Multicut viewed through the eyes of vertex cover

Jianer Chen* Jiahao Fan† Iyad A. Kanj‡ Yang Liu§ Fenghui Zhang¶

Abstract

We take a new look at the MULTICUT problem in trees through the eyes of the VERTEX COVER problem. This connection, together with other techniques that we develop, allows us to significantly improve the $O(k^6)$ upper bound on the kernel size for MULTICUT, given by Bousquet et al., to $O(k^3)$. We exploit this connection further to present a parameterized algorithm for MULTICUT that runs in time $O^*(\rho^k)$, where $\rho = (\sqrt{5}+1)/2 \approx 1.618$. This improves the previous (time) upper bound of $O^*(2^k)$, given by Guo and Niedermeier, for the problem.

1 Introduction

In the MULTICUT problem in trees we are given a tree T, a set of requests $R \subseteq V(T) \times V(T)$ between pairs of vertices in T, and a nonnegative integer k, and we are asked to decide if we can remove at most k edges from the tree to disconnect all the requests in R (i.e., every path in the tree that corresponds to a request in R contains at least one of the removed edges). This problem is NP-hard, and its optimization version can be approximated to within ratio 2 [7].

We consider the MULTICUT problem from the parameterized complexity perspective. We mention that the parameterized complexity of several graph separation problems, including variants of the MULTICUT problem, was studied with respect to different parameters by Marx in [9]. Guo and Niedermeier [8] showed that the MULTICUT problem is fixed-parameter tractable by giving an $O^*(2^k)$ time algorithm for the problem. (The asymptotic notation $O^*(f(k))$ denotes time complexity of the form $f(k) \cdot p()$, where p is a polynomial in the input length.) They also showed that MULTICUT has an exponential-size kernel. Recently, Bousquet, Daligault, Thomassé, and Yeo, improved the upper bound on the kernel size for MULTICUT to $O(k^6)$ [3].

In this paper we take a new look at MULTICUT through the eyes of the VERTEX COVER problem. This connection allows us to give an upper bound of $O(k^3)$ on the kernel size for MULTICUT, significantly improving the previous $O(k^6)$ upper bound given by Bousquet et al. [3]. We exploit this connection further to give a parameterized algorithm for MULTICUT that runs in $O^*(\rho^k)$ time, where $\rho = (\sqrt{5}+1)/2 \approx 1.618$ (golden ratio) is the positive root of the polynomial $x^2 - x - 1$, thus improving the $O^*(2^k)$ time algorithm, given by Guo and Niedermeier [8]. To obtain the $O(k^3)$ upper bound on the kernel size, we first group the vertices in the tree such that no request exists between any two vertices within the same group. We then introduce an ordering that orders the

*Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843.chen@cs.tamu.edu

†Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843. grantfan@cs.tamu.edu

‡School of Computing, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604. ikanj@cs.depaul.edu.

§Department of Computer Science, University of Texas-Pan American, Edinburg, TX 78539. yliu@cs.panam.edu

¶Google Kirkland, 747 6th Street South, Kirkland, WA 98033. fhzhang@gmail.com

leaves in a group with respect to every other group. This ordering allows us to introduce a set of reduction rules that limits the number of leaves in a group that have requests to the vertices in another group. At the core of this set of reduction rule is a rule that utilizes the crown kernelization algorithm for VERTEX COVER [1]. All the above allows us to upper bound the number of leaves in the reduced instance by $O(k^2)$, improving the $O(k^4)$ upper bound on the number of leaves obtained in [3]. Finally, we show that the size of the reduced instance is at most the number of leaves in the reduced instance multiplied by a linear factor of k, thus yielding an upper bound of $O(k^3)$ on the size of the kernel. To obtain the $O^*(((\sqrt{5}+1)/2)^k)$ time algorithm, we first establish structural connections between MULTICUT and VERTEX COVER that allow us to simplify the instance of MULTICUT. We then exploit the simplified structure of the resulting instance to present a simple search-tree algorithm for MULTICUT that runs in time $O^*(((\sqrt{5}+1)/2)^k)$. We note that, even though some connection between MULTICUT and VERTEX COVER was observed in [7, 8], this connection was not developed or utilized in kernelization algorithms, nor in parameterized algorithms for MULTICUT.

We mention that, very recently, the multicut problem on general graphs was shown to be fixed-parameter tractable independently by Bousquet, Daligault, and Thomassé [2], and by Marx and Razgon [10], thus answering a long-standing open problem in parameterized complexity theory.

2 Preliminaries

We assume familiarity with the basic notations and terminologies about graphs and parameterized complexity. For more information, we refer the reader to [5, 6, 11, 12].

2.1 Graphs, forests, trees, and caterpillars

For a graph H we denote by $V(H)$ and $E(H)$ the set of vertices and edges of H, respectively; $n(H) = |V(H)|$ and $e(H) = |E(H)|$ are the number of vertices and edges in H. For a set of vertices $S \subseteq V(H)$, we denote by $H[S]$ the subgraph of H induced by the set of vertices in S. For a vertex $v \in H$, $H - v$ denotes $H[V(H) \setminus \{v\}]$, and for a subset of vertices $S \subseteq V(H)$, $H - S$ denotes $H[V(H) \setminus S]$. By *removing* a subgraph H' of H we mean removing $V(H')$ from H to obtain $H - V(H')$. Two vertices u and v in H are said to be *adjacent* or *neighbors* if $uv \in E(H)$. For two vertices $u, v \in V(H)$, we denote by $H - uv$ the graph $(V(H), E(H) \setminus \{uv\})$, and by $H + uv$ the *simple* graph $(V(H), E(H) \cup \{uv\})$. By *removing* an edge uv from H we mean setting $H = H - uv$. For a subset of edges $E' \subseteq E(H)$, we denote by $H - E'$ the graph $(V(H), E(H) \setminus E')$. For a vertex $v \in H$, $N(v)$ denotes the set of neighbors of v in H. The *degree* of a vertex v in H, denoted $deg_H(v)$, is $|N(v)|$. The *degree* of H, denoted $\Delta(H)$, is $\Delta(H) = \max\{deg_H(v) : v \in H\}$. The *length* of a path in a graph H is the number of edges in it. A *matching* in a graph is a set of edges such that no two edges in the set share an endpoint. A *vertex cover* for a graph H is a set of vertices such that each edge in H is incident to at least one vertex in this set. A vertex cover for H is *minimum* if its cardinality is minimum among all vertex covers of H; we denote by $\tau(H)$ the cardinality/size of a minimum vertex cover of H.

