Networks, Vol. 46(3), pp. 124-135, 2005

Fixed-Parameter Tractability and Data Reduction for Multicut in Trees*

Jiong Guo and Rolf Niedermeier
Institut für Informatik,
Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2,
D-07743 Jena, Germany.
{guo,niedermr}@minet.uni-jena.de.

May 26, 2005

Abstract

We study an NP-complete (and MaxSNP-hard) communication problem on tree networks, the so-called Multicut in Trees: given an undirected tree and some pairs of nodes of the tree, find out whether there is a set of at most k tree edges whose removal separates all given pairs of nodes. Multicut has been intensively studied for trees as well as for general graphs mainly from the viewpoint of polynomial-time approximation algorithms. By way of contrast, we provide a simple fixed-parameter algorithm for Multicut in Trees showing

*Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4. Parts of this work have been done while the authors were at the Universität Tübingen.

fixed-parameter tractability with respect to parameter k. Moreover, based on some polynomial-time data reduction rules which appear to be of particular interest from an applied point of view, we show a problem kernel for MULTICUT IN TREES by an intricate mathematical analysis.

Keywords. NP-hard problems, MULTICUT IN TREES, exact algorithms, fixed-parameter tractability, data reduction rules, problem kernel.

1 Introduction

Many hard network problems become easy when restricted to trees. There are, however, notable exceptions of important graph problems that remain hard even on trees. A well-known example is the BANDWIDTH MINIMIZATION problem restricted to trees of maximum node degree three, where it remains NP-complete [17]. In this paper we deal with another well-studied graph problem that remains NP-complete when restricted to trees [13].

The problem is MULTICUT IN TREES, where we are asked to remove a minimum number of edges of an unweighted tree in order to disconnect all pairs of nodes given in a set of "route demands." We refer to Sect. 2 for the formal definition. See Costa, Létocart, and Roupin [8] for a recent survey on MULTICUT problems. It follows from the work of Garg, Vazirani, and Yannakakis [13] that MULTICUT IN TREES is NP-complete and MaxSNP-hard. Whereas the latter implies that polynomial-time approximation schemes are out of reach, a factor-2 polynomial-time approximation algorithm is known [13]. By way of contrast, we investigate the exact solvability of this problem by exponential-time algorithms.

Exact algorithms for NP-hard problems have become a flourishing field of research [15, 24]. In particular, fixed-parameter algorithms may often be seen as valuable alternatives to approximation algorithms [9, 10, 11, 12, 18, 19]. The basic idea is to try to derive exact algorithms with a combinatorial explosion (i.e., exponential running time factor) that can be confined to some (hopefully small) problem parameters. As to fixed-parameter algorithms for MULTICUT IN TREES, we are aware of only one result concerning a completely different "parameterization." For the more general edge-weighted case of MULTICUT IN TREES with respect to the parameter $d :=$ "maximum number of paths passing through a node or an edge," a fixed-parameter dynamic programming algorithm derives from our very recent algorithm for the so-called "TREE-LIKE WEIGHTED SET COVER" [14]. The corresponding combinatorial explosion amounts to 3^d. By way of contrast, for (unweighted) MULTICUT IN TREES we give a fixed-parameter algorithm for the parameter $k :=$ "number of edges that are removed." The corresponding combinatorial explosion amounts to 2^k. Moreover, we provide an in-depth analysis on data reduction by preprocessing for MULTICUT IN TREES. Altogether, we show that MULTICUT IN TREES is amenable to two core techniques of parameterized algorithm design—bounded search trees and reduction to a problem kernel. The main mathematical contribution of this paper is the derivation of the bound on the problem kernel.

Since our main technical result is the derivation of a polynomial-time problem kernelization for MULTICUT IN TREES, let us discuss the general issue in more detail. Quoting Fellows [11, Page 9] from one of his recent surveys, "data reduction and kernelization rules are one of the primary outcomes of research on parameterized complexity." It has become commonplace now, going back to work of Cai et al. [5], that every fixed-parameter tractable problem is kernelizable. This observation exclusively relies on asymptotic mathematical considerations concerning the running time bounds, though, and it is of no algorithmic or practical use. Hence, the usefulness of problem kernelization is tied to the concrete development of effective data reduction rules that work in polynomial time and, for instance, can be used in a preprocessing phase to shrink the given problem instance[1]. As a matter of practical experience, we are far from being allowed to expect that showing fixed-parameter tractability "automatically" brings along data reduction rules for a problem kernelization. In fact, many fixed-parameter tractable problems still await the development of effective data reduction rules. Weihe [22, 23] gave a striking example for the effectiveness of data reduction rules for a domination-like graph problem occurring in the context of railway optimization. Two simple data reduction rules followed by simple brute-force search on small isolated components sufficed to optimally solve all real-world instances he considered. The "drawback" is that his rules obviously do not suffice to mathematically prove a problem kernel for the considered problem—in fact, according to parameterized complexity theory [10], the considered problem is fixed-parameter intractable and, because of that, will not allow for a problem kernel in the strict mathematical sense following the formal definition given in Section 2. By way of contrast, for the fixed-parameter tractable graph problem Vertex Cover, there is even a linear size problem kernel [1, 6] that can be computed in polynomial-time based on efficient data reduction.

In this paper, we provide a seemingly first example for a problem kernel of size exponential with respect to parameter k (more precisely, size $O(k^{3k})$)

[1]Observe, however, that as a matter of theoretical [20] as well as practical experience, data reduction rules are not only useful in a preprocessing phase but should be applied again and again during the whole solution process.

where it seems hard to show a polynomial or even linear size problem kernel. At first glance, this seems a little disappointing because the size of the problem kernel exceeds the size of the search tree we derive ($O(2^k)$). However, firstly, one has to take into account that Multicut in Trees is a more general problem than Vertex Cover, already making problem kernelization a harder thing to do. Secondly, the developed data reduction rules are of comparable simplicity as the ones developed by Weihe for his problem such that we nevertheless may expect a strong practical impact of our rules. Observe that all our bounds are purely worst-case results (relying on very special or even artificial cases that may very rarely occur) and the practical experiences for real-world or other test data sets may be much better. Thirdly, our extensive worst-case analysis of the problem kernel size and the discovered "worst-case structures" may help to spot future points of attack for improved kernelization strategies etc. on the one hand or to get a better understanding of what really makes the problem so hard on the other hand. Fourthly, we consider it as a worthwhile task of also purely mathematical interest to show upper size bounds on problem kernels. It took us a significant amount of time until we were able to prove the above stated worst-case bound on the problem kernel and we conjecture that it will be a hard task to reduce this bound to a polynomial in k.

2 Preliminaries

We prove fixed-parameter tractability results for an NP-complete problem. Formally, a (parameterized) problem is *fixed-parameter tractable* if it has a solution algorithm that runs in $O(f(k) \cdot n^c)$ time, where f is an arbitrary computable function only depending on an input parameter k, n is the problem size, and c is a constant [10, 19] (see [9, 11, 12, 18] for recent surveys). A core tool in the development of fixed-parameter algorithms are *data reduction rules*, often yielding a *reduction to a problem kernel*. Here, the goal is, given any problem instance I with parameter k, to transform it in polynomial time into a new instance I' with parameter k' such that the size of I' is bounded by a function depending only on k', $k' \leq k$, and (I, k) has a solution iff (I', k') has a solution. See Abu-Khzam et al. [1] for a recent and thorough investigation of reduction to a problem kernel (also called *kernelization*) for the best-studied parameterized problem Vertex Cover (the problem parameter there being the size of the vertex cover set) from a theoretical as well as a practical side.

