Parameterized Complexity of Length-bounded Cuts and Multicuts

Pavel Dvořák · Dušan Knop

Abstract We show that the Minimum Length-Bounded Cut 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 $L + 1$ long. We derive an FPT algorithm for a more general multi-commodity length-bounded cut problem when additionally parameterized by the number of terminals.

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 path-width and L.

We also derive an FPT algorithm for the Minimum 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 · polynomial kernel · tree-depth · tree-width

Research was supported by the project SVV-2016-260332.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 616787.

Author supported by the project Kontakt LH12095, project GAUK 1784214 and project CE-ITI P202/12/G061.

P. Dvořák · D. Knop
Faculty of Mathematics and Physics, Malostranské náměstí 25, 118 00 Praha 1
Tel.: +420-221914230
Fax: +420-257531014
E-mail: koblich@iuuk.mff.cuni.cz, knop@kam.mff.cuni.cz

1 Introduction

The study of network flows and cuts began in 1950s by the work of Ford and Fulkerson [11]. 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 the source and the 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. Throughout the paper we denote by n the number of vertices of input graph G and by m the number of edges of G.

We state the cut problem formally:

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. [15]. They also found algorithms for finding a minimum 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. [15] 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 MLBC. 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 parameterized complexity study of this and similar topics was made by Golovach and Thilikos [12] who studied parameterization 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. Very recently Fluschnik et al. [10] showed that, unless a collapse in the Polynomial Hierarchy occurs, there is no polynomial kernel with respect to parameters L and the size of the solution.

The MLBC problem has its applications in network design and in telecommunications. Huygens et al. [14] 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 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 [6].

Note that the standard use of Courcelle's theorem [4] 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's 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 [18]) 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 $L \in \mathbb{N}$ a minimum L-cut between s and t can be found in time $O\big(L^{6k^2} \cdot n\big)$.*

Corollary 1 *Let G be a graph, $k = \mathrm{td}(G)$ and s and t be two distinct vertices of G. Then for any $L \in \mathbb{N}$ a minimum L-cut between s and t can be found in time $O\big(\max\{2^{6k^3} \cdot n, nm\}\big)$.*

Proof As k is the tree-depth of G it follows that the length of any path in G can be upper-bounded by 2^k (this follows from Proposition 6.2 in Nešetřil, de Mendez book [17]). It is a folklore fact, that k is also an upper-bound on the tree-width of G. Thus, we can use Theorem 1 for $L < 2^k$. If $L \geq 2^k$, then L-cut is standard minimum cut in G. For this case we use Orlin's algorithm [19] for max flow/min cut problem with running time $O(nm)$. □

Corollary 2 *Let $G = (V, E)$ be a graph of tree-width k, $s \neq t \in V$ and $L \in \mathbb{N}$. A minimum L-cut between s and t can be found in time $O\big(n^{6k^2+1}\big)$.*

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

Path-width versus Tree-depth Admitting an FPT algorithm for a problem when parameterized by the path-width (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 path-width (tree-width) usually uses exponential (in the width) space, while the tree-depth version uses only polynomial space (in the tree-depth).

From this point of view, it is 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 are aware of, in the time of writing this article, is by Gutin et al. [13]. The Minimum Length-Bounded Cut problem is also a problem of this kind—as Theorem 2 and Corollary 1 demonstrate.

Theorem 1 gives us that the MLBC problem is fixed parameter tractable (FPT) when parameterized by the length of paths and the tree-width and that it belongs to XP when parameterized by the tree-width only (and is 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* NP $\subseteq$ coNP/poly.

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

Length-bounded Multicut 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 $S = \{s_1, \ldots, s_q\} \subseteq V$ be a subset of vertices of the graph $G = (V, E)$ and let $\mathbf{a} \in \mathbb{N}^{\binom{S}{2}}$ be a vector, where $\mathbb{N}^{\binom{S}{2}}$ is a set of all natural number vectors indexed by pairs of vertices in S. We call a subset of edges $F \subseteq E$ of G an $\mathbf{a}$-*bounded S-multicut* if the length of the shortest path between s_i and s_j in the graph $(V, E \setminus F)$ is at least $a_{s_i,s_j} + 1$ for every $s_i, s_j \in S, i < j$. Again if F has smallest possible size, we call it *minimum* $\mathbf{a}$-*bounded S-multicut*. We call the vertices $s_1, \ldots, s_q$ *terminals*. Let $L \geq \max_{s_i,s_j \in S\colon i<j} a_{s_i,s_j}$, we say that the problem is *L-limited*.

Problem:	Minimum Length-Bounded MultiCut (MLBMC)
Instance:	graph $G = (V, E)$, set $S \subset V$ and $\mathbf{a} \in \mathbb{N}^{\binom{S}{2}}$
Goal:	find a minimum $\mathbf{a}$-bounded S-multicut $F \subset E$

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

2 Preliminaries

In this section we recall some standard definitions from 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.

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

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

We call the elements of I the *nodes*, the elements of the set F the *decomposition edges* and a set B_X is a bag of node X.

We define a width of a tree decomposition $\mathcal{T} = (\{B_X : X \in I\}, T)$ as $\max_{X \in I} |B_X| - 1$ 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 [16] For algorithmic purposes it is common to define a *nice tree decomposition* of the graph. We rooted the decomposition and naturally orient the decomposition edges towards the root. For an oriented decomposition edge (X, Y) from X to Y we call Y the *parent* of X and X a *child* of Y. If there is an oriented path from X to Y we say that X is a *descendant* of Y.

We also adjust a tree decomposition such that for each decomposition edge (X, Y) it holds that B_X and B_Y differ in at most one vertex. The in-degree of each node is at most 2 and if the in-degree of the node Z is 2 then for its children X, Y holds that $B_X = B_Y = B_Z$ (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 X an introduce node of the vertex v, if it has a single child Y and $B_X \setminus B_Y = \{v\}$. We call the node X a forget node of the vertex v, if it has a single child Y and $B_Y \setminus B_X = \{v\}$. If the node Z has two children X and Y, we call it a join node (of the nodes X and Y). Finally we call a node X a leaf node, if it has no child.

