, any st-path has the length at least $2(N + M) + N^4 + N^2 + N$.

We remove $N^4 + N^2 + N$ edges from each highland and there are $k(k-1)$ highlands in G'. Therefore, G' has L-cut of the size $k(k-1)(N^4 + N^2 + N)$. □

Theorem 11. *If (G', s, t) has an L-cut of size $k(k-1)(N^4 + N^2 + N)$ then G has a clique of size k.*

Proof Let C be an L-cut of G'. Every shortest st-path going through every highland has to be extended by $N^4 + N^2 + N$. By Proposition 3 (Item 1), exactly one low butte and exactly one high butte of each highland has to be ridged. We remove $(N^4 + N^2 + N)$ from every highland in G'. Therefore, there can be only edges from ridged buttes in C.

For fixed i, highlands $H^{i,j}$ are the highlands which low buttes represent vertices from V_i. We claim that ridged low buttes of $H^{i,1}, \ldots, H^{i,k}$ represent the same vertex. Suppose for contradiction, there exists two low ridged buttes B_ℓ of $H^{i,\ell}$ and B_m of $H^{i,m}$ which represent different vertex from V_i. Without loss of generality $H^{i,\ell}$ and $H^{i,m}$ are next to each other (i.e. $|\ell - m| = 1$) and the distance from s to $s(B_\ell)$ is smaller than the distance from s to $s(B_m)$. Let B'_ℓ be a butte of $H^{i,m}$ such that it has the same distance from s as the butte B_ℓ (see Figure 6). The path s–$t(B'_\ell)$–$t(B_\ell)$–t does not go through any ridged low butte. Therefore, this path is shorter than L, which is contradiction. We can use the same argument to show that there are not two high ridged buttes of highland $H^{i,j}$ and $H^{j,i}$ which represent different edges from E_{ij}.

We put into the k-clique $K \subset V(G)$ the vertex $v_i \in V_i$ if and only if an arbitrary butte representing the vertex v_i is ridged. We proved in the previous paragraph that exactly one vertex from V_i can be put into the clique K. Let $e_{ij} \in E_{ij}$ be an edge represented by ridged high buttes. We claim that $v_i \in e_{ij}$. Let $B \in H^{i,j}$ be a butte representing v_i with height $N^2 + \ell$. Then by Proposition 3 (Item 2), butte $B' \in H^{i,j}$ of height $N^4 + N - \ell$ has to be ridged. By construction of G', only buttes representing edges incident with v_i have such height. Therefore,

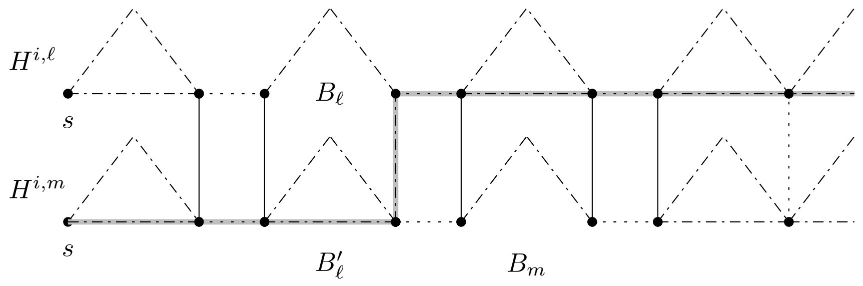

Fig. 6. How to miss every ridged low butte if there are ridged two low buttes representing two different vertices from one color class. Ridged butte is depicted as triangle without hypotenuse.

chosen edges are incident with chosen vertices and they form the k-clique of the graph G. □

Observation 12 *Graph G' has path-width in $O(k^2)$.*

Proof Let H be a graph created from G by replacing every butte by a single edge and contract the valley paths between high buttes into single edges, see Figure 7 transformation a. Let U be a vertex set containing s, t and every highland center. Let H' be a graph created from H by removing all vertex from U, see Figure 7 transformation b.