We need some special notation concerning networks (trees). We often *contract an edge e*. Let $e = \{v, w\}$ and let $N(v)$ and $N(w)$ denote the sets of neighbors of v and w, respectively. Then, contracting e means that we replace v and w by one new node x and we set $N(x) := (N(v) \cup N(w)) \setminus \{v, w\}$. Using an adjacency list representation of graphs, edge contraction can be done in constant time. We occasionally consider paths P_1 and P_2 in the tree and we write $P_1 \subseteq P_2$ when the node set (and edge set) of P_2 contains that of P_1.

The (unweighted) Multicut in trees problem is defined as follows:

> **Input:** An undirected tree $T = (V, E)$, $n := |V|$, and a collection H of m pairs of nodes in V, $H = \{(u_i, v_i) \mid u_i, v_i \in V, u_i \neq v_i, 1 \leq i \leq m\}$.
>
> **Task:** Find a subset E' of E of minimum size whose removal separates each pair of nodes in H.

Note that by removing edges a tree decomposes into subtrees forming a forest. Then, two nodes are *separated* if they are in different trees of the forest. An edge subset E' of E as specified above is called a *multicut.* We refer to a pair of nodes $(u_i, v_i) \in H$ as a *demand path* P due to the fact that, in a tree, the path is uniquely determined by u_i and v_i. To turn MULTICUT IN TREES into a parameterized problem, we will add a nonnegative integer k as a further input and we replace the above task by the following one.

> **Task:** Find a subset E' of E with $|E'| \leq k$ such that the removal of E' separates each pair of nodes in H.

MULTICUT IN TREES was shown to be NP-complete and MaxSNP-hard even for an input tree being a star [13][2]. Garg, Vazirani, and Yannakakis [13] gave a factor-2 approximation algorithm that also works for the more general case with edge weights. Călinescu, Fernandes, and Reed [7] provided a polynomial-time approximation scheme (PTAS) for finding unweighted multicuts in graphs with bounded degree and bounded treewidth. Very recently, as a corollary to a result for "TREE-LIKE WEIGHTED SET COVER" we gave a fixed-parameter algorithm for weighted MULTICUT IN TREES, solving the problem optimally in $O(3^d \cdot mn^2)$ time [14]. Herein, parameter d denotes the maximum number of paths passing through a node or an edge of the given tree. Here, we complement the above results by showing fixed-parameter tractability for MULTICUT IN TREES with respect to the perhaps most natural parameterization—the parameter k now being the size of E'. Our exact solution algorithm has the exponential factor 2^k. The mathematically most demanding part, however, is to show that few simple and efficient data reduction rules running in polynomial time lead to a problem

[2]More specifically, this special case is shown to be equivalent to VERTEX COVER, also with respect to approximability [13].

kernel. Finally, we mention in passing that Anand et al. [4] proved a fixed-parameter tractability result for the related multicommodity flow problem in trees. More specifically, for the parameter l referring to the "rejected flows," they have the combinatorial explosion $2^l l!$ in the super-polynomial part of the complexity of their fixed-parameter algorithm.

3 A Simple Algorithm and First Data Reduction Rules

In this section we start with easy observations concerning the fixed-parameter tractability of Multicut in Trees. In particular, we sketch a simple search tree with size bounded above by 2^k—this size bound seems hard to improve, though.

3.1 Bounded Search Tree Algorithm

We show the fixed-parameter tractability of Multicut in Trees by giving a simple depth-bounded search tree of size at most 2^k: Consider an instance of Multicut in Trees with an undirected and unrooted tree T and a collection of node pairs H. We first root T at an arbitrary node. Then, for each node pair $(u, v) \in H$, we find the *least common ancestor* of u and v, i.e., the node w that is an ancestor of both u and v and that has the greatest depth in T measured by the distance from the root. Observe that w has to be on the unique path between u and v. Then, we process the node pairs in H in the non-ascending order of the depth of their least common ancestors. For a node pair u and v which is not yet separated, if the least common ancestor w is one of u and v, then we delete the edge which is incident to this node and lies on the uniquely determined path between u and v; otherwise, the path between u and v has two edges incident to w. We branch into two cases, each case representing the deletion of one of the two edges. After deleting an edge, we remove all non-connected node pairs from H and then proceed with the next node pair in H. Since only k edge deletions are allowed, we have a bounded search tree of size at most 2^k. This results in the following theorem.

Theorem 1. MULTICUT IN TREES *can be solved in* $O(2^k \cdot mn)$ *time, where* k *denotes the maximum number of tree edges that may be removed.*

Proof. The correctness of the algorithm follows directly from its description. At each node of the search tree, we delete all node pairs from H which are no longer connected. This can be clearly done in $O(mn)$ time. $\square$

3.2 Parameter-Independent Data Reduction Rules

The following four simple data reduction rules are of central importance for deriving a problem kernel for MULTICUT IN TREES in the next section. We can often observe that data reduction rules are very useful in practice. In particular, this is true for data reduction rules that are *independent* of the parameter k as, for example, the ones given in the case of the NP-complete DOMINATING SET problem [2, 3]. We call a data reduction rule independent of the parameter k if it can be applied without any knowledge of the value of k. Four more data reduction rules whose applicability depends on the parameter k will be given in Section 4.

Idle Edge. If there is a tree edge with no demand path passing through it, then contract this edge.

Unit Path. If a demand path has length one, then the corresponding edge e has to be in E'. Contract e and remove all demand paths passing through e and decrease the parameter k by one.

Dominated Edge. If all demand paths that pass through edge e_1 of T also pass through edge e_2 of T, then contract e_1.

Dominated Path. If $P_1 \subseteq P_2$ for two demand paths, then delete P_2.

We term such reduction rules as *correct* if a Multicut in Trees instance (T, H) with parameter k has a yes-solution iff the newly generated instance (T', H') with parameter k' has a yes-solution. Observe that $k' < k$ only if the Unit Path rule applies.

Lemma 1. *The above four reduction rules are correct and they can be executed in $O(mn^3 + m^3n)$ worst-case time such that finally no more rules are applicable.*

Proof. The correctness of the Idle Edge and Unique Path rules is easy to observe. The Dominated Edge rule is correct since, if all demand paths that pass through edge e_1 also pass through edge e_2, then adding e_1 to E' is never better than adding e_2 to E'. The Dominated Path rule follows from the observation that if $P_1 \subseteq P_2$ for two demand paths, then each edge removal which destroys P_1 also destroys P_2.

Next, we estimate the running time for each particular rule. Then, we estimate the maximum overall running time of successive applications of these rules until none of them applies any more.

Idle Edge. During a depth-first traversal of the tree, we clearly can mark each edge e as to whether or not a path passes through e. Accordingly, e may be contracted. This is doable in $O(mn)$ time.

Unit path. Inspecting each demand path, this rule is executable in $O(m)$ time.

Dominated Edge. Basically, for each pair of edges in the tree we compare their corresponding sets of demand paths, i.e., the demand paths passing through these edges, respectively. Doing this for all of the $O(n^2)$ pairs, each comparison taking $O(m)$ time, we end up with $O(mn^2)$ time in total.

Dominated Path. Comparing all $O(m^2)$ pairs of demand paths, in each case we basically have to compare two paths of length $O(n)$, leading to $O(m^2n)$ running time.

Eventually, we have to estimate for each rule how often it may apply. Clearly, we have $O(n)$ possible applications for the first three rules. As to the fourth rule, $O(m)$ is an upper bound for the possible number of applications. Altogether, we thus can conclude that after $O(mn^3+m^3n)$ worst-case running time for applying the rules, none of them will be applicable any longer. □

Obviously, the running time bound of Lemma 1 gives a very rough estimate. In particular, it is conceivable that the reduction rules will perform much better in practical implementations and tests. This is a typical observation also for other data reduction rules with relatively high polynomial worst-case running times, as, for example, was observed for the data reduction rules for DOMINATING SET [2, 3].