Proposition 1 *[16] 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 $O(n)$.*

Grafted Tree Decomposition So far we have described a standard nice tree decomposition. Now, we will describe how to change a nice tree decomposition to a *grafted tree decomposition*. Let X be an introduce node and Y its child. We add another two nodes X^c and X^s such that $B_{X^c} = B_{X^s} = B_X$. We remove decomposition edge (Y, X) and add three decomposition edges

$(Y, X^c), (X^s, X)$ and (X^c, X) (see Figure 1). Note that after this operation, X^s is a leaf of the decomposition, X^c is an introduce node and X is a join node. Note that by these further modifications we preserve linear number of nodes in the decomposition.

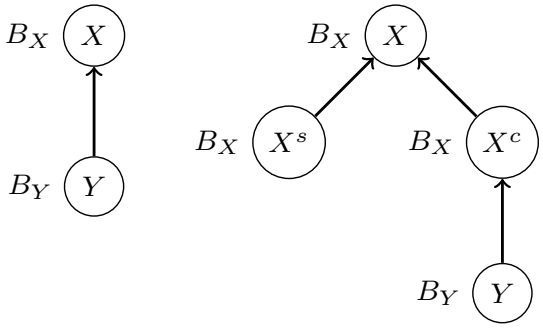

Fig. 1 Change of Introduction nodes in the grafted tree decomposition.

Note that in the grafted tree decomposition for each edge e there is at least one leaf X^s of the decomposition satisfying $e \subseteq B_{X^s}$. By the definition of tree decomposition, we know there is a node X^c such that $e \subseteq B_{X^c}$. If X^c is not a leaf node, then we may suppose that X^c is an introduce node (for join or forget node choose its descendant). However, in the grafted decomposition any introduce node X^c has a sibling X^s that is a leaf node and $B_{X^c} = B_{X^s}$.

Auxiliary Subgraphs For every edge $e \in E(G)$ we choose an arbitrary leaf node X such that $e \in B_X$ and say that the edge e *belongs* to the leaf node X. By this process we have chosen set $E_X \subset E(G)$ for each leaf node X. Note that the sets E_X for all leaves X of the decomposition forms a partition of the set $E(G)$. We further use the notion of *auxiliary graph* G_X. For a leaf node X we set a graph $G_X = (B_X, E_X)$. For a non-leaf node Y we set a graph $G_Y = (V, E)$, where

$$V = B_Y \cup \bigcup_{X \text{ child of } Y} V(G_X)$$
$$E = \bigcup_{X \text{ child of } Y} E(G_X).$$

Vector Notation We often use integer vectors whose entries are indexed by pairs of vertices. We use bold characters for vectors $(\mathbf{a}, \mathbf{b})$ and italic characters for entries ($a_{x,y}, b_{x,y}$ are entries of $\mathbf{a}$, $\mathbf{b}$ respectively, for a pair of vertices x, y). Let S be a set of vertices and $\mathbf{a}, \mathbf{b} \in \mathbb{N}^{\binom{S}{2}}$. We write $\mathbf{a} \preceq \mathbf{b}$, if $a_{x,y} \leq b_{x,y}$ for all $\{x, y\} \in \binom{S}{2}$.

2.1 Preliminaries on Refuting Polynomial Kernels

Here we present a 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. [5].

In the following we denote by Σ a finite 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 (AND-cross-composition) Let $L \subseteq \Sigma^*$ be an unparameterized language and $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized 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 AND-cross-composition, satisfying the following conditions. The algorithm $\mathcal{A}$ takes on input a sequence of strings $x_1, x_2, \ldots, x_r \in \Sigma^*$ that are equivalent with respect to $\mathcal{R}$, runs in polynomial time in $\sum_{i=1}^{r} |x_i|$, and outputs one instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that:

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

We say that language L has a *polynomial kernel* if there is a kernelization algorithm that takes on input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, runs in polynomial time in $|x|$ and k, and outputs an equivalent instance $(x', k') \in \Sigma^* \times \mathbb{N}$ with $|x'| \leq p(k')$ and $k' \leq q(k)$, where $p(\cdot), q(\cdot)$ are polynomials. With this framework, it is possible to refute even stronger data reduction techniques—namely polynomial compression:

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 input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, 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 easy to see that the polynomial kernelization is a special case of the polynomial compression. It is possible to refute existence of polynomial compression (and polynomial kernel) using AND-cross-composition with the help of use of the following theorem and a complexity assumption that is unlikely to hold—namely $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Theorem 5 ([5,3]) *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 FPT Algorithm for the L-bounded Cut

In this section we present our approach to the L-bounded cut for the graphs of bounded tree-width. First, we give a more detailed study of the length constraints for the length-bounded multicut and the triangle inequalities. From this we derive Lemma 1 for merging solutions for edge-disjoint graphs. Second, we describe how to use dynamic programming in different nodes of the tree decomposition, which gives us the final algorithm.

Triangle Inequalities Let $\mathcal{I} = \big(G = (V, E), S, \mathbf{a}\big)$ be an instance of MLBMC and $F \subseteq E(G)$ be a solution of $\mathcal{I}$. Note that the distances between terminals in $G' = (V, E \setminus F)$ satisfy the triangle inequalities. This means that for any three terminals $s, t, u \in S$ and the distance function $\mathrm{dist}\colon V \times V \to \mathbb{N}$ in G' it holds that $\mathrm{dist}(s, u) + \mathrm{dist}(u, t) \geq \mathrm{dist}(s, t) \geq a_{s,t} + 1$. We say that a vector $\mathbf{a} \in \mathbb{N}^{\binom{S}{2}}$ *satisfies sharp triangle inequalities* if $a_{s,u} + a_{u,t} + 1 \geq a_{s,t}$ (i.e. $a_{s,u} + a_{u,t} > a_{s,t}$) for all distinct terminals $s, t, u \in S$. Thus, it makes sense to restrict instances of MLBMC problem only to those satisfying sharp triangle inequalities. We will formalize this idea in Observation 1.