A *tree* is a connected acyclic graph. A *leaf* in a tree is a vertex of degree at most 1. A nonleaf vertex in a tree is called an *internal* vertex. The *internal degree* of a vertex v in a tree is the number of nonleaf vertices in $N(v)$. For two vertices u and v, the *distance* between u and v in T, denoted $dist_T(u, v)$, is the length of the unique path between u and v in T. A leaf x in a tree is said to be *attached* to vertex u if u is the unique neighbor of x in the tree. A *caterpillar* is a tree consisting of a path with leaves attached to the vertices on the path. A *forest* is a collection of disjoint trees.

Let T be a tree with root r. For a vertex $u \neq r$ in $V(T)$, we denote by $\pi(u)$ the parent of u in T. A *sibling* of u is a child $v \neq u$ of $\pi(u)$ (if exists), an *uncle* of u is a sibling of $\pi(u)$, and a *cousin* of u is a child of an uncle of u. A vertex v is a *nephew* of a vertex u if u is an uncle of v. For a vertex $u \in V(T)$, T_u denotes the subtree of T rooted at u. The *children* of a vertex u in $V(T)$, denoted $children(u)$, are the vertices in $N(u)$ if $u = r$, and in $N(u) - \pi(u)$ if $u \neq r$. A vertex u is a *grandparent* of a vertex v if $\pi(v)$ is a child of u. A vertex v is a *grandchild* of a vertex u if u is a grandparent of v.

2.2 Parameterized complexity

A *parameterized problem* is a set of instances of the form (x, k), where $x \in \Sigma^*$ for a finite alphabet set Σ, and k is a non-negative integer called the *parameter*. A parameterized problem Q is *fixed parameter tractable*, or simply FPT, if there exists an algorithm that on input (x, k) decides if (x, k) is a yes-instance of Q in time $f(k)n^{O(1)}$, where f is a recursive function independent of $n = |x|$. A parameterized problem Q is *kernelizable* if there exists a polynomial-time reduction that maps an instance (x, k) of Q to another instance (x', k') of Q such that: (1) $|x'| \leq g(k)$ for some recursive function g, (2) $k' \leq k$, and (3) (x, k) is a yes-instance of Q if and only if (x', k') is a yes-instance of Q. The instance (x', k') is called the *kernel* of (x, k).

2.3 Multicut in trees and forests

Let $\mathcal{F}$ be a forest. A *request* is a pair (u, v), where $u, v \in V(\mathcal{F})$. Let R be a set of requests. A subset of edges $E' \subseteq E(\mathcal{F})$ is said to be an *edge cut*, or simply a *cut*, for R if for every request (u, v) in R, there is no path between u and v in $\mathcal{F} - E'$. The *size* of a cut E' is $|E'|$. A cut E' is *minimum* if its cardinality is minimum among all cuts. The multicut problem in trees is defined as follows: given a tree T, a set of requests $R \subseteq V(T) \times V(T)$, and a nonnegative integer parameter k, decide if there exists a cut for R of size at most k. Since some of the folklore operations performed on the tree end up cutting edges from the tree, researchers who work on the multicut problem consider a generalization of the problem to forests. We follow suit and define the following problem:

MULTICUT
Given: A forest $\mathcal{F}$ and a set of requests $R \subseteq V(\mathcal{F}) \times V(\mathcal{F})$
Parameter: k
Question: Is there a cut for R of size at most k?

Clearly, any request between two vertices in different trees in $\mathcal{F}$ can be discarded from R.

Let $(\mathcal{F}, R, k)$ be an instance of MULTICUT, and let uv be an edge in $E(\mathcal{F})$. If we know that edge uv can be included in the solution sought, then we can remove uv from $\mathcal{F}$ and decrement the parameter k by 1; we say in this case that we *cut* edge uv. By *cutting* a leaf we mean cutting the unique edge incident to it. If T is a rooted tree in $\mathcal{F}$ and $u \in T$ is not the root, we say that we *cut* u to mean that we cut the edge $u\pi(u)$. (Note that after cutting an edge uv that is not incident to a leaf we obtain two trees: one containing u and the other containing v. Obviously, any request in R that goes across the two trees can be discarded.) On the other hand, if we know that edge uv can be excluded from the solution sought, we say in this case that edge uv is *kept*, and we can *contract* it by identifying the two vertices u and v, i.e., removing u and v and creating a new vertex with neighbors $(N(u) \cup N(v)) \setminus \{u, v\})$. If edge uv is contracted and w is the new vertex, then any request in R of the form (u, x) or (v, x) is replaced by the request (w, x).

A leaf x in $\mathcal{F}$ is said to be *good* if there exists another leaf y such that x and y are attached to the same vertex in $\mathcal{F}$ and (x, y) is a request in R; otherwise, x is said to be a *bad* leaf.[1] We define an auxiliary graph for $\mathcal{F}$, denoted G for simplicity, as follows. The vertices of G are the good leaves in $\mathcal{F}$, and two vertices x and y in G are adjacent (in G) if and only if x and y are attached to the same vertex of $\mathcal{F}$ and there is a request between x and y in R. Without loss of generality, we shall call the vertices in G with the same names as their corresponding good leaves in $\mathcal{F}$, and it will be clear from the context whether we are referring to the good leaves in $\mathcal{F}$ or to their corresponding vertices in G. Note that there is no edge in G between two good leaves that are attached to different vertices even though there could be a request between them. Therefore, G consists of isolated subgraphs, each is not necessarily connected and is induced by the set of good leaves that are attached to the same vertex in $\mathcal{F}$. For an internal vertex $u \in \mathcal{F}$ we denote by G_u the subgraph of G induced by the good leaves that are attached to u (if any).