4 Problem Kernel for MULTICUT IN TREES

In this section we introduce four more data reduction rules and, based on these rules, we prove the problem kernel for MULTICUT IN TREES by giving an upper bound on the size of the reduced input tree. Recall that a parameterized problem such as MULTICUT IN TREES is said to have a problem kernel if, after the application of the data reduction rules, the resulting instance (T, H) with parameter k has size $f(k)$ for a function f depending only on k. In order to simplify the presentation, we adopt a stepwise manner. That is, first we show a bound for a very special case of a tree, a *caterpillar*, and then for a *spider of caterpillars* which is also a special case of a tree but can contain several caterpillars as subtrees. Finally, a bound will be given for general trees, implying the main result of this work.

4.1 Some Notations and Definitions

In the next subsections the bound on the size of the input tree $T = (V, E)$ (and, thus, also the set of demand pairs H) will be achieved by first partitioning the nodes of T into six disjoint sets and then giving for each of these node sets a bound on its size. For an undirected and unrooted tree T, we distinguish two sorts of nodes, *leaves* having only one incident edge and *inner nodes* having more than one incident edge. The sets of leaves and inner nodes are denoted by L and I, respectively. For an inner node v, we call the leaves (if existing) adjacent to it *v's leaves.*

The desired partition of V is then defined as follows:

- $I_1 := \{ v \in I \mid |N(v) \cap I| \leq 1\}$;
  Observe that, if we delete all leaves from T, the nodes in I_1 become leaves in the resulting tree.

- $I_2 := \{ v \in I \mid |N(v) \cap I| = 2\}$;
  Note that $I_2 \neq \emptyset$ only if $|I_1| \geq 2$.

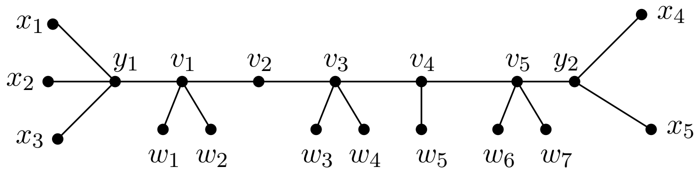

Figure 1: A caterpillar: There is no I_3-node and no L_3-leaf. Nodes y_1 and y_2 are the two I_1-nodes. In particular, $L_1 = \{x_1, x_2, x_3, x_4, x_5\}$, $I_2 = \{v_1, v_2, v_3, v_4, v_5\}$, and $L_2 = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$.

- $I_3 := \{\, v \in I \mid |N(v) \cap I| \geq 3\}$;

  Set I_3 is empty iff, by deleting all leaves from T, the resulting tree is a path.

- $L_1 := \{\, v \in L \mid N(v) \subseteq I_1\}$;

- $L_2 := \{\, v \in L \mid N(v) \subseteq I_2\}$;

- $L_3 := \{\, v \in L \mid N(v) \subseteq I_3\}$;

We use the terms L_i-*leaves* and I_i-*nodes* for $1 \leq i \leq 3$ in the obvious way.

The definitions of the special trees considered in the next subsections—caterpillar and spider of caterpillars—are as follows.

Definition 1. Caterpillar

Given a tree $T = (V, E)$, we partition V as described above. A tree $T = (V, E)$ is a caterpillar *if $|I_1| = 2$ and $|I_3| = 0$. Then, the inner nodes of I_2 form a path between the two nodes in I_1. We call this path the* backbone *of the caterpillar.*

Definition 2. Spider of Caterpillars

Given a tree $T = (V, E)$, we partition V as described above. A tree $T =$

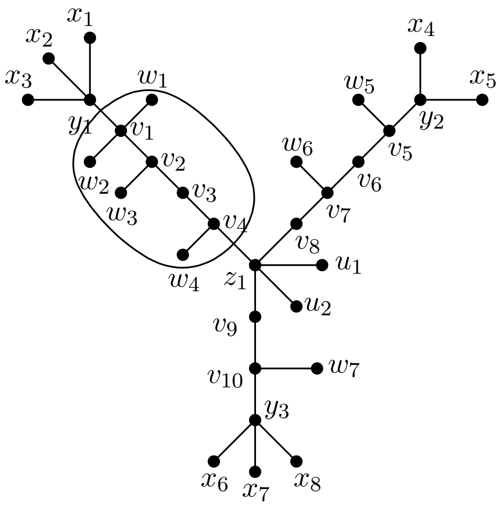

Figure 2: A spider of caterpillars: Node z_1 is the only I_3-node with $L_3 = \{u_1, u_2\}$. $I_1 = \{y_1, y_2, y_3\}$, $I_2 = \{v_1, \ldots, v_{10}\}$, $L_1 = \{x_1, \ldots, x_8\}$, and $L_2 = \{w_1, \ldots, w_7\}$. The *oval* depicts a maximal caterpillar component.

(V, E) *is a* spider of caterpillars *if* $|I_3| = 1$. *Then, the inner nodes of* I *induce a spider with the* I_3*-node as the center node. The paths induced by the* I_2*-nodes are called the* backbones *of the spider.*

Figure 1 and Figure 2 display examples for a caterpillar and a spider of caterpillar, respectively.

Finally, we define the "caterpillar component" of a tree as follows:

Definition 3. *Given a tree* $T = (V, E)$*, we partition* V *as described above. A* caterpillar component *of* T *is an induced subtree of* T *exclusively consisting of* I_2*-nodes and their* L_2*-leaves.*

Note that a caterpillar component can be contained in other caterpillar components. We call a caterpillar component *maximal* if it is not contained in any other caterpillar component. See Figure 2 for an example of a maximal caterpillar component. Clearly, the set of all maximal caterpillar components of a tree is unique and no two maximal caterpillar components intersect each other. We say that an I_1-node or an I_3-node is *adjacent to a caterpillar component* if it is adjacent to an I_2-node of the component.

4.2 Further Parameter-Dependent Data Reduction Rules

In this subsection we extend our set of so far four reduction rules (Section 3.2) by four more rules. We need these rules to show the bound on the size of the reduced input tree, the problem kernel. Note, however, that these rules depend on the parameter value k.

Disjoint Paths. If an instance of MULTICUT IN TREES has more than k pairwise edge-disjoint demand paths, then there is no solution with parameter value k.

Overloaded Edge. If more than k length-two demand paths pass through an edge e, then contract e, remove all demand paths going through e, and decrease the parameter k by one.

Overloaded Caterpillar. If there are $k+1$ demand paths (v, u_1), (v, u_2), ..., (v, u_{k+1}) such that nodes $u_1, \ldots, u_{k+1}$ belong to the same caterpillar component that does not contain v, then (one of) the longest of these demand paths can be deleted.

Overloaded L_3-Leaves. If there are $k+1$ demand paths (v, u_1), (v, u_2), ..., (v, u_{k+1}) such that nodes $u_1, \ldots, u_{k+1}$ are all L_3-leaves of an I_3-node u, then remove all these demand paths and add a new demand path between v and u.

Lemma 2. *The above four reduction rules are correct and they can be executed, together with the four rules in Section 3.2, in polynomial time such that finally no further rule is applicable.*

Proof. Disjoint Paths. The correctness of this reduction rule is obvious since, for every two edge-disjoint demand paths, we need to add at least two edges to E'. The maximum edge-disjoint paths problem can be solved for trees in polynomial time [13].

Overloaded Edge. The correctness of this rule follows from the fact that if edge e were not contracted, then one would need to remove more than k edges in order to cut all length-two demand paths passing through e. Note that the Overloaded Edge rule is "similar in spirit" to the removal of high-degree vertices in the well-known data reduction rule for Vertex Cover attributed to Buss [10]. It can be clearly done in $O(n \cdot m)$ time.