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 it satisfies sharp triangle inequalities.

Observation 1 *Let $\mathcal{I} = \big(G = (V, E), S, \mathbf{a}\big)$ be an instance of MLBMC and $F \subseteq E$ be a solution of $\mathcal{I}$. Now, there exists a length constraint $\mathbf{b} \in \mathbb{N}^{\binom{S}{2}}$ such that $\mathbf{b} \succeq \mathbf{a}$ and F is a solution of $(G, S, \mathbf{b})$.*

Proof Let $\mathrm{dist}\colon V \times V \to \mathbb{N}$ be a distance function in graph $(V, E \setminus F)$. We define $b_{s,t}$ as $\mathrm{dist}(s, t) - 1$ for all distinct terminals $s, t \in S$; thus F is $\mathbf{b}$-bounded S-multicut. It is straightforward to check the vector $\mathbf{b}$ is a length constraint because the function dist satisfies triangle inequality. Now, $\mathbf{b} \succeq \mathbf{a}$, since $a_{s,t} < \mathrm{dist}(s, t)$ for all distinct terminals $s, t \in S$. Since $\mathbf{b} \succeq \mathbf{a}$, the instance $(G, S, \mathbf{b})$ does not admit a solution of size smaller than $|F|$. Therefore, the set F is a solution of $(G, S, \mathbf{b})$. □

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 constraint $\mathbf{a} \in \mathbb{N}^{\binom{S}{2}}$ it holds that a minimum length $\mathbf{a}$-bounded S-multicut F for G is a disjoint union of the minimum length $\mathbf{a}$-bounded S-multicuts 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 $\mathbf{a}$-bounded S-multicut F' on G it holds that $F' \cap E_1$ is an $\mathbf{a}$-bounded S-multicut on G_1 (and vice versa for G_2). Hence, if F' would be a cut of smaller size than 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 $F = F_1 \cup F_2$ is a valid solution. To see this we prove that every path between two terminals is not shorter than L. Let P be a path in $(V(G), E(G) \setminus F)$ between terminals $s, t \in S$. We prove that the length of P is at least $a_{s,t} + 1$ by an induction over a number $h := |V(P) \cap S|$. If $h = 2$ then because G_1 and G_2 are edge disjoint, we may (by symmetry) assume that $P \subset G_1$. Therefore, the length of P is at least $a_{s,t} + 1$ because F_1 is a valid solution.

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. Thus, by induction hypothesis and sharp triangle inequalities, we have $|P| = |P_1| + |P_2| \geq a_{s,u} + 1 + a_{u,t} + 1 \geq a_{s,t} + 1$, what was to be demonstrated. □

We use dynamic programming techniques on a grafted tree decomposition $\mathcal{T}$ of an input graph. First, we want to root the decomposition $\mathcal{T}$ 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.

We solve the L-cut by reducing it to simple instances of generalized MLBMC problem.

We reduce the problem to the $\mathbf{a}$-bounded S-multicut for k terminals, where $k = tw(G) + 1$ (the additional one is for changing the decomposition).

Let $X = \{x_1, \ldots, x_k\}$ be a set of vertices, $\mathbf{a}$ be a length constraint, let $I \subset [k]$ and let $Y = \{x_i \in X : i \in I\}$. By $\mathbf{a}|_Y$ we denote the length constraint $\mathbf{a}$ 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.

Dynamic programming tables Recall that for each node X of a tree decomposition 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 length constraints $\mathbf{a} = (a_{x_1,x_2}, \ldots, a_{x_{k-1},x_k})$ of Tab_X (denoted by $Tab_X[\mathbf{a}]$) for the node $X = \{x_1, \ldots, x_k\}$ contains the size of a minimum $\mathbf{a}$-bounded X-multicut for the set X in the graph G_X. Note that for two length constraints $\mathbf{a} \preceq \mathbf{b}$ it holds that $Tab_X[\mathbf{a}] \leq Tab_X[\mathbf{b}]$.

3.1 Node lemmata

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 com-

pute 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 [9,20]:

Proposition 2 ([9,20]) *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 constraints and a leaf node X the table Tab_X of sizes of minimum length-bounded multicuts can be computed in time $O\big(L^{k^2} \cdot 2^{k^2} \cdot k^3\big)$, where $k = |X|$.*

Proof Fix one L-limited length constraint $\mathbf{a}$. Let $G_X = (V, E)$ and $F \subseteq E$. We run the Floyd-Warshall algorithm (stated as Proposition 2) in graph $G_X^F = (V, E \setminus F)$. We check all $O(k^2)$ pairs of terminals if their distance is sufficiently large, i.e. if F is an $\mathbf{a}$-bounded S-multicut.

We iterate over all subsets of E and pick the minimum cut. As $|E| \leq \binom{k}{2}$ there are $O\big(2^{k^2}\big)$ choices for F. This gives us a running time $O\big(2^{k^2} \cdot k^3\big)$ for a single length constraint $\mathbf{a}$. We set the entry for $\mathbf{a}$ in Tab_X as

$$Tab_X[\mathbf{a}] := \min_{F \subseteq E\colon F \text{ is a } \mathbf{a}\text{-bounded } S\text{-multicut}} |F|.$$

Finally there are $O\big(L^{k^2}\big)$ L-limited length constraints, 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 constraints and let $k = |X|$. Then the table Tab_X can be computed in time $O\big(L^{k^2}\big)$ 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 multicuts. Note also that $B_X = V(G_Y) \cap V(G_Z)$ 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 $O\big(L^{k^2}\big)$ entries in the table Tab_X we have the complexity we wanted to prove. □

As the forget node represents forgetting a vertex, its table can be calculated by forgetting part of the table of the child node.