It is not difficult to see that if a set of vertices C in G is a vertex cover for G then $E_C = \{uw \in E(\mathcal{F}) \mid w \in C\}$, which has the same cardinality as C, cuts all the requests between every two good leaves that are attached to the same vertex in $\mathcal{F}$. On the other hand, for any cut, the edges in the cut that are incident to the leaves in G is a vertex cover of G. It follows that the cardinality of the set of edges that are incident to the leaves in G in a minimum cut of $\mathcal{F}$ is at least the size of a minimum vertex cover for G.

3 The kernel

In this section we prove an upper bound of $O(k^3)$ on the kernel size for MULTICUT. Let $(\mathcal{F}, R, k)$ be an instance of MULTICUT, and let T be tree in $\mathcal{F}$. Two requests (u, v) and (p, q) in R are said to be *disjoint* if the path between u and v in $\mathcal{F}$ is edge-disjoint from the path between p and q in $\mathcal{F}$. A request (p, q) *dominates* a request (u, v) if the path from p to q in $\mathcal{F}$ is a subpath of the path from u to v in $\mathcal{F}$. The following reduction rules for MULTICUT are folklore, easy to verify, and can be implemented to run in polynomial time (see [3, 8] for proofs). Therefore, we omit their proofs.

Reduction Rule 3.1 (Useless edge) *If no request in R is disconnected by the removal of edge $uv \in E(\mathcal{F})$, then remove edge uv from $\mathcal{F}$.*

Reduction Rule 3.2 (Useless pair) *If $(u, v) \in R$ where u, v are in two different trees of $\mathcal{F}$, then remove (u, v) from R.*

Reduction Rule 3.3 (Unit request) *If $(u, v) \in R$ and $uv \in E(\mathcal{F})$, then remove uv from $\mathcal{F}$ and decrement k by 1.*

Reduction Rule 3.4 (Disjoint requests) *If there are $k+1$ pairwise disjoint requests in R, then reject the instance $(\mathcal{F}, R, k)$.*

Reduction Rule 3.5 (Unique direction) *Let x be a leaf or an internal degree-2 vertex in $\mathcal{F}$. If all the requests from x have the same direction (i.e., can be disconnected by the removal of a single edge from $\mathcal{F}$), then if x is a leaf then contract the edge incident to x, and if x is an internal degree-2 vertex then contract the edge incident to x that is not on any of the paths corresponding to the requests from x.*

[1] We note that we differ from the terminology used in [3]. What we call good leaves are called bad leaves in [3], and vice versa. The reason for calling them good leaves is that it is much easier to get an upper bound on their number that is not worse than the upper bound obtained on the bad leaves, which involves some quite sophisticated techniques.

Reduction Rule 3.6 (Domination/Inclusion) *If a request (p, q) dominates another request (u, v) then remove (u, v) from R.*

It was shown in [3] that the number of good leaves (called bad leaves there) is $O(k^2)$. We introduce a reduction rule next that allows us to derive the same upper bound on the number of good leaves in $\mathcal{F}$, and which uses Buss' kernelization algorithm for the VERTEX COVER problem [4] (this reduction rule was implicitly observed in [8]). (The VERTEX COVER problem is: Given a graph H and a parameter k, decide if there is a vertex cover for H of size at most k.) The reason for introducing this reduction is twofold: First to emphasize the importance of VERTEX COVER in kernelization algorithms for MULTICUT, and second because we shall use a different kernelization algorithm (crown reduction) later to bound the number of bad leaves that have requests to good leaves in the reduced instance. (Recall that the graph G is the graph whose vertices are the good leaves in $\mathcal{F}$ and whose edges correspond to the requests between good leaves that are attached to the same vertex in $\mathcal{F}$.)

Reduction Rule 3.7 (Bound on good leaves) *Apply Buss' kernelization algorithm for* VERTEX COVER *[4] to (G, k): for every vertex x in G whose degree (in G) is at least $k + 1$, cut leaf x in $\mathcal{F}$. If the number of remaining good leaves in $\mathcal{F}$ is more than $2k^2$, then reject the input instance $(\mathcal{F}, R, k)$.*

PROOF. A leaf corresponding to a vertex $x \in G$ of degree at least $k + 1$ must be cut, otherwise, at least $k + 1$ edges must be cut to disconnect all requests from x, and hence no solution of size at most k for MULTICUT exists.

By Buss' algorithm, if the resulting graph contains more than $2k^2$ nonisolated vertices (i.e., remaining good leaves) then G has no vertex cover of size at most k, and obviously $(\mathcal{F}, R, k)$ is a no-instance of MULTICUT. □

We shall assume henceforth that none of Reduction Rules 3.1 – 3.7 applies to $(\mathcal{F}, R, k)$. We shall also assume that isolated vertices are removed from $\mathcal{F}$ at all times.

The statements in the following lemma were shown in [3], and can be easily verified by the reader:

Lemma 3.1 *The following are true:*

a. *(Claim 5 in [3]) The number of internal vertices in $\mathcal{F}$ of internal degree 1 is at most k.*

b. *(Claim 6 in [3]) The number of internal vertices in $\mathcal{F}$ of internal degree at least 3 is at most k.*

(Part (a) is true because at least two good leaves must be attached to each such vertex by the unique direction reduction rule above, and by the disjoint requests rule, there can at most k such vertices. Part (b) is true because the number of such vertices is not more than the number of internal vertices of internal degree 1.)

We now define a grouping of the vertices in $\mathcal{F}$ into three types of groups.