Overloaded Caterpillar. In order to cut more than k demand paths by removing only k edges one has to remove an edge that is covered by at least two demand paths. Then, however, a longest demand path is always cut and, hence, it can be omitted. Since there can be $O(n^2)$ caterpillar components, one for each node pair, this rule can be done in $O(n^3 \cdot m)$ time.

Overloaded L_3-Leaves. In order to cut these more than k demand paths by removing only k edges one has to remove at least one edge on the path between u and v. Then, cutting these demand paths is equivalent to cutting a demand path between u and v. This rule can clearly be done in $O(n \cdot m)$ time, since there are at most m demand paths starting at a node.

Together with the polynomial running time of the four rules in Section 3.2, we get the polynomial running time for all these rules. □

Our main result concerning problem kernelization refers to reduced instances of MULTICUT IN TREES:

Definition 4. *We call an instance of* MULTICUT IN TREES reduced *when none of the eight given data reduction rules applies.*

4.3 Some Observations on Reduced Instances

With the data reduction rules given in Sections 3.2 and 4.2, we arrive at the following observations on a reduced instance of MULTICUT IN TREES. Without loss of generality, we assume that the reduced tree instance has at least three nodes.

Lemma 3. *In a reduced instance, each I_1-node has at least two L_1-leaves adjacent to it.*

Proof. Consider an I_1-node u of the reduced instance. It has at least one L_1-leaf. Suppose that u has only one L_1-leaf called v. Since the instance has at least three nodes, there is another node $w \neq v$ adjacent to u. From the assumption that u has only one L_1-leaf, w is an inner node. Due to the Idle Edge rule there must be a demand path starting at v. Furthermore, because of the Unit Path rule all demand paths going through edge $\{u, v\}$ have to go through edge $\{u, w\}$ as well. This, however, means that the Dominated Edge rule could be applied to $\{u, v\}$, a contradiction to the fact that the given instance is reduced. □

Lemma 4. *In a reduced instance, for each L_1-leaf v adjacent to an I_1-node u, there exists a demand path between v and another L_1-leaf of u.*

Proof. Assume that there is an L_1-leaf v adjacent to u with $u \in I_1$ and there is no demand path between v and other L_1-leaves of u. Note that by

Lemma 3 u has at least two L_1-leaves. Since the instance is reduced, due to the Idle Edge rule there must be a demand path starting at v. Moreover, the Unit Path rule implies that each demand path starting at v then also has to pass an edge different from $\{u, v\}$. This implies that u has a uniquely determined inner node w adjacent to it and all demand paths starting at v also pass $\{u, w\}$. But then the Dominated Edge rule would apply to edge $\{u, v\}$, a contradiction to the fact that the instance is reduced. □

Lemma 5. *In a reduced instance, there are at most k edge-disjoint demand paths.*

Proof. The claim follows directly from the Disjoint Paths rule. □

Lemma 6. *In a reduced instance, there are at most k^2 length-2 demand paths.*

Proof. The claim follows from the fact that there are at most k edge deletions allowed and, due to the Overloaded Edge rule, deleting one edge can destroy at most k length-2 demand paths. □

Lemma 7. *In a reduced instance, there can be at most $2k^2$ L_1-leaves.*

Proof. This claim directly follows from Lemma 4 and Lemma 6. □

Lemma 8. *In a reduced instance, there can be at most k I_1-nodes and at most $k - 1$ I_3-nodes.*

Proof. Lemma 3 and Lemma 4 imply that for each I_1-node, there is at least one length-2 demand path between two of its L_1-leaves. Moreover, the length-2 demand paths for different I_1-nodes are pairwise edge-disjoint. Then, by Lemma 5, there can be at most k I_1-nodes. Furthermore, consider the subgraph T' of the input tree T that is induced by the inner nodes of T. It is clear that T' is a tree and the leaves of T' correspond one-to-one to the I_1-nodes of T. Since, in a tree with k leaves, there are at most $k-1$ inner nodes having at least three neighbors, it is easy to derive that $|I_3| \leq k-1$. $\square$

Now, with Lemma 7 and Lemma 8, it "only" remains to show that the sizes of sets I_2, L_2, and L_3 of a reduced MULTICUT IN TREES instance can be bounded by a function in k. To this end, we need the following two lemmas which are decisive for showing the size bound of L_3 and $I_2 \cup L_2$, respectively.

Lemma 9. *For each I_3-node u in a reduced instance, each of its L_3-leaves is the starting point of at least two demand paths which pass through two distinct neighbors of u.*

Proof. Consider an L_3-leaf v of u. If only one demand path starts at v, then either the Unit Path rule or the Edge Domination rule would apply to edge $\{u, v\}$. If all demand paths starting at v passed only through one neighbor $w \neq v$ of u, then the Edge Domination rule would apply to edges $\{u, v\}$ and $\{u, w\}$. This is a contradiction to the fact that the input instance is reduced. $\square$

Lemma 10.

1. *In a reduced instance, an I_2-node v having no L_2-leaf adjacent to it has to be a starting point of at least two demand paths, passing through two distinct inner nodes adjacent to v.*
2. *In a reduced instance, for an I_2-node v with some L_2-leaves adjacent to it, each of these L_2-leaves has at least two demand paths passing through two distinct neighbors of v.*

Proof. 1. Consider an I_2-node v with two adjacent inner nodes u and w. If there is no demand path starting at v and passing through u, then all demand paths passing through edge $\{u, v\}$ also pass through edge $\{v, w\}$ and, hence, the Edge Domination rule would apply. This contradicts the fact that the input instance is reduced. If there is no demand path starting at v and passing through w, an analogous argument applies.

2. Consider an I_2-node v with two adjacent inner nodes u and w where v has r L_2-leaves $w_1, w_2, \ldots, w_r$. Note that due to the Unit Path rule all demand paths have length at least two. If there is only one demand path starting at w_i, $1 \leq i \leq r$, then clearly the Edge Domination rule applies to the edge $\{v, w_i\}$. The Edge Domination rule also applies to $\{v, w_i\}$ when all demand paths starting at w_i either pass through edge $\{u, v\}$, or $\{v, w\}$, or $\{v, w_j\}$ for $i \neq j$. □

In the following we prove the size of the problem kernel first for caterpillars, then for spiders of caterpillars, and finally for general trees. More precisely, in the case of caterpillars where there is neither an I_3-node nor a L_3-leaf, we show how to bound the size of $I_2 \cup L_2$. In the case of spiders of caterpillars where there is only one I_3-node, we present the basic idea for showing the size bound for L_3. The problem kernel size for general trees follows by combining the arguments developed for the first two cases.

4.4 Problem Kernel for Caterpillars

With the observations made in Section 4.3, we show the problem kernel for MULTICUT IN TREES when the input tree is restricted to be a caterpillar[3].

[3] MULTICUT IN CATERPILLARS is also NP-complete, even if the tree nodes have at most five neighbors [16]. The reduction is very similar to the one used by Călinescu et al. [7]

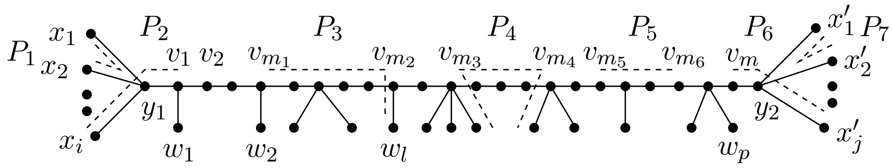

Figure 3: An instance of MULTICUT IN CATERPILLARS. There are m I_2-nodes, $v_1, \ldots, v_m$, and p L_2-leaves, $w_1, \ldots, w_p$. The backbone of this caterpillar is the path between v_1 and v_m. The dashed lines denote seven edge-disjoint demand paths, $P_1, \ldots, P_7$.