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

Proof Fix one length constraint $\mathbf{a}$ and compute the set $\mathcal{A}(\mathbf{a})$ of all B_Y-augmented length constraints. Formally, $\mathbf{b} \in \mathcal{A}(\mathbf{a})$ if $\mathbf{b}$ is a length constraint for B_Y and $\mathbf{b}|_{B_X} = \mathbf{a}$. After this we set

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

In the worst case for every entry in Tab_X we search whole Tab_Y, which gives us 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 vertices to the auxiliary graph.

Lemma 5 (Introduce Nodes) *Let X be an introduce node, Y its child, let L be the limit on length constraints and let $k = |X|$. Then the table Tab_X can be computed in time $O(L^{k^2})$ from the table Tab_Y.*

Proof Let v be the introduced vertex, i.e. $v \in B_X \setminus B_Y$. Let $\mathcal{T}'$ be a subtree of the grafted tree decomposition $\mathcal{T}$ rooted in X. Recall that each edge of the auxiliary graph G_X belongs to some leaf node of $\mathcal{T}'$. Since v is introduced in the root of $\mathcal{T}'$, there is no leaf node of $\mathcal{T}'$ containing v. Thus, there is no edge incident to v in G_X. Therefore, we can set $Tab_X[\mathbf{a}] := Tab_Y[\mathbf{a}|_{B_Y}]$, because v is arbitrarily far from any vertex in G_Y, especially from the set B_Y. □

3.2 Proofs of Theorems

We use Lemma 2, 3, 4 and 5 to prove Theorem 4.

Theorem 6 *Let $G = (V, E)$ be a graph of tree-width k, $S \subseteq V$ with $|S| = q$ and let $p := q + k - 1$. Then for any $L \in \mathbb{N}$ and any L-limited length constraint $\mathbf{b} \in [L]^{\binom{S}{2}}$ a minimum $\mathbf{b}$-bounded S-multicut can be computed in time $O(L^{3p^2} \cdot n)$.*

Proof First, we compute all L-limited length-constraints in advance to work with constraints instead of vectors that do not have to fulfill triangle inequalities. This takes additional time $O(L^{k^2} \cdot k^3)$ which can be upper-bounded by $O(L^{2k^2})$ for $L \geq 2$, which we can suppose because the problem for $L = 1$ can be trivially computed in linear time.

We create grafted tree decomposition $\mathcal{T}$ in linear time (see Section 2). We need that all terminals would be in the root of $\mathcal{T}$. Let S' be a set S without one vertex. We add the set S' to every bag of the decomposition $\mathcal{T}$—this increases the width of $\mathcal{T}$ by at most $|S'| = q - 1$. Thus, there exists a node R of $\mathcal{T}$ such that $S \subseteq B_R$. We rooted the decomposition $\mathcal{T}$ in R.

We compute Tab_X for all nodes X of the decomposition $\mathcal{T}$ using Lemma 2, 3, 4 and 5. Now, we are able to read the value of the solution from Tab_R. Since $L \geq 2$ and size of every bag in $\mathcal{T}$ is at most $p = q + k$, we can upper-bound the processing time for any type of node by $O(L^{3p^2})$. There are $O(n)$ nodes in $\mathcal{T}$ which gives us the claimed time. □

Theorem 1 and Theorem 4 are corollaries of Theorem 6.

Proof (Proof of Theorem 1) Since for two terminals the notions of length vector and length constraint are the same, we can use Theorem 6 to prove Theorem 1. For $\mathrm{tw}(G) = 1$ (G is a tree) can be the problem computed trivially in linear time. Otherwise for $\mathrm{tw}(G) = k >= 2$, we have running time $O(L^{3(k+1)^2} \cdot n)$ which can be upper-bounded by $O(L^{6k^2} \cdot n)$ as claimed. □

Proof (Proof of Theorem 4) Let $(G, S, \mathbf{a})$ be an input and $p = \mathrm{tw}(G) + |S|$. By Observation 1 we know that there exists a length constraints $\mathbf{b}$ such that instances $(G, S, \mathbf{a})$ and $(G, S\mathbf{b})$ has the same solution. Therefore, it suffice to try all length constraints $\mathbf{c} \succeq \mathbf{a}$ and select the smallest computed cut. By Theorem 6, the processing for one length constraint is $O\big(L^{3p^2} \cdot n\big)$. There are at most $L^{|S|^2}$ length constraints which gives us the claimed time $O\big(L^{4p^2} \cdot n\big)$. □

4 Hardness of the *L*-bounded Cut

In this section we prove that Decision version of Length-Bounded Cut parameterized by path-width is W[1]-hard by FPT-reduction from k-Multicolor Clique.

Problem:	Decision version of Length-Bounded Cut (DLBC)
Instance:	Graph $G = (V, E)$, vertices s, t, positive integers L, K
Question:	Is there an L-bounded cut of size at most K?

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 $i \in [k]$ and they are pairwise disjoint
Parameter:	k
Goal:	find a clique of the size k

Notation 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 [5] 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. Throughout this section 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 DLBC parameterized by path-width we need the following steps:

1. Create an DLBC instance $G' = (V', E'), s, t, L, K$ from the k-Multicolor Clique instance $G = (V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k, E)$ in time $f(k)|G|^{O(1)}$ for a computable function f.
2. Prove that G contains a k-clique if and only if G' contains an L-bounded cut of the size K.
3. Prove the path-width of G' is smaller than $g(k)$ where g is a computable function.

Our ideas were inspired by work of Michael Dom et al. [7]. They proved W[1] hardness of Capacitated Vertex Cover and Capacitated Dominating Set parameterized by the tree-width of the input graph. We remark that their reduction also proves W[1] hardness of these problems parameterized by path-width.