Definition 3.1 A *type-I group* consists of an internal vertex u of $\mathcal{F}$ that has at least one good leaf attached to it, together with all the leaves (bad and good) that are attached to u; we say that vertex u *forms* the type-I group. A *type-II* group consists of an internal vertex u in $\mathcal{F}$ of internal degree at least 3 that does not have any good leaves attached to it, together with all the (bad)

leaves attached to u (if any); we say that vertex u *forms* the type-II group. A *type-III group* consists of the vertices (internal and leaves) of a caterpillar in $\mathcal{F}$ such that: (1) every internal vertex of the caterpillar has internal degree 2 in $\mathcal{F}$, and (2) there is no request between any two vertices (internal-internal, leaf-internal, nor leaf-leaf) of the caterpillar. Note that condition (2) implies that all the leaves attached to the internal vertices in a type-III group are bad leaves.

Lemma 3.2 *$V(F)$ can be partitioned in polynomial time into $O(k)$ type-I, type-II, and type-III groups.*

Proof. An internal vertex $u \in \mathcal{F}$ either has good leaves attached to it or does not. If it does, it forms with the leaves attached to it a type-I group. Note that every vertex of internal degree 1 in $\mathcal{F}$ must have good leaves attached to it by the unique direction rule, and hence every such vertex, together with all the leaves attached to it, forms a type-I group. If an internal vertex u does not have good leaves attached to it, then by the preceding statement, either u has internal degree at least 3, or u has internal degree 2 in $\mathcal{F}$. If u has internal degree at least 3, then u, together with all the (bad) leaves attached to it, forms a type-II group. Now every vertex that does not form a type-I group or a type-II group must be an internal vertex of $\mathcal{F}$ of internal degree exactly 2 such that all the leaves attached to it are bad leaves; let $\mathcal{F}'$ be the subgraph of $\mathcal{F}$ induced by the set of such vertices, together with the bad leaves attached to them. Clearly, $\mathcal{F}'$ consists of a collection of disjoint caterpillars, and note that every internal vertex in $\mathcal{F}'$ has internal degree 2 in $\mathcal{F}$. Each caterpillar in $\mathcal{F}'$ can be partitioned in a greedy fashion into vertex-disjoint subcaterpillars such that each subcaterpillar satisfies conditions (1) and (2) above, as follows. To partition a caterpillar, start from an internal vertex that is an endpoint of the caterpillar, and traverse the caterpillar towards the other endpoint as long as condition (2) above is not violated. The first time a vertex u is reached such that condition (2) is violated at u, or at one of the leaves attached to u, the subcaterpillar traversed so far up to the vertex before u forms a type-III group, and the process is repeated starting from u; the process stops when the other endpoint of the caterpillar is reached. Note that, for any two type-III groups in the same caterpillar in $\mathcal{F}'$ that were constructed consecutively in the above process, there exists a request between some vertex of the first group and another vertex in the other group. It follows from all the above that $V(\mathcal{F})$ can the be partitioned into groups, each belonging to one of the three types defined above.

The number of type-I groups in $(\mathcal{F}, R, k)$ is $O(k)$, and the number of type-II groups in $(\mathcal{F}, R, k)$ is $O(k)$. This can be seen as follows. There is at least one request between the leaves that are attached to a vertex that forms a type-I group. By the disjoint requests rule, there can be at most k type-I groups in $(\mathcal{F}, R, k)$. The fact that the number of type-II groups in $(\mathcal{F}, R, k)$ is $O(k)$ follows from part (b) of Lemma 3.1. Therefore, it suffices to show that the number of type-III groups constructed by the above process is $O(k)$. First, note that the number of caterpillars in $\mathcal{F}'$ is $O(k)$. This can be seen as follows. If we remove all the leaves of $\mathcal{F}$ and replace every caterpillar in $\mathcal{F}'$ by an edge, we obtain a forest whose vertices are the internal vertices of $\mathcal{F}$ that form the type-I and type-II groups, and whose edges correspond to the caterpillars in $\mathcal{F}'$ in a one-to-one fashion. Since the number of type-I and type-II groups is $O(k)$, it follows that the number of edges in the resulting forest, and hence the number of caterpillars in $\mathcal{F}'$ is $O(k)$. Now let C be a caterpillar in $\mathcal{F}'$. Note that, for any two type-III groups in the same caterpillar in $\mathcal{F}'$ that were constructed consecutively in the above process, there exists a request between some vertex of the first group and another vertex in the other group. Therefore, if C is partitioned into ℓ subcaterpillars, and hence ℓ type-III groups, then there are at least $\lfloor \ell/2 \rfloor$ many disjoint requests in C. It follows by the disjoint requests rule that there can be at most $2k$ (proper) subcaterpillars resulting from the above process. This, in addition to the fact that the number of caterpillars in $\mathcal{F}'$ is $O(k)$ (it is

possible that some caterpillars are not partitioned at all), implies that the total number of type-III groups constructed above is $O(k)$.

Clearly, the above grouping can be constructed in polynomial time. □

Definition 3.2 Let G_i be a type-I, type-II, or a type-III group. The *intergroup edges* of G_i are the edges in $\mathcal{F}$ with exactly one endpoint in G_i; the *intergroup degree* of G_i, denoted d_i, is the number of intergroup edges of G_i. Note that if G_i is a type-I or a type-II group, where u is the internal vertex in $\mathcal{F}$ that forms G_i, then d_i is the internal degree of u in $\mathcal{F}$. On the other hand, if G_i is a type-III group then $d_i = 2$. The *internal vertices of* G_i are the internal vertices of $\mathcal{F}$ that are in G_i. The *internal edges of* G_i are the edges between the internal vertices of G_i. Note that only type-III groups can have internal edges. The *leaves (resp. good/bad leaves) of* G_i are the leaves (resp. good/bad leaves) attached to the internal vertices of G_i.