Graph H' is unconnected and it contains k grids of size $(k-1) \times (N-2)$ and $\binom{k}{2}$ grids of size $2 \times (M-2)$. Path-width of $(k-1) \times (N-2)$ grids is in $O(k)$, therefore $\text{pw}(H') \in O(k)$. If we add set U to every node of a path decomposition of H' we get proper path decomposition of H. Since $|U| \in O(k^2)$, path-width of H is in $O(k^2)$. The edge subdivision does not increase path-width. Moreover, replacing edges by buttes does not increase it either (up to multiplication constant) because butte has the constant path-width (Observation 8). Therefore, $\text{pw}(G) = c\,\text{pw}(H)$ for some constant c and $\text{pw}(G) \in O(k^2)$.

□

And thus Theorem 2 easily follows from Observations 9 and 12 and Theorems 10 and 11.

6 Polynomial kernel is questionable

In this section, we prove that for the Minimum Length Bounded Cut problem it is unlikely to admit a polynomial kernel when parametrized by the length L and the path-width (tree-width) of the input graph. We will prove this fact by the use of a AND-composition framework—that is by designing an AND-composition algorithm from the unparameterized gapped version of the Minimum Length Bounded Cut problem into itself.

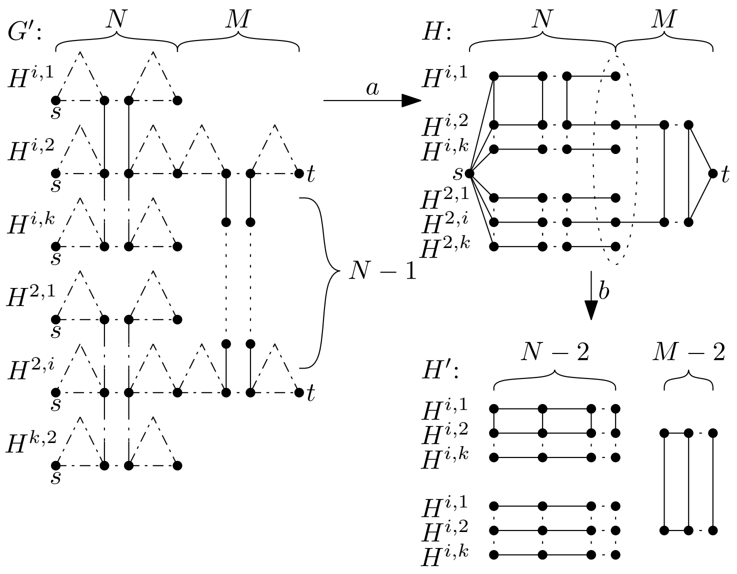

Fig. 7. The transformation a replace all buttes in G' by single edges and contract long valley paths into single edges. The transformation b removes vertices s and t and all highland centers (highlighted by dotted ellipsis) from H.

PROBLEM:	Decision version of Minimum Length Bounded Cut
Instance:	Graph $G = (V, E)$, vertices s_1, s_2, positive integers L, K
Question:	Is there an L length bounded cut consisting of exactly K edges

Baier et al. [2] proved that this problem is NP-complete even if we restrict the input instances such that

- the desired length $L \geq 4$ is constant,
- either there is an L bounded cut of size exactly K or every L bounded cut has at least $1.13K$ edges.

Thus, it is possible to define the polynomial relation $\mathcal{R}$ as follows. We will consider instances (G, s_1, s_2, L, K) of the decision problem with constant L. Two instances $(G, s_1, s_2, L, K), (G', s'_1, s'_2, L, K')$ are equivalent if $|V(G)| = |V(G')|$ and $K = K'$. It is clear that $\mathcal{R}$ is a polynomial equivalence relation.

AND-composition We will take graphs $G_1, G_2, \ldots, G_t$ the input instances that are equivalent under the relation $\mathcal{R}$ for the Gapped Minimum Length Bounded Cut problem with the constant $L = 4$. As a composition we will take the disjoint union of graphs $G_1, G_2, \ldots, G_t$ and unify sources and sinks of the resulting graph and denote the graph as G.