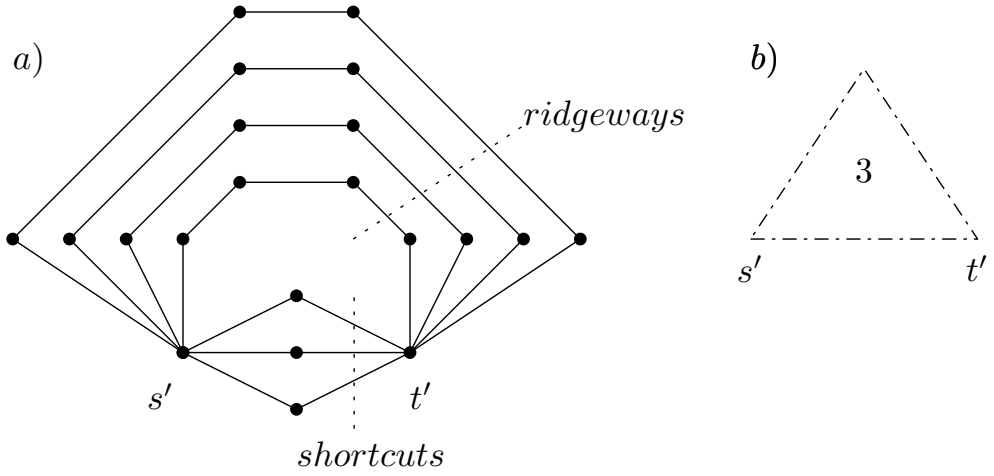

Fig. 2 a) Example of a butte for $h = 3$ and $Q = 4$. b) Simple diagram for a butte of height 3.

4.1 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 a selected edge and $u \in V_i, v \in V_j$ are 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.

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 their 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 an easy but important observation about the butte path-width:

Observation 2 *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 certainly has 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 such that all paths between u and v going through B enter into B in s' and leave it in t' (see Figure 3). The important properties of the butte B are:

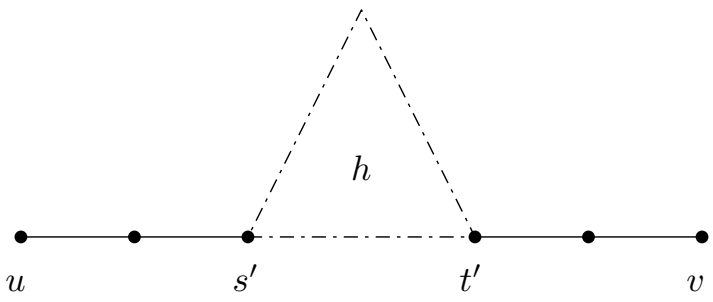

Fig. 3 Example of a path going through a butte.

1. By removing one edge from all h shortcuts of B, we extend the distance between u and v by h. If a cut C contains one edge of every shortcut of butte B we say the cut C *ridges* the butte B.
2. Suppose the size of a cut C is bounded by $K \in \mathbb{N}$ and C contains only edges in B. If we increase Q to be bigger than K then C cannot separate u and v (if C ridges B, then distance between u and v is only extended).

4.2 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 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 a 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.

The formal description of a highland is in the following definition.

Definition 8 Let $X, Y \in \mathbb{N}$. A 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.

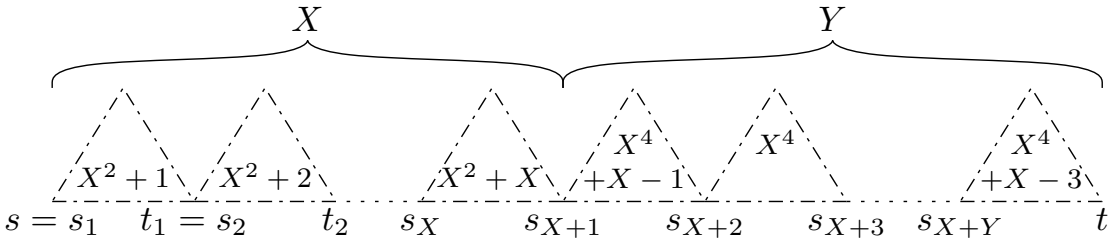

Fig. 4 Example of a highland $H(X, Y, s, t)$.

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 an L-cut of size $X^4 + X^2 + X$, which cuts all paths of length L and shorter between s and t then:*

1. *The cut C ridges 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 Every butte has at least $X^2 + 1$ 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 the cut C ridges every low butte then the shortest st-path is extended by $\sum_{i=1}^{X}(X^2 + i) = X^3 + \frac{X^2}{2} + \frac{X}{2}$. 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 than 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 than the bound.
2. The height of ridged low butte B_i is $X^2 + i$. Therefore, the length of the shortest st-path when the edges in C are removed is $2(X + Y) + X^2 + i + h(B_j)$ and the size of C is $X^2 + i + h(B_j)$. 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, C is not an L-cut. If $h(B_j) > X^4 + X - i$ then $|C| > X^4 + X^2 + X$ which is bigger than the bound.

□

4.3 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):