Lemma 3.3

$$\sum_{G_i \text{ is a group}} d_i = O(k).$$

PROOF. Since there are $O(k)$ type-III groups by Lemma 3.2, each of intergroup degree 2, it suffices to show that the sum of the internal degrees of the vertices forming the type-I and type-II groups is $O(k)$. There are at most $O(k)$ type-I groups. Therefore, the sum of the intergroup degree of type-I groups whose internal vertices have degree 1 or 2 is $O(k)$. It can be easily verified (by a standard inductive proof) that the sum of the intergroup degrees of type-I and type-II groups whose internal vertices have degree at least 3 is at most three times the number of internal vertices of internal degree 1 in $\mathcal{F}$, which is $O(k)$ by part (a) of Lemma 3.1. □

We introduce next a reduction rule that is used to bound the number of bad leaves that have requests to good leaves. We apply the crown reduction kernelization algorithm, described in [1], to the instance (G, k) of VERTEX COVER. This algorithm partitions $V(G)$ into three sets I, H, and O, such that: (1) I is an independent set of G, and no edge exists between the vertices in I and those in O, (2) there exists a minimum vertex cover of G containing H, (3) there exists a matching M that matches every vertex in H to a vertex in I, and (4) $|O| \leq 3k$ if a solution to (G, k) exists [1].

Reduction Rule 3.8 (Crown reduction) *Apply the crown reduction algorithm to (G, k) to partition $V(G)$ into the three sets H, I, O. If $|O| > 3k$ or $|H| > k$, then reject the instance $(\mathcal{F}, R, k)$.*

PROOF. If $|O| > 3k$ or $|H| > k$, then there exists no vertex cover for G of size at most k, and hence no cut for R of size at most k. □

Consider G_u, where u is a vertex in $\mathcal{F}$ that forms a type-I group. Denote by H_u, I_u, O_u the intersection of H, I, O with $V(G_u)$, respectively. Clearly, the matching M matching H into I in G induces a matching M_u in G_u that matches H_u into I_u. Let OUT_u be the set of vertices in I_u that are not matched by M_u (i.e., $I_u \setminus V(M_u)$). We have the following lemma:

Lemma 3.4 *Let u be a vertex in $\mathcal{F}$ that forms a type-I group. Any vertex cover of G_u that contains ℓ vertices from OUT_u has size at least $\tau(G_u) + \ell$.*

PROOF. The above statement is true because (1) any vertex cover of G_u contains at least $|H_u|$ vertices from $V(M_u)$, and (2) there is a minimum vertex cover of G_u that contains H_u (and hence excludes all the vertices in I_u). Therefore, if a vertex cover of G_u contains ℓ vertices from OUT_u, then it must contain at least $|H_u| + \ell$ vertices from $H_u \cup I_u$, and hence the size of such a vertex cover must be at least $\tau(G_u) + \ell$. □

Corollary 3.5 *Let u be a vertex in $\mathcal{F}$ that forms a type-I group G_i. If $(\mathcal{F}, R, k)$ has a solution, then it has a solution that cuts at most $d_i - 1 = d_u - 1$ leaves from OUT_u, where d_u is the internal degree of u in $\mathcal{F}$.*

PROOF. If S is a solution to $(\mathcal{F}, R, k)$ that cuts at least $d_i = d_u$ leaves from OUT_u, then we can remove all the edges in S that are incident to the leaves in G_i, and replace them with the edges that are incident to the leaves corresponding to a minimum vertex cover of G_u plus the d_i intergroup edges of G_i. □

Lemma 3.6 *If there exists a solution to the instance $(\mathcal{F}, R, k)$, then there exists a solution S to $(\mathcal{F}, R, k)$ such that, for any group G_i: if G_i is a type-I or a type-II group then S cuts at most $d_i - 1$ bad leaves of G_i, and if G_i is a type-III group then the number of bad leaves and internal edges of G_i that are cut by S is at most $d_i - 1 = 1$.*

PROOF. Suppose that the instance $(\mathcal{F}, R, k)$ has a solution S, and let G_i be a group. If G_i is a type-I or a type-II group such that S cuts at least d_i bad leaves from G_i, then the edges in S that are incident to all the bad leaves of G_i can be replaced with the intergroup edges of G_i. Similarly, if G_i is a type-III group such that the number of edges in S that are incident to bad leaves, or are internal edges of G_i, is at least $d_i = 2$, then those edges in S can be replaced with the two intergroup edges of G_i. By performing the above replacement for every group G_i, we obtain a solution S that satisfies the statement of the lemma. □

Next, we introduce reduction rules to bound the number of bad leaves in $(\mathcal{F}, R, k)$. The main idea behind these reduction rules is to use several orderings (defined later) on the set bad leaves of a group G_i w.r.t. to another group G_j, to limit the number of bad leaves of G_i that have requests to bad leaves or vertices of G_j to at most $d_i \times d_j$. For a leaf x of a group G_i, we shall refer to the internal vertex in G_i that x is attached to by $\nu(x)$.

Reduction Rule 3.9 (**Bound on the number of bad leaves in a group that have requests to a certain vertex**)
Let x be a vertex, and let G_i be a group. If there are at least d_i bad leaves in G_i that have requests to x, then let L_x be the list containing the bad leaves in G_i that have requests to x sorted in a nondecreasing order of their distance to x, where ties are broken arbitrarily. For every bad leaf z in G_i whose rank in L_x is at least d_i, replace the request (z, x) in R with the request $(\nu(z), x)$.