Now it is easy to see that the path-width of the graph G of our construction is at most $\max_{i=1,2,\ldots,t} |G_i| + 2$ as we can form bags of the path decomposition by whole graphs G_i add the source and the sink into every bag and connect them into a path. This is indeed a correct path decomposition as the only two vertices in common are in every bag.

As we have taken the parameter L to be constant, this finishes arguments about parameters. And thus also the proof of Theorem 3.

7 Conclusions

There is another standard generalization of the length bounded cut problem – where we add to each edge also its length. If this length is integral it is possible to extend and use our techniques (we only subdivide edges longer than 1 – this doesn't raise the tree-width of the graph on the input). On the other hand, if we allow fractional numbers, it is uncertain how to deal with such a generalization.

Acknowledgments Authors thank to Jiří Fiala, Petr Kolman and Lukáš Folwarczný for fruitful discussions about the problem. We would like to mention that part of this research was done during Summer REU 2014 at Rutgers University.

References

1. J. Adámek and V. Koubek, *Remarks on flows in network with short paths*, Commentationes mathematicae Universitatis Carolinae, 12 (1971), pp. 661 – 667.
2. G. Baier, T. Erlebach, A. Hall, E. Köhler, P. Kolman, O. Pangrác, H. Schilling, and M. Skutella, *Length-bounded cuts and flows*, ACM Trans. Algorithms, 7 (2010), pp. 4:1–4:27.
3. B. Courcelle, *Graph rewriting: An algebraic and logic approach*, Handbook of Theoretical Computer Science, (1990), pp. 194–242.
4. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, 2015.
5. G. Dahl and L. Gouveia, *On the directed hop-constrained shortest path problem*, Operations Research Letters, 32 (2004), pp. 15 – 22.
6. M. Dom, D. Lokshtanov, S. Saurabh, and Y. Villanger, *Capacitated domination and covering: A parameterized perspective*, in Parameterized and Exact Computation, M. Grohe and R. Niedermeier, eds., vol. 5018 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 78–90.
7. A. Drucker, *New limits to classical and quantum instance compression*, in 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012, 2012, pp. 609–618.
8. R. W. Floyd, *Algorithm 97: Shortest path*, Commun. ACM, 5 (1962), p. 345.
9. J. Flum and M. Grohe, *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
10. L. R. Ford and D. R. Fulkerson, *Maximal flow through a network*, Canadian journal of mathematics, 8 (1956), pp. 399–404.
11. P. A. Golovach and D. M. Thilikos, *Paths of bounded length and their cuts: Parameterized complexity and algorithms*, Discrete Optimization, 8 (2011), pp. 72 – 86.

12. G. Gutin, M. Jones, and M. Wahlström, *Structural parameterizations of the mixed chinese postman problem*, in Algorithms – ESA 2015, N. Bansal and I. Finocchi, eds., vol. 9294 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2015, pp. 668–679.
13. D. Huygens, M. Labbé, A. R. Mahjoub, and P. Pesneau, *The two-edge connected hop-constrained network design problem: Valid inequalities and branch-and-cut*, Networks, 49 (2007), pp. 116–133.
14. A. Itai, Y. Perl, and Y. Shiloach, *The complexity of finding maximum disjoint paths with length constraints*, Networks, 12 (1982), pp. 277–286.
15. T. Kloks, *Treewidth, Computations and Approximations (Lecture notes in computer science, 842)*, Springer-Verlag New York, Inc., 1994.
16. J. Nešetřil and P. O. de Mendez, *Tree-depth, subgraph coloring and homomorphism bounds*, European Journal of Combinatorics, 27 (2006), pp. 1022 – 1041.
17. S. Warshall, *A theorem on boolean matrices*, J. ACM, 9 (1962), pp. 11–12.