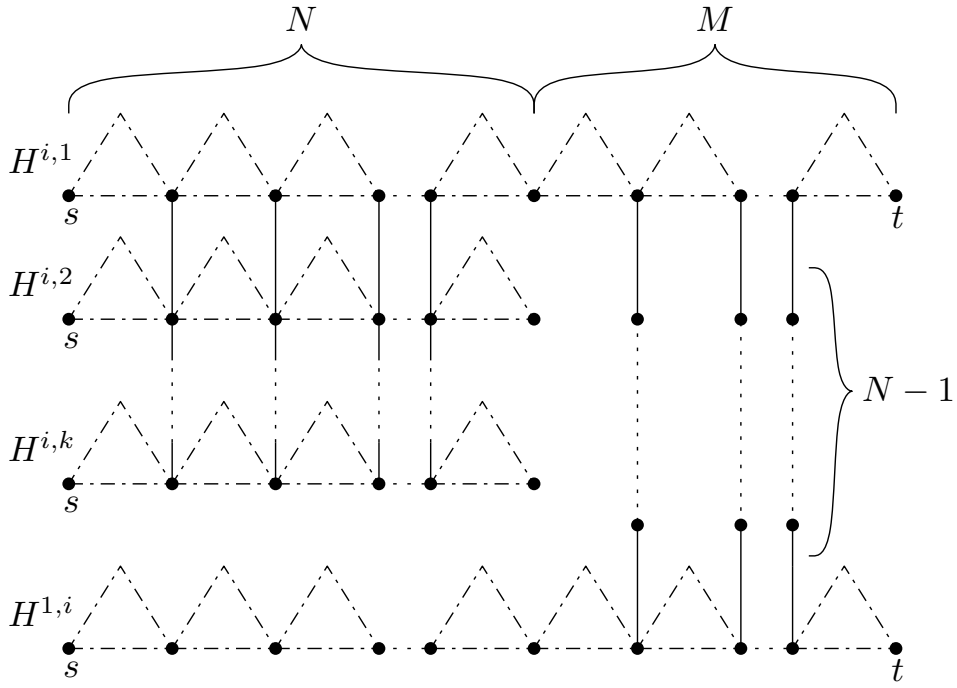

Fig. 5 Some part of the graph G'. All vertices labeled s and t are actually two vertices s and t in the graph G'. We divided them for better illustration. Highlands $H^{i,2}$ and $H^{i,k}$ have also high buttes, but we omitted them.

1. For every $1 \leq i, j \leq k, i \neq j$ we create highland $H^{i,j}(N, M, s, t)$ of buttes $B_1^{i,j}, \ldots, B_{N+M}^{i,j}$.
2. Let $V_i = \{v_1, \ldots, v_N\}$. The vertex $v_\ell \in V_i$ is represented by the low butte $B_\ell^{i,j}$ 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 $E_{ij} = \{e_1, \ldots, e_M\}, i < j$. Edge $e_\ell = \{u, v\} \in E_{ij} (u \in V_i, v \in V_j)$ is represented by the high butte $B_{N+\ell}^{i,j}$ of the highland $H^{i,j}$ and by the high butte $B_{N+\ell}^{j,i}$ of the highland $H^{j,i}$. Note that two buttes representing the same edge have 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) $h(B_{N+\ell}^{i,j}) = N^4 + N - h_i$
   (b) $h(B_{N+\ell}^{j,i}) = N^4 + N - h_j$
4. We add edge $\left\{t(B_\ell^{i,j}), t(B_\ell^{i,j+1})\right\}$ for every $1 \leq i \leq k, 1 \leq j < k, i \neq j$ and $1 \leq \ell < N$.
5. We add paths of length $N-1$ connecting $t(B_\ell^{i,j})$ and $t(B_\ell^{j,i})$ for every $1 \leq i, j \leq k, i \neq j$ and $N+1 \leq \ell < N+M$.
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*.

Observation 3 *Graph G' can be computed in polynomial time in the size of graph G.*

Theorem 7 *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 Suppose G has a k-clique $\{v_1, \ldots, v_k\}$ where $v_i \in V_i$ for every i and $e_{ij} = \{v_i, v_j\} \in E_{ij}$. We create an L-cut C. For every i the cut C ridges all $k-1$ buttes representing the vertex v_i in G'. And for every $i < j$ the cut C ridges both buttes representing the edge e_{ij}.

We claim that the set C is an L-cut. Let $H^{i,j}$ be an arbitrary highland. We show there is no st-path of length at most L in $H^{i,j}$. Let $h(B_v) = N^2 + \ell$ where B_v is an 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 the same height. Thus, a path through the low buttes of highlands using some valley path is always longer than a 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.

The remaining paths to consider are those using valley paths among high buttes, because buttes representing the same edge have different heights. The butte B_v representing the vertex v_i extends the shortest path at least by $N^2 + 1$. The butte B_e representing the edge $e_{i,j}$ extends 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 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 8 *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. By Proposition 3 (Item 2) 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 whose 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 exist two low ridged buttes B_ℓ of $H^{i,\ell}$ and B_m of $H^{i,m}$ which represent different vertices from V_i. It follows that there have to be two highlands $H^{i,p}, H^{i,p+1}$ such that their ridged low buttes represent different vertices. Thus, we can suppose that $H^{i,\ell}$ 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

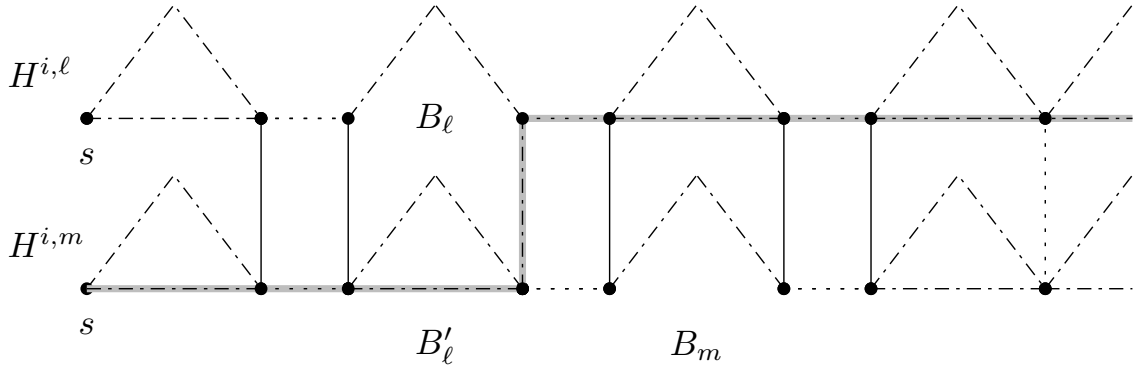

Fig. 6 How to miss every ridged low butte if there are ridged two low buttes representing two different vertices from one color class. Ridged butte is depicted as triangle without hypotenuse.

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

Observation 4 *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 vertices from U, see Figure 7 transformation b.

Graph H' is unconnected and it contains k grids of size $k \times (N-2)$ 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 $\mathrm{pw}(H') \in O(k)$. 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 2). Therefore, $\mathrm{pw}(G) = c\,\mathrm{pw}(H)$ for some constant c and $\mathrm{pw}(G) \in O(k^2)$.

□

Thus Theorem 2 easily follows from Observations 3 and 4 and Theorems 7 and 8.