Consider a reduced instance with a caterpillar $T = (V, E)$, i.e., there are no I_3-nodes, no L_3-leaves, and there are exactly two I_1-nodes, y_1 and y_2, as illustrated in Figure 3. Since the number of L_1-leaves is bounded by $2k^2$ (Lemma 7), in order to give a bound on the size of V it suffices to show that $|I_2| + |L_2|$ is bounded by a function of k.

As illustrated in Figure 3, we assume that the inner nodes of the caterpillar are ordered on a line, the first is y_1, the last is y_2, and the I_2-nodes in between are ordered from left to right in ascending order of their indices. An I_2-node v_i is to the right of another I_2-node v_j if $i > j$. Furthermore, we use H_{I_2} to denote the set of the demand paths in H which pass through at least one I_2-node. We define the "right backbone endpoint" of a demand path in H_{I_2} as the I_2-node with highest index among the I_2-nodes that the demand path passes through. The "left backbone endpoint" is defined symmetrically. In Figure 3, $H_{I_2} = \{P_2, P_3, P_4, P_5, P_6\}$. The left backbone endpoint of demand path P_3 is v_{m_1} and the right backbone endpoint of P_3 is v_{m_2}.

By Lemma 5, there can be at most k edge-disjoint demand paths in T.

to show the NP-completeness of MULTICUT IN BINARY TREES (Theorem 6.1 in [7]).

In the following, we show that there is a maximum cardinality set of edge-disjoint demand paths with some special properties. These special properties are useful for giving a bound on the size of the reduced caterpillar.

Lemma 11. *In polynomial time one can find a maximum cardinality set of edge-disjoint demand paths $\mathcal{P} := \{P_1, P_2, \ldots, P_l\}$ with $l \leq k$ which has the following two properties.*

Property (1). *One demand path is between two L_1-leaves of y_1 and one is between two L_1-leaves of y_2. Let P_1 and P_l denote these two demand paths; then we have $\{P_1, P_l\} \subseteq (\mathcal{P} \setminus H_{I_2})$.*

Property (2). *If the paths in $\mathcal{P} \cap H_{I_2}$ are ordered in ascending order of the indices of their left backbone endpoints, i.e., for $P_i, P_j \in (\mathcal{P} \cap H_{I_2})$ with $i < j$, P_i's left backbone endpoint is to the left of P_j's left backbone endpoint, then, for each $P_i \in (\mathcal{P} \cap H_{I_2})$ with $1 < i < l$,*

- *there is no other demand path $P \in (H_{I_2} \setminus \mathcal{P})$ such that P is edge-disjoint to all paths in $\mathcal{P} \setminus \{P_i\}$ and P_i's right backbone endpoint is to the right of P's right backbone endpoint;*
- *there is no other demand path $P \in (H_{I_2} \setminus \mathcal{P})$ such that P is edge-disjoint to all paths in $\mathcal{P} \setminus \{P_i\}$, P and P_i have the same right backbone endpoint, and P_i's left backbone endpoint is to the left of P's left backbone endpoint.*

Proof. Since a maximum cardinality set of edge-disjoint demand paths can be found in polynomial time [13], we only need to show how to, in polynomial time, modify an arbitrary maximum cardinality set of edge-disjoint demand paths $\mathcal{P}$ such that it fulfills the above properties.

Property (1). By Lemma 4, there always exists for an I_1-node a demand path between two of its L_1-leaves. Without loss of generality, assume that there is a demand path P between x_1 and x_2, two L_1-leaves of y_1. If $\mathcal{P}$ contains no demand path between two of y_1's L_1-leaves, i.e., $P \notin \mathcal{P}$, then there must be a demand path P' in $\mathcal{P}$ passing through one of the edges $\{x_1, y_1\}$ and $\{x_2, y_1\}$; otherwise, P would be edge-disjoint to all demand paths in $\mathcal{P}$, a contradiction to the maximality of $\mathcal{P}$. Moreover, P' cannot end in y_1 since T is reduced with respect to the Unit Path rule. Therefore, P' has to pass the edge $\{y_1, v_1\}$. Then, replacing P' by P in $\mathcal{P}$, the resulting set remains a maximum cardinality set of edge-disjoint demand paths.

Property (2). For each $P_i \in (\mathcal{P} \cap H_{I_2})$, we can in $O(m \cdot n)$ time find all demand paths $P \in (H_{I_2} \setminus \mathcal{P})$ which are edge-disjoint to all paths in $\mathcal{P} \setminus \{P_i\}$, where m denotes the number of demand paths in H. Let v_i^l and v_i^r denote P_i's left and right backbone endpoints. For each of these paths P with v^l and v^r denoting P's left and right backbone endpoints, to check whether v_i^r is to the right of v^r or $v_i^r = v_r$ and v_i^l is to the left of v^l can be done in constant time. If there exists such a demand path P for P_i, replace P_i by P; the demand paths in $\mathcal{P}$ remain pairwise edge-disjoint. $\square$

Based on a maximum cardinality set of edge-disjoint demand paths $\mathcal{P}$ as given in Lemma 11, we prove the main theorem of this subsection.

Theorem 2. MULTICUT IN CATERPILLARS *has a problem kernel which consists of a caterpillar containing at most $O(k^{k+1})$ nodes.*

Proof. Suppose that we have computed a maximum cardinality set of edge-disjoint demand paths $\mathcal{P}$ as described in Lemma 11. We assume that P_1 is between x_1 and x_2, two L_1-leaves of y_1, and P_l is between x_1' and x_2', two L_1-leaves of y_2. Note that there can be more than one path in $\mathcal{P}$ between two L_1-leaves of y_1 (or y_2). However, it will be clear from the following analysis that we can derive a better bound on the size of the caterpillar if there is more than one path in $\mathcal{P}$ between L_1-leaves of y_1 (or y_2). Therefore, we assume that none of $P_2, \ldots, P_{l-1}$ is between two L_1-leaves of y_1 or y_2. We use v_{l_i} and v_{r_i} to denote the left and right backbone endpoints of demand path P_i with $2 \leq i \leq l-1$, respectively. Furthermore, we partition the I_2-nodes together with their L_2-leaves into $2l-1$ sets and bound from above the size of each set. These sets are

$$\begin{aligned}
A_1 &:= \{ v_j \mid 1 \leq j \leq l_2 \} \cup \{ \text{their } L_2\text{-leaves} \}; \\
A_2 &:= \{ v_j \mid l_2 < j \leq r_2 \} \cup \{ \text{their } L_2\text{-leaves} \}; \\
A_3 &:= \{ v_j \mid r_2 < j \leq l_3 \} \cup \{ \text{their } L_2\text{-leaves} \}; \\
A_4 &:= \{ v_j \mid l_3 < j \leq r_3 \} \cup \{ \text{their } L_2\text{-leaves} \}; \\
&\vdots \\
A_{2l-2} &:= \{ v_j \mid l_{l-1} < j \leq r_{l-1} \} \cup \{ \text{their } L_2\text{-leaves} \}; \\
A_{2l-1} &:= \{ v_j \mid r_{l-1} < j \leq m \} \cup \{ \text{their } L_2\text{-leaves} \}.
\end{aligned}$$

Informally, the sets with odd indices contain the I_2-nodes which are not on any demand path in $\mathcal{P}$ together with the left backbone endpoints of these demand paths, while the sets with even indices contain the remaining I_2-nodes. Note that some of these sets can be empty since two consecutive demand paths can share an endpoint. In particular, if P_2 (or P_{l-1}) starts at an L_1-leaf and ends at v_1 (or v_m), then $A_2 = \emptyset$ (or $A_{2l-2} = \emptyset$).