PROOF. Suppose that the above reduction rule applies to a group G_i in $(\mathcal{F}, R, k)$ and some vertex x, and let $(\mathcal{F}, R', k)$ be the resulting instance. Clearly, any solution to $(\mathcal{F}, R', k)$ is also a solution to $(\mathcal{F}, R, k)$. Therefore, it suffices to prove that if there exists a solution for $(\mathcal{F}, R, k)$ then there also exists a solution for $(\mathcal{F}, R', k)$. Suppose that there is a solution to $(\mathcal{F}, R, k)$. By Lemma 3.6, we can assume that there is a solution S that cuts at most $d_i - 1$ bad leaves from G_i. It follows from the preceding statement that if y is the the bad leaf whose rank in L_x is d_i, then S must cut an edge on the path between $\nu(y)$ and x (otherwise S would cut the first d_i bad leaves in L_x). Consequently, any request (z, x) from a bad leaf z whose rank in L_x is at least d_i that was replaced with the request $(\nu(z), x)$ is cut by S, and S is a solution to $(\mathcal{F}, R', k)$. □

Definition 3.3 Let G_i and G_j be two distinct groups. If x is bad leaf in G_i that has a request to at least one internal vertex in G_j, then among all internal vertices in G_j that x has requests to, we define the *vertex-offset* of x with respect to G_j, denoted *v-offset*$_j(x)$, to be *the* vertex of minimum distance to x. If x is a bad leaf in G_i that has a request to at least one bad leaf in G_j, then among

all bad leaves in G_j that x has requests to, we define the *leaf-offset* of x with respect to G_j, denoted *l-offset*$_j(x)$, to be *a* leaf of minimum distance to x (note that there could be several such leaves). Let u be the vertex in G_j with the minimum distance to the vertices in G_i (i.e., the vertex in G_j that is "closest" to G_i). We define an order $\preceq_j^v$ on the set of vertex-offsets of the bad leaves in G_i that have requests to internal vertices in G_j as follows. For two vertex-offsets y and y' of two bad leaves in G_i, $y \preceq_j^v y'$ if and only if the distance from u to y is smaller or equal to the distance from u to y'. The $\preceq_j^v$ order on the vertex-offsets in G_j of the bad leaves in G_i having requests to internal vertices in G_j induces an order $\preceq_j^v$ on the bad leaves in G_i in a natural way: for two bad leaves x and x' in G_i that have requests to internal vertices in G_j: $x \preceq_j^v x'$ if and only if v-offset$_j(x) \preceq_j^v$ v-offset$_j(x')$. Similarly, we can define an order $\preceq_j^l$ on the set of leaf-offsets of the bad leaves in G_i, which induces an order $\preceq_j^l$ on the bad leaves in G_i (that have requests to bad leaves in G_j) as follows: for two bad leaves x and x' in G_i that have requests to bad leaves in G_j: $x \preceq_j^l x'$ if and only if l-offset$_j(x) \preceq_j^v$ l-offset$_j(x')$. [2]

Reduction Rule 3.10 (**Bound on the number of bad leaves in a group that have requests to internal vertices in another group**)
Let G_i and G_j be two distinct groups. If there are at least d_i bad leaves in G_i that have requests to internal vertices of G_j, then consider all bad leaves in G_i that have requests to internal vertices of G_j, and sort them in a non-decreasing order with respect to the order $\preceq_j^v$; let L_i be the sorted list. For every bad leaf x in L_i whose rank in L_i is at least d_i, replace every request (x, p) in R from x to an internal vertex p of G_j with the request $(\nu(x), p)$.

Proof. Suppose that the above reduction rule applies to two groups G_i and G_j in $(\mathcal{F}, R, k)$, and let $(\mathcal{F}, R', k)$ be the resulting instance. Clearly, any solution to $(\mathcal{F}, R', k)$ is also a solution to $(\mathcal{F}, R, k)$. Therefore, it suffices to prove that if there exists a solution for $(\mathcal{F}, R, k)$ then there also exists a solution for $(\mathcal{F}, R', k)$. Suppose that there is a solution to $(\mathcal{F}, R, k)$. By Lemma 3.6, we can assume that there is a solution S such that if G_i is a type-I or a type-II group then S cuts at most $d_i - 1$ bad leaves of G_i, and if G_i is a type-III group then the number of bad leaves and internal edges of G_i that are cut by S is at most 1.

Let x be a bad leaf in L_i whose rank in L_i is at least d_i. If S does not cut x, then S must cut an edge on the path between $\nu(x)$ and the vertex-offset of x w.r.t. G_j, v-offset$_j(x)$, and hence, every request from x to an internal vertex p of G_j that was replaced with $(\nu(x), p)$ is cut by S. Suppose now that S cuts x, and note that if G_i is a type-III group then because S cuts x, S does not cut *any* bad leaf of G_i other than x, nor does it cut an internal edge of G_i. Therefore, if G_i is a type-III group, then since no internal edge of G_i is cut by S, we may contract all the internal edges of G_i to obtain a single internal vertex, say w, such that the leaves attached to w are precisely the leaves of G_i. If G_i is a type-I or a type-II group, also denote by w the internal vertex that determines group G_i. By our assumption, S cuts at most $d_i - 1$ bad leaves that are attached to w. We claim that S must cut an edge on the path between w and v-offset$_j(x)$. Suppose not, then since there are $d_i - 1$ leaves that appear before x in L_i, S must cut the first $d_i - 1$ leaves that appear before x in L_i; this is true because the vertex-offsets of these leaves w.r.t. G_j appear no later than that of x. Since S cuts x, it follows that S cuts d_i bad leaves of G_i, contradicting our assumption. It follows that S must cut an edge on the path between w and v-offset$_j(x)$. Since S does not cut any internal edge from G_i in case G_i is a type-III group, it follows that every request between x and an internal vertex p of G_j that was replaced with $(\nu(x), p)$ is cut by S.

[2] Clearly $\preceq_j^v$ and $\preceq_j^v$ are well-defined orders because they are defined based on the $\leq$ order.