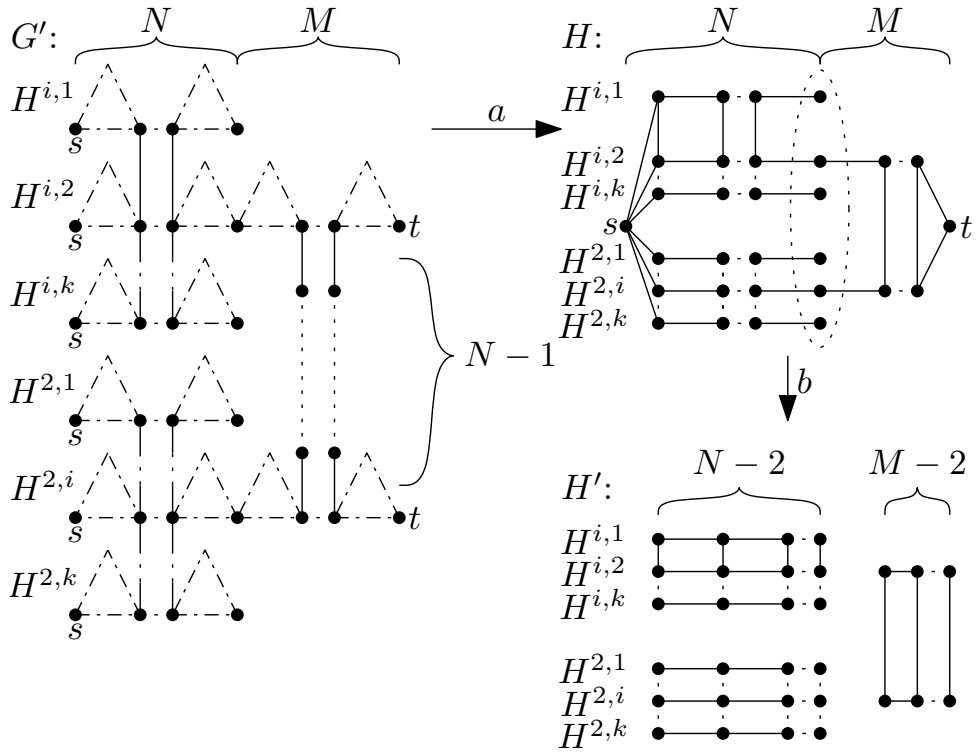

Fig. 7 The transformation a replaces all buttes in G' by single edges and contract long valley paths into single edges. The transformation b removes vertices s and t and all highland centers (highlighted by dotted ellipse) from H.

5 Polynomial kernel is questionable

In this section, we prove that the Minimum Length-Bounded Cut problem is unlikely to admit a polynomial kernel when parameterized by the length L and the path-width (tree-width) of the input graph. We will prove this fact by the use of an AND-cross-composition framework—that is by designing an AND-cross-composition algorithm from the Multicolor Clique problem:

Problem:	Multicolor Clique
Instance:	k-partite graph G and positive integer k
Question:	Has the graph G a clique of size k?

It is unparameterized version of k-Multicolor Clique from the previous section. The problem is NP-hard by well known reduction from the Clique problem [5]. Note that in the input graph G all cliques have size at most k. I.e., an instance (G, k) is an *yes*-instance if and only if the largest clique of G has the largest possible size k.

We define the polynomial equivalence relation $\mathcal{R}$ as follows. Two instances of the Multicolor Clique problem $(G, k), (G', k')$ are equivalent if $|V(G)| = |V(G')|, |E(G)| = |E(G')|$ and $k = k'$. It is clear that $\mathcal{R}$ is a polynomial equivalence relation.

AND-cross-composition We take the instances $(G_1, k), (G_2, k), \ldots, (G_r, k)$ that are equivalent under the relation $\mathcal{R}$. To every instance (G_i, k) we ap-

ply our reduction described in the previous section and get an instance $(G'_i, s_i, t_i, L_i, K_i)$ of DLBC problem. Let $n = |V(G_i)|, m = |E(G_i)|$. By the reduction,

$$L_i = 2(n+m) + n^4 + n^2 + n - 1$$
$$K_i = k(k-1)(n^4 + n^2 + n).$$

Since all instances of Multicolor Clique are equivalent under $\mathcal{R}$, for all $i, j \leq r$ holds that $L_i = L_j$ and $K_i = K_j$. Thus, all instances of DLBC has a form (G'_i, s_i, t_i, L, K). We take the disjoint union of graphs $G'_1, G'_2, \ldots, G'_r$ and unify all sources s_i to a vertex s and sinks t_i to a vertex t of the resulting graph and denote the graph as G. As we build the AND-cross-composition we set the budget of the resulting instance to $r \cdot K$, i.e., we have an instance (G, s, t, L, rK) of DLCB problem.

By the reduction, the minimum L-bounded cut in every graph G'_i has size at least K. Therefore, the graph G has the minimum L-bounded cut of size at least rK. If all instances (G_i, k) of Multicolor Clique are *yes*-instances, then there is an L-bounded cut in the graph G of size rK. If there is an *no*-instance (G_j, k) of Multicolor Clique, then every L-bounded cut in G'_j has size at least $K + 1$. Thus, every L-bounded cut in the graph G has size at least $rK + 1$.

So far we created an instance $\mathcal{I}$ of MLBC from r instances $(G_1, k), \ldots, (G_r, k)$ of Multicolor Clique and the instance $\mathcal{I}$ is *yes*-instance if and only if all instances (G_i, k) are *yes* instances. It remains to bound the parameters path-width of G and L. It is discussed above that L is polynomial in size of G_i. It is easy to see that the path-width of the graph G of our construction is at most

$$\max_{i=1,2,\ldots,r} |V(G'_i)| - 1.$$

We can put each graph G'_i into a bag B_i and connect them into a path. Only two common vertices among bags are the vertices s and t, which arise by unifying vertices s_i, t_i respectively. Thus, s and t are in all bags and we described the correct path decomposition such that $|B_i| = |V(G'_i)|$.