First, consider the nodes in A_1. By Lemma 10, each I_2-node with no L_2-leaf has a demand path starting at it and going to its left, and each L_2-leaf in A_1 has a demand path starting at it and going to the left of the adjacent I_2-node. However, a demand path starting at a node v in A_1 and ending at a node left to it cannot end at a node in A_1; otherwise, this demand path would be edge-disjoint to all demand paths in $\mathcal{P}$, which contradicts the maximality of $\mathcal{P}$. With the same argument, this demand path cannot end at y_1 or one of $x_3, \ldots, x_i$. Thus, the other endpoint of the demand path can only be x_1 or x_2. Since T is reduced, there can be at most $2k$ demand paths starting at x_1 and x_2 and ending at one of the I_2-nodes and the L_2-leaves (due to the Overloaded Caterpillar rule). Thus, there can be at most $2k$ I_2-nodes without L_2-leaf and L_2-leaves in A_1. Since there are at most as many I_2-nodes with L_2-leaves as there are L_2-leaves, we can conclude that

$$|A_1| \leq 4k. \tag{1}$$

This analysis works analogously for $A_2, A_3, \ldots, A_{2l-1}$. The demand paths starting at a node in A_2 and going to its left cannot end at an A_2-node; otherwise, we have a demand path P which is edge-disjoint to P_1 and P_3 and which has either a right backbone endpoint to the left of v_{r_2} or a left backbone endpoint to the right of v_{l_2}, which contradicts the fact that $\mathcal{P}$ has Property (2) in Lemma 11. Then, the demand paths starting at a node in A_2 and going left can have only the nodes in A_1, y_1, or $x_1, \ldots, x_i$ as the other endpoint. For a node v in A_1, consider the demand paths starting at v and going right and ending at some I_2-nodes or their L_2-leaves. Since all I_2-nodes to the right of v together with their L_2-leaves induce a caterpillar component of T and v is outside this caterpillar component, then, using the fact that T is reduced with respect to the Overloaded Caterpillar rule, there can be at most k demand paths starting from v and ending at some nodes to the right of it. Therefore, with $|L_1| \leq 2k^2$ (Lemma 7), we get

$$|A_2| \leq k \cdot (|\{x_1, x_2, \ldots, x_i\} \cup \{y_1\}| + |A_1|) \leq k \cdot (2k^2 + 1 + |A_1|).$$

Analogously, we have a bound on $|A_r|$ for an arbitrary r with $3 \le r \le 2l-1$,

$$|A_r| \le k \cdot (2k^2 + 1 + \sum_{j=1}^{r-1} |A_j|). \tag{2}$$

Therefore,

$$\begin{aligned}
\sum_{j=1}^{2l-1} |A_j| &= \sum_{j=1}^{2l-2} |A_j| + |A_{2l-1}| \\
&\overset{(2)}{\le} \sum_{j=1}^{2l-2} |A_j| + k \cdot (2k^2 + 1 + \sum_{j=1}^{2l-2} |A_j|) \\
&\le (k+1) \cdot (\sum_{j=1}^{2l-2} |A_j| + 2k^2 + 1) \\
&\overset{(2)}{\le} (k+1) \cdot (\sum_{j=1}^{2l-3} |A_j| + k \cdot (2k^2 + 1 + \sum_{j=1}^{2l-3} |A_j|) + 2k^2 + 1) \\
&= (k+1)^2 \cdot (\sum_{j=1}^{2l-3} |A_j| + 2k^2 + 1) \\
&\le (k+1)^{2l-2} \cdot (|A_1| + 2k^2 + 1) \\
&\overset{(1)}{\le} (k+1)^{2l-2}(4k + 2k^2 + 1) \\
&= O(k^{2l}).
\end{aligned}$$

However, this bound can be improved if we take into account the symmetry of the caterpillar structure: Observe that the analysis for $|A_{2l-1}|$ can be done in the same way as for $|A_1|$, the analysis for $|A_{2l-2}|$ can be done in the same way as for $|A_2|$, and so on. Therefore, the bound on the number of I_2-nodes and L_2-leaves is as follows:

$$
\begin{aligned}
|I_2| + |L_2| &= \sum_{j=1}^{2l-1} |A_j| \\
&= \sum_{j=1}^{l} |A_j| + \sum_{j=l+1}^{2l-1} |A_j| \\
&\overset{(2)}{\leq} (k+1) \cdot (\sum_{j=1}^{l-1} |A_j| + 2k^2 + 1) + (k+1) \cdot (\sum_{j=l+2}^{2l-1} |A_j| + 2k^2 + 1) \\
&\leq (k+1)^2 \cdot (\sum_{j=1}^{l-2} |A_j| + 2k^2 + 1) + (k+1)^2 \cdot (\sum_{j=l+3}^{2l-1} |A_j| + 2k^2 + 1) \\
&\leq (k+1)^{l-1} \cdot (|A_1| + 2k^2 + 1) + (k+1)^{l-2} \cdot (|A_{2l-1}| + 2k^2 + 1) \\
&\overset{(1)}{\leq} (k+1)^{l-1}(4k + 2k^2 + 1) + (k+1)^{l-2}(4k + 2k^2 + 1) \\
&= O(k^{l+1}).
\end{aligned}
$$

Since there are at most k edge-disjoint paths in $\mathcal{P}$, that is, $l \leq k$, $|I_2| + |L_2| = O(k^{k+1})$. Together with $|L_1| \leq 2k^2$, $|I_1| = 2$, and $|I_3| = |L_3| = 0$, we have the claimed problem kernel size. □

4.5 Problem Kernel for Spiders of Caterpillars

The next special case of a tree, a spider of caterpillars T (also see Figure 2), has exactly one I_3-node u which is also the central node of the spider induced by the inner nodes. There are at most k I_1-nodes due to Lemma 8 and thus the number of maximal caterpillar components is bounded above by k. Each of these maximal caterpillar components is adjacent to u and to one I_1-node. We call the subgraph of T that consists of a maximal caterpillar component, its adjacent I_1-node, and the L_1-leaves of this I_1-node a *semi-caterpillar*. The backbone of a semi-caterpillar then means the path induced by the I_2-nodes in the maximal caterpillar component.

In the following we adapt the analysis in the proof of Theorem 2 to show the upper-bound on the size of T. Recall that the proof of Theorem 2 is heavily based on a special maximum cardinality set of edge-disjoint demand paths as described in Lemma 11. Therefore, we firstly need to show that, in a spider of caterpillars T, we can also find such special maximum cardinality sets of edge-disjoint demand paths.

Lemma 12. *For each of the semi-caterpillars of a spider of caterpillars T, one can in polynomial time find a maximum cardinality set $\mathcal{P} := \{P_1, P_2, \ldots, P_l\}$ of edge-disjoint demand paths passing through only edges of this semi-caterpillar which has the following two properties.*

Property (1). *Let y denote the only I_1-node in this semi-caterpillar. Then, one demand path in $\mathcal{P}$ is between two L_1-leaves of y. Let P_1 denote this demand path; then we have $P_1 \in (\mathcal{P} \setminus H_{I_2})$.*

Property (2). *If the paths in $\mathcal{P} \cap H_{I_2}$ are ordered in ascending order of the indices of their left backbone endpoints, i.e., for $P_i, P_j \in (\mathcal{P} \cap H_{I_2})$ with $i < j$, P_i's left backbone endpoint is to the left of P_j's left backbone endpoint, then, for each $P_i \in (\mathcal{P} \cap H_{I_2})$ with $1 < i < l$,*

- *there is no other demand path $P \in (H_{I_2} \setminus \mathcal{P})$ passing through only edges of this semi-caterpillar such that P is edge-disjoint to all paths in $\mathcal{P} \setminus \{P_i\}$ and P_i's right backbone endpoint is to the right of P's right backbone endpoint;*
- *there is no other demand path $P \in (H_{I_2} \setminus \mathcal{P})$ passing through only edges of this semi-caterpillar such that P is edge-disjoint to all paths in $\mathcal{P} \setminus \{P_i\}$, P and P_i have the same right backbone endpoint, and P_i's left backbone endpoint is to the left of P's left*

backbone endpoint.