Since x was arbitrarily chosen to be a leaf whose rank in L_i is at least d_i, S is a solution to $(\mathcal{F}, R', k)$. □

Reduction Rule 3.11 (**Bound on the number of bad leaves in a group that have requests to bad leaves in another group**)
Suppose that Reduction Rule 3.9 does not apply to $(\mathcal{F}, R, k)$. Let G_i and G_j be two distinct groups. If there are at least $(d_i - 1) \times d_j + 1$ bad leaves in G_i that have requests to bad leaves in G_j, then consider all the bad leaves in G_i that have requests to bad leaves of G_j, and sort them in a non-decreasing order with respect to the order $\preceq_j^l$; let L_i be the sorted list. For every bad leaf x in G_i whose rank in L_i is at least $(d_i - 1) \times d_j + 1$, replace every request (x, y) in R from x to a bad leaf y of G_j with the request $(\nu(x), y)$.

Proof. Suppose that the above reduction rule applies to two groups G_i and G_j in $(\mathcal{F}, R, k)$, and let $(\mathcal{F}, R', k)$ be the resulting instance. Clearly, any solution to $(\mathcal{F}, R', k)$ is also a solution to $(\mathcal{F}, R, k)$. Therefore, it suffices to prove that if there exists a solution for $(\mathcal{F}, R, k)$ then there also exists a solution for $(\mathcal{F}, R', k)$. Suppose that there is a solution to $(\mathcal{F}, R, k)$. By Lemma 3.6, we can assume that there is a solution S such that S cuts at most $d_i - 1$ bad leaves from G_i and at most $d_j - 1$ bad leaves from G_j, and that if G_i is a type-III group then the number of bad leaves and internal edges of G_i that are cut by S is at most 1.

Let x be a bad leaf in L_i whose rank in L_i is at least $(d_i - 1) \times d_j + 1$. If S does not cut x, then S must cut an edge on each path between $\nu(x)$ and a bad leaf in G_j that x has a request to, and hence, every request from x to a bad leaf y of G_j that is replaced with $(\nu(x), y)$ is cut by S. Suppose now that S cuts x, and note that if G_i is a type-III group then because S cuts x, S does not cut any bad leaf of G_i other than x, nor does it cut an internal edge of G_i. Therefore, if G_i is a type-III group, then since no internal edge of G_i is cut by S, we may contract all the internal edges of G_i to obtain a single internal vertex w, such that the leaves attached to w are precisely the leaves of G_i. If G_i is a type-I or a type-II group, also denote by w the internal vertex that determines group G_i. By our assumption, S cuts at most $d_i - 1$ bad leaves that are attached to w. We claim that S must cut an edge on the path from w to the internal vertex of G_j, v, that l-offset$_j(x)$ is attached to. Suppose not, then since there are at least $(d_i - 1) \times d_j + 1$ bad leaves that appear no later than x in L_i, and hence the internal vertices in G_j that the leaf-offsets of these bad leaves (w.r.t. G_j) are attached to, appear no later than v, all requests between these leaves and their leaf-offsets must be cut by leaves from G_i and G_j. By our assumption, S cuts at most $d_i - 1$ bad leaves from G_i. Moreover, since Reduction Rule 3.9 is not applicable, every bad leaf in G_j has requests to at most $d_i - 1$ bad leaves in G_i, and hence can cut the requests from at most $d_i - 1$ bad leaves in G_i. Since S cuts at most $d_j - 1$ bad leaves from G_j, those bad leaves can cut the requests from at most $(d_j - 1) \times (d_i - 1)$ bad leaves from G_i, and hence, the bad leaves in G_i and G_j that are in S can together cut the requests of at most $(d_i - 1) + (d_j - 1) \times (d_i - 1) = (d_i - 1) \times d_j$ bad leaves of G_i. This is a contradiction since there are at least $(d_i - 1) \times d_j + 1$ bad leaves in L_i that appear no later than x in L_i. It follows that S must cut an edge on the path between w and the internal vertex of G_j that l-offset$_j(x)$ is attached to. Since S does not cut any internal edge from G_i in case G_i is a type-III group, every request (x, y) between x and a bad leaf y of G_j that was replaced with $(\nu(x), y)$ is cut by S. Consequently, S is a solution to $(\mathcal{F}, R', k)$. □

Reduction Rule 3.12 (**Bound on the number of bad leaves in a group that have requests to good leaves in OUT_u for a type-I group G_j formed by vertex u**)
Suppose that Reduction Rule 3.9 does not apply to $(\mathcal{F}, R, k)$. Let u be a vertex such that u forms a type-I group G_j, and let $G_i \neq G_j$ be a group. If there are at least $d_j \times (d_i - 1) + 1$ many bad leaves

in G_i that have requests to leaves in OUT_u, let L_i be the list of bad leaves in G_i that have requests to vertices in OUT_u sorted in a non-decreasing order of their distance from u. For each bad leaf x in L_i whose rank is at least $d_j \times (d_i - 1) + 1$, replace every request (x, y) in R from x to a leaf y in OUT_u with the request $(\nu(x), y)$.

PROOF. Suppose that the above reduction rule applies to a group G_i and a type-I group G_j, formed by vertex u, in $(\mathcal{F}, R, k)$, and let $(\mathcal{F}, R', k)$ be the resulting instance. Clearly, any solution to $(\mathcal{F}, R', k)$ is also a solution to $(\mathcal{F}, R, k)$. Therefore, it suffices to prove that if there exists a solution for $(\mathcal{F}, R, k)$ then there also exists a solution for $(\mathcal{F}, R', k)$. Suppose that there is a solution to $(\mathcal{F}, R, k)$. Let S be a solution to $(\mathcal{F}, R, k)$. By Corollary 3.5, we can assume that S cuts at most $d_u - 1 = d_j - 1$ leaves from OUT_u. By Lemma 3.6, we can also assume that S cuts at most $d_i - 1$ bad leaves from G_i. Since Reduction Rule 3.9 is not applicable, every leaf in OUT_u has requests to at most $d_i - 1$ bad leaves in G_i. It follows from the above that the edges in S that are incident to leaves in OUT_u cut the requests of at most $(d_j - 1) \times (d_i - 1)$ bad leaves in L_i. Since at most $d_i - 1$ bad leaves in G_i are cut by S, it follows that there exists a bad leaf z in L_i of rank at most $d_j \times (d_i - 1) + 1$ that is not cut by S, and such that z has a request to a leaf in OUT_u that is not cut by S either. Therefore, S must cut an edge that is on the path from u to the internal vertex of G_i that the leaf in L_i of rank $d_j \times (d_i - 1) + 1$ is attached to, and consequently, S is a solution to $(\mathcal{F}, R', k)$. □

Definition 3.4 The instance $(\mathcal{F}, R, k)$ is said to be *reduced* if non of Reduction Rules 3.1 – 3.12 is applicable to it.