Thus, we have AND-cross-composition from NP-hard problem to DLBC parameterized by path-width and L. By Theorem 5, we can refute existence of polynomial kernel for DLBC parameterized by path-width and L and this finishes the proof of Theorem 3.

Acknowledgements We would like to thank our colleagues Jiří Fiala, Petr Kolman and Lukáš Folwarczný for fruitful discussions. Furthermore, our great thanks belongs to anonymous referees for the work they have done and the suggested improvements that led to this improved version of the paper.

The results presented in this paper are an extension of those presented at the conference TAMC 2015 [8].

Finally, we would like to mention that part of this research was done during international REU 2014, when both authors were visiting DIMACS (Rutgers University).

References

1. Adámek, J., Koubek, V.: Remarks on flows in network with short paths. Commentationes mathematicae Universitatis Carolinae **12**(4), 661 – 667 (1971)
2. Baier, G., Erlebach, T., Hall, A., Köhler, E., Kolman, P., Pangrác, O., Schilling, H., Skutella, M.: Length-bounded cuts and flows. ACM Trans. Algorithms **7**(1), 4:1–4:27 (2010). DOI 10.1145/1868237.1868241. URL http://doi.acm.org/10.1145/1868237.1868241
3. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. Journal of Computer and System Sciences **75**(8), 423 – 434 (2009). DOI http://dx.doi.org/10.1016/j.jcss.2009.04.001. URL http://www.sciencedirect.com/science/article/pii/S0022000009000282
4. Courcelle, B.: Graph rewriting: An algebraic and logic approach. Handbook of Theoretical Computer Science pp. 194–242 (1990). URL http://ci.nii.ac.jp/naid/10009893142/en/
5. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015). DOI 10.1007/978-3-319-21275-3. URL http://dx.doi.org/10.1007/978-3-319-21275-3
6. Dahl, G., Gouveia, L.: On the directed hop-constrained shortest path problem. Operations Research Letters **32**(1), 15 – 22 (2004). DOI 10.1016/S0167-6377(03)00026-9. URL http://www.sciencedirect.com/science/article/pii/S0167637703000269
7. Dom, M., Lokshtanov, D., Saurabh, S., Villanger, Y.: Capacitated domination and covering: A parameterized perspective. In: M. Grohe, R. Niedermeier (eds.) Parameterized and Exact Computation, *Lecture Notes in Computer Science*, vol. 5018, pp. 78–90. Springer Berlin Heidelberg (2008). DOI 10.1007/978-3-540-79723-4_9. URL http://dx.doi.org/10.1007/978-3-540-79723-4_9
8. Dvořák, P., Knop, D.: Parametrized complexity of length-bounded cuts and multi-cuts. In: Theory and Applications of Models of Computation - 12th Annual Conference, TAMC 2015, Singapore, May 18-20, 2015, Proceedings, pp. 441–452 (2015). DOI 10.1007/978-3-319-17142-5_37. URL http://dx.doi.org/10.1007/978-3-319-17142-5_37
9. Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM **5**(6), 345 (1962). DOI 10.1145/367766.368168. URL http://doi.acm.org/10.1145/367766.368168
10. Fluschnik, T., Hermelin, D., Nichterlein, A., Niedermeier, R.: Fractals for kernelization lower bounds, with an application to length-bounded cut problems. In: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy, pp. 25:1–25:14 (2016). DOI 10.4230/LIPIcs.ICALP.2016.25. URL http://dx.doi.org/10.4230/LIPIcs.ICALP.2016.25
11. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Canadian journal of mathematics **8**(3), 399–404 (1956)
12. Golovach, P.A., Thilikos, D.M.: Paths of bounded length and their cuts: Parameterized complexity and algorithms. Discrete Optimization **8**(1), 72 – 86 (2011). DOI 10.1016/j.disopt.2010.09.009. URL http://www.sciencedirect.com/science/article/pii/S1572528610000678
13. Gutin, G., Jones, M., Wahlström, M.: Structural parameterizations of the mixed chinese postman problem. In: N. Bansal, I. Finocchi (eds.) Algorithms – ESA 2015, *Lecture Notes in Computer Science*, vol. 9294, pp. 668–679. Springer Berlin Heidelberg (2015). DOI 10.1007/978-3-662-48350-3_56. URL http://dx.doi.org/10.1007/978-3-662-48350-3_56
14. Huygens, D., Labbé, M., Mahjoub, A.R., Pesneau, P.: The two-edge connected hop-constrained network design problem: Valid inequalities and branch-and-cut. Networks **49**(1), 116–133 (2007). DOI 10.1002/net.20146. URL http://dx.doi.org/10.1002/net.20146
15. Itai, A., Perl, Y., Shiloach, Y.: The complexity of finding maximum disjoint paths with length constraints. Networks **12**(3), 277–286 (1982). DOI 10.1002/net.3230120306. URL http://dx.doi.org/10.1002/net.3230120306
16. Kloks, T.: Treewidth, Computations and Approximations (Lecture notes in computer science, 842). Springer-Verlag New York, Inc. (1994)

17. Nešetřil, J., de Mendez, P.: Sparsity: Graphs, Structures, and Algorithms. Algorithms and Combinatorics. Springer Berlin Heidelberg (2012). URL https://books.google.cz/books?id=n8UuUePJ_iIC
18. Nešetřil, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. European Journal of Combinatorics **27**(6), 1022 – 1041 (2006). DOI http://dx.doi.org/10.1016/j.ejc.2005.01.010. URL http://www.sciencedirect.com/science/article/pii/S0195669805000570
19. Orlin, J.B.: Max flows in o(nm) time, or better. In: Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013, pp. 765–774 (2013). DOI 10.1145/2488608.2488705. URL http://doi.acm.org/10.1145/2488608.2488705
20. Warshall, S.: A theorem on boolean matrices. J. ACM **9**(1), 11–12 (1962). DOI 10.1145/321105.321107. URL http://doi.acm.org/10.1145/321105.321107