Proof. Observe that a semi-caterpillar is a subgraph of a caterpillar. Therefore, the proof of Lemma 11 can be easily adapted to prove this lemma. □

We can then extend Theorem 2 to spiders of caterpillars.

Theorem 3. Multicut in Spiders of Caterpillars *has a problem kernel which consists of a spider of caterpillars containing at most $O(k^{2k+1})$ nodes.*

Proof. For each semi-caterpillar of a spider of caterpillars T, after computing a maximum cardinality set $\mathcal{P}$ of edge-disjoint demand paths as described in Lemma 12, we can bound from above the size of this semi-caterpillar by $O(k^{2l})$ with $l = |\mathcal{P}|$ by using the arguments in the proof of Theorem 2. Note that, since a semi-caterpillar does not have the symmetrical structure of a caterpillar, we can only give the bounds for each set $A_1, A_2, \ldots, A_{2l-1}$ one-by-one from A_1 to A_{2l-1}. Therefore, $|I_2 \cup L_2| = O(k^{2k})$.

It remains to give a bound on $|L_3|$. Now, let u denote the only I_3-node. By Lemma 6, the number of L_3-leaves that have a demand path of length 2 starting at them is bounded by $2k^2$. Thus, we omit such L_3-leaves from further consideration. At each of the remaining L_3-leaves starts at least one demand path which ends at one node of the semi-caterpillars of T. From the Overloaded L_3-Leaves rule we know that at an arbitrary node v, there can start at most k demand paths which end at some L_3-leaves of u. Thus, with $|L_1| \leq 2k^2$, $|I_1| \leq k$, and $|I_2 \cup L_2| = O(k^{2k})$, we have

$$|L_3| \leq k \cdot |I_1 \cup I_2 \cup L_1 \cup L_2| = O(k^{2k+1}).$$

Altogether, the size of T is bounded by $O(k^{2k+1})$ and we have the claimed problem kernel size. □

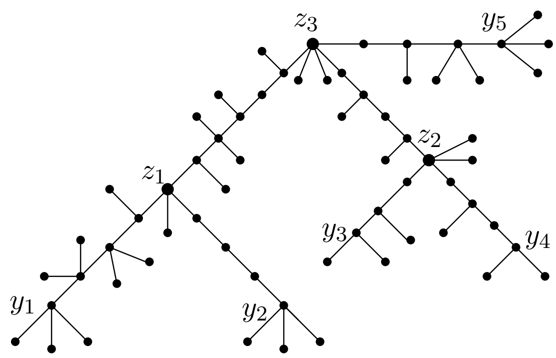

Figure 4: An example of a general tree: $I_1 = \{y_1, \ldots, y_5\}$ and $I_3 = \{z_1, z_2, z_3\}$. The tree is rooted at z_3. Node z_1 has maximum depth among I_3-nodes.

4.6 Problem Kernel for General Trees

Based on the results of Sections 4.4 and 4.5, we have now all results and techniques in place to develop a problem kernel for MULTICUT IN TREES. For general trees T, there can be more than one I_3-node. We assume that there is at least one I_3-node in T and root T at an arbitrary I_3-node. Consider an I_3-node having maximum depth in the now rooted tree among all I_3-nodes. Observe that this I_3-node together with all adjacent maximal caterpillar components which are not adjacent to any other I_3-nodes, i.e., the subtree of T rooted at this I_3-node, induces a structure similar to a spider of caterpillars. With this observation, we process all I_3-nodes in a bottom-up manner and, for each I_3-node, we give a bound on the size of the subtree rooted at it. See Figure 4 for an example.

Theorem 4. MULTICUT IN TREES *has a problem kernel which consists of a tree containing at most $O(k^{3k})$ nodes.*

Proof. First, consider an I_3-node u with maximum depth among all I_3-

nodes, for instance, z_1 in Figure 4. The subtree of T rooted at u, denoted by $T[u]$, can be seen as a spider of caterpillars with u as the center node. Moreover, each L_3-leaf of u has at least one path starting at it and ending at a node of $T[u]$ (Lemma 9). Following from the analysis in Section 4.5,

$$|T[u]| = O(k^{2l_u+1}), \tag{3}$$

where l_u denotes the number of maximum edge-disjoint demand paths using only edges of $T[u]$.

Then, consider the maximal caterpillar component C between u and its I_3-*parent* v, the first I_3-node on the path from u to the root. In Figure 4, z_3 is the I_3-parent of both z_1 and z_2. Recall that, in Section 4.5, we bounded the size of a maximal caterpillar component of a spider of caterpillars based on the fact that the caterpillar component is adjacent to an I_1-node that has at most $2k^2$ L_1-leaves, i.e., the maximal caterpillar component is in a semi-caterpillar. Here, the maximal caterpillar component C between u and v is not adjacent to any I_1-node. However, the analysis in the proof of Theorem 2 can be easily extended to deal with a caterpillar component adjacent to an I_3-node that is the root of a subtree with bounded size. We can treat C as a caterpillar component adjacent to an I_1-node with as many L_1-leaves as the size of the subtree. Then, we partition the nodes of C as in the proof of Theorem 2 into $A_1, A_2, \ldots, A_{2l_C-1}$, where l_C denotes the number of maximum edge-disjoint demand paths using only edges of C. The bound on the size of A_1 is then $k \cdot |T[u]|$, since each node in A_1 has to be the start node of at least one demand path ending at a node of $T[u]$ (Lemma 10). Then, $|A_1| \leq k \cdot |T[u]|$, $|A_2| \leq k \cdot (|A_1| + |T[u]|)$, and so on. With the size bound on $T[u]$, we have

$$|C| \overset{(3)}{=} O(k^{2l_u+1+2l_C}). \tag{4}$$

In the next step, we consider the subtree rooted at u's I_3-parent v, $T[v]$. Accordingly, we call all I_3-nodes that have v as their I_3-parent v's I_3-*children*, i.e., u is an I_3-child of v. Let $u_1, \ldots, u_s$ denote v's I_3-children with $u = u_1$. Then, subtree $T[v]$ can be divided in the following disjoint subtrees:

- the subtrees $T[u_1], \ldots, T[u_s]$ rooted at v's I_3-children,
- the caterpillar components $C_1, \ldots, C_s$ between v and its I_3-children,
- the caterpillar components $C'_1, \ldots, C'_r$ between v and the I_1-nodes that have v as their I_3-parent,
- and the star induced by v and its L_3-leaves.

In Figure 4, the tree T is rooted at z_3 which is the I_3-parent of z_1 and z_2. Here, the disjoint subtrees of T by dividing T at z_3 are the subtrees $T[z_1]$ and $T[z_2]$, the caterpillar components between z_3 and z_1 and between z_3 and z_2, the caterpillar component between z_3 and y_5, and the star consisting of z_3 and its L_3-leaves.

When arriving at v in the course of the bottom-up process, we have already the size bounds on all $T[u_i]$ and C_i for $1 \leq i \leq s$. In order to show that $T[v]$ has bounded size, it remains to give size bounds on $C'_1, \ldots, C'_r$ and the set of v's L_3-leaves, respectively. Since each of $C'_1, \ldots, C'_r$ with the adjacent I_1-node forms a semi-caterpillar, we have

$$|C'_1 \cup \cdots \cup C'_r| = O(k^{2l_{C'}}) \tag{5}$$

as shown in Section 4.5, where $l_{C'}$ denotes the number of edge-disjoint demand paths using only the edges of $C'_1, \ldots, C'_r$.

Let L_3^v denote the set of v's L_3-leaves. By Lemma 9, each L_3^v-leaf has at least one demand path starting at it and ending at one node in $T[v] \setminus (L_3^v \cup \{v\})$. Thus, using the Overloaded L_3-Leaves rule, we get

$$|L_3^v| \leq k \cdot |T[v] \setminus (L_3^v \cup \{v\})|. \tag{6}$$

Furthermore, $T[v] \setminus (L_3^v \cup \{v\})$ is the union of $T[u_1], \ldots, T[u_s]$, $C_1, \ldots, C_s$, and $C'_1, \ldots, C'_r$. Let $l_1 = l_{u_1} + l_{u_2} + \cdots + l_{u_s}$ denote the number of edge-disjoint demand paths passing only the edges of $T[u_1], \ldots, T[u_s]$, and let $l_2 = l_{C_1} + l_{C_2} + \cdots + l_{C_s}$ denote the number of edge-disjoint demand paths passing only the edges of $C_1, \ldots, C_s$. We have

$$\begin{aligned}
|T[v]| &= \sum_{i=1}^{s} |T[u_i]| + \sum_{i=1}^{s} |C_i| + \sum_{j=1}^{r} |C'_j| + |L_3^v| + 1 \\
&\overset{(6)}{\leq} (k+1) \cdot \Big(\sum_{i=1}^{s} |T[u_i]| + \sum_{i=1}^{s} |C_i| + \sum_{j=1}^{r} |C'_j|\Big) + 1 \\
&\overset{(3),(4),(5)}{=} (k+1) \cdot (O(k^{2l_1+1}) + O(k^{2l_1+2l_2+1}) + O(k^{2l_{C'}})) + 1 \\
&= O(k^{2l_v+2}),
\end{aligned} \tag{7}$$

where l_v denotes the number of edge-disjoint demand paths in $T[v]$ and $l_v \geq l_1 + l_2 + l'_C$.

Finally, at the root r of T, we have then $l_r \leq k$. Starting from an I_3-node with maximum distance to the root r of T during the bottom-up process, we can encounter at most k I_3-nodes. Therefore, at the root r, we get $|T[r]| \overset{(7)}{=} O(k^{2l_r+k}) = O(k^{3k})$. This gives the claimed problem kernel size. □

By using the interleaving technique introduced in [20], we get a second fixed-parameter algorithm for Multicut in Trees.

Theorem 5. MULTICUT IN TREES *can be solved in* $O(k^{3k} + m^3 n + n^3 m)$ *time, where* k *denotes the maximum number of tree edges that may be removed.*

5 Conclusion

We obtained the first fixed-parameter tractability result for (unweighted) MULTICUT IN TREES with respect to the parameter "solution size," the most immediate parameterization of this problem, by giving a bounded search tree algorithm and a problem kernel. We mention in passing that if the value of the parameter k is not given in advance (that is, in fact, dealing with the original optimization problem), then by simple binary search starting with $k = 0, 1, 2, 4, 8, \ldots$ and so on we can find the optimal k value at the cost of an additional factor $O(\log k)$ for the running time. Our result complements previous work [13] mainly dealing with the polynomial-time approximability of this problem. We claim that our exact algorithm is conceptually simple enough to be worth implementing. In particular, we feel that our data reduction rules for MULTICUT IN TREES are so natural that they probably should be combined with every algorithmic approach tackling this problem, also including approximation algorithms.

Clearly, the immediate challenge is to improve the search tree and problem kernel size significantly. We felt that the latter will be a particularly hard task when only using the given set of data reduction rules. Among others, it is a long-standing open problem to improve the approximation factor for MULTICUT IN TREES to a constant smaller than two since it would directly imply the same improvement for VERTEX COVER, a problem that has been open for more than two decades. In parallel, we can ask whether the search tree size for Multicut in Trees can be made smaller than 2^k, a perhaps simpler question to answer. Note that the search tree for Vertex Cover is below 1.3^k [6, 21] but Multicut in Trees is the more general problem. Finally, we aim at implementations of and experiments with our algorithms. To this end, experiences with problem instances drawn from practical applications would be of high interest.

Acknowledgment. We thank Falk Hüffner (Friedrich-Schiller-Universität Jena), Douglas R. Shier (Networks), and two anonymous referees for useful hints improving the presentation.

References

[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons, Kernelization algorithms for the vertex cover problem: theory and experiments, Proc of ACM-SIAM ALENEX'04, 2004, pp. 62–69.

[2] J. Alber, N. Betzler, and R. Niedermeier, Experiments on data reduction for optimal domination in networks, Proc of International Network Optimization Conference, 2003, pp. 1–6.

[3] J. Alber, M. R. Fellows, and R. Niedermeier, Polynomial-time data reduction for dominating set, J ACM 51 (2004), 363–384.

[4] R. S. Anand, T. Erlebach, A. Hall, and S. Stefanakos, Call control with k rejections, J Comput Syst Sci 67 (2003), 707–722.

[5] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows, Advice classes of parameterized tractability, Ann Pure Appl Logic 84 (1997), 119–138.

[6] J. Chen, I. A. Kanj, and W. Jia, Vertex cover: further observations and further improvements, J Algorithms 41 (2001), 280–301.

[7] G. Călinescu, C. G. Fernandes, and B. Reed, Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width, J Algorithms 48 (2003), 333–359.

[8] M. Costa, L. Létocart, and F. Roupin, Minimal multicut and maximal integer multiflow: a survey, European Journal of Operational Research 162 (2004), 55–69.

[9] R. G. Downey, Parameterized complexity for the skeptic, Proc 18th IEEE Annual Conference on Computational Complexity, 2003, pp. 147–169.

[10] R. G. Downey and M. R. Fellows, Parameterized complexity, Monographs in Computer Science, Springer-Verlag, New York, 1999.

[11] M. R. Fellows, Blow-ups, win/win's, and crown rules: some new directions in FPT, Proc 29th WG, LNCS 2880, 2003, pp. 1–12.

[12] M. R. Fellows, New directions and new challenges in algorithm design and complexity, parameterized, Proc 8th WADS, LNCS 2748, 2003, pp. 505–519.

[13] N. Garg, V. V. Vazirani, and M. Yannakakis, Primal-dual approximation algorithms for integral flow and multicut in trees, Algorithmica 18 (1997), 3–30.

[14] J. Guo and R. Niedermeier, Exact algorithms for tree-like weighted set cover, Manuscript, April 2005, submitted for publication to Journal of Discrete Algorithms.

[15] J. Hromkovič, Algorithmics for hard problems, 2nd Edition, Springer-Verlag, Berlin, 2002.

[16] E. Kenar and J. Uhlmann, Multicut in graphs, Manuscript, WSI für Informatik, Universität Tübingen, Feb. 2005.

[17] B. Monien, The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete, SIAM J. Algebraic Discrete Methods 7 (1986), 505–512.

[18] R. Niedermeier, Ubiquitous parameterization—invitation to fixed-parameter algorithms, Proc 29th MFCS, LNCS 3153, 2004, pp. 84–103.

[19] R. Niedermeier, Invitation to fixed-parameter algorithms, Oxford University Press, forthcoming, 2005.

[20] R. Niedermeier and P. Rossmanith, A general method to speed up fixed-parameter-tractable algorithms, Inf Process Lett 73 (2000), 125–129.

[21] R. Niedermeier and P. Rossmanith, On efficient fixed-parameter algorithms for weighted vertex cover, J Algorithms 47 (2003), 63–77.

[22] K. Weihe, Covering trains by stations or the power of data reduction, Proc 1st Workshop on Algorithms and Experiments, 1998, pp. 1–8. http://rtm.science.unitn.it/alex98/proceedings.html.

[23] K. Weihe, On the differences between "practical" and "applied" (invited paper), Proc 4th WAE, LNCS 1982, 2000, pp. 1–10.

[24] G. J. Woeginger, Exact algorithms for NP-hard problems: a survey, Proc 5th International Workshop on Combinatorial Optimization, LNCS 2570, 2003, pp. 185–208.