Lemma 3.7 *Let $(\mathcal{F}, R, k)$ be a reduced instance. The number of bad leaves in $\mathcal{F}$ that have requests to good leaves in $\mathcal{F}$ is $O(k^2)$.*

PROOF. Let G_i be a group with bad leaves. By Reduction Rule 3.9, every good leaf in G has requests to at most $d_i - 1$ bad leaves in G_i. Therefore, the number of bad leaves in G_i that have requests to good leaves in $O \cup V(M)$ is at most $|O \cup V(M)| \times (d_i - 1) < 5k \times d_i$, after noting that $|V(M)| = 2|H| \leq 2k$ and that $|O| \leq 3k$ (Reduction Rule 3.8). Thus, the number of bad leaves in $\mathcal{F}$ that have requests to good leaves in $O \cup V(M)$ is at most $5k \times \sum_{G_i} d_i = O(k^2)$ (by Lemma 3.3). Therefore, it suffices to show that the number of bad leaves in $\mathcal{F}$ that have requests to good leaves in OUT_u, over all type-I groups G_j formed by some vertex u, is $O(k^2)$.

In effect, let G_j be a type-I group formed by a vertex u. For every group $G_i \neq G_j$, by Reduction Rule 3.12, the number of bad leaves in G_i that have requests to good leaves in OUT_u is at most $d_i \times d_j$. Therefore, the total number of bad leaves in $\mathcal{F}$ that have requests to good leaves in OUT_u is at most $d_j \times \sum_{G_i} d_i = d_j \times O(k)$ (by Lemma 3.3). By summing over all type-I groups G_j, the number of bad leaves that have requests to good leaves is $O(k) \times \sum_{G_j \text{ is of type-I}} d_j = O(k) \times O(k) = O(k^2)$. □

Lemma 3.8 *Let $(\mathcal{F}, R, k)$ be a reduced instance. The number of leaves in $\mathcal{F}$ is $O(k^2)$.*

PROOF. By Reduction Rule 3.7, the number of good leaves in $\mathcal{F}$ is $O(k^2)$. Therefore, it suffices to show that the number of bad leaves in $\mathcal{F}$ is $O(k^2)$ as well.

Every bad leaf appears in some group G_i, and must have a request to a good leaf, an internal vertex, or a bad leaf of another group G_j. By Lemma 3.7, the number of bad leaves in $\mathcal{F}$ that have requests to good leaves is $O(k^2)$. Next, we bound the number of bad leaves in G_i that have requests to an internal vertex or to a bad leaf in another group G_j.

Fix a group $G_j \neq G_i$. By Reduction Rule 3.10, the number of bad leaves in G_i that have requests to internal vertices of G_j is less than d_i. Therefore, the total number of bad leaves in G_i that have requests to internal vertices in $\mathcal{F}$ is $\sum_{G_j \neq G_i} d_i = O(k) \times d_i$ (since the number of groups is $O(k)$ by Lemma 3.2). Summing over all groups G_i, we obtain that the total number of bad leaves in $\mathcal{F}$ that have requests to internal vertices in $\mathcal{F}$ is $O(k) \times \sum_{G_i} d_i = O(k) \times O(k) = O(k^2)$. By Reduction Rule 3.11, the number of bad leaves in G_i that have requests to bad leaves in G_j is at most $d_i \times d_j$. Therefore, the number of bad leaves in $\mathcal{F}$ that have requests to bad leaves in G_i is at most $d_i \times \sum_{G_j \neq G_i} d_j = d_i \times O(k)$. Summing over all groups G_i, we obtain that the number of bad leaves in $\mathcal{F}$ that have requests to bad leaves in $\mathcal{F}$ is $O(k) \times \sum_{G_i} d_i = O(k) \times O(k) = O(k^2)$. □

Lemma 3.9 *Let $(\mathcal{F}, R, k)$ be a reduced instance. The number of vertices in $\mathcal{F}$ that are not internal degree-2 vertices is $O(k^2)$.*

Proof. Let Y be the set of vertices in $\mathcal{F}$ that are not internal degree-2 vertices. A vertex in Y is either a leaf, a vertex with leaves attached to it, or an internal vertex of degree at least 3 (note that every vertex in $\mathcal{F}$ with internal degree 1 must have leaves attached to it). By Lemma 3.8, the number of leaves in $\mathcal{F}$ is $O(k^2)$, which also implies that the number of vertices in $\mathcal{F}$ that have leaves attached to them is $O(k^2)$. By part (b) of Lemma 3.1, the number of internal vertices in $\mathcal{F}$ of internal degree at least 3 is $O(k)$. It follows that $|Y| = O(k^2)$. □

Lemma 3.10 *Let $(\mathcal{F}, R, k)$ be a reduced instance. The number of internal degree-2 vertices in $\mathcal{F}$ is $O(k^3)$.*

Proof. Let Z be the set of internal degree-2 vertices in $\mathcal{F}$, and let $Y = V(\mathcal{F}) - Z$. By Lemma 3.9, we have $|Y| = O(k^2)$; this will be used to show that $|Z| = O(k^3)$.

For every tree T in $\mathcal{F}$, pick an internal vertex r_T in T of internal degree 1 and root the tree T at r_T. The ancester/descendant relationship in every tree T of $\mathcal{F}$ becomes defined. We define the following auxiliary digraph D. The set of vertices of D is Z. We add edges to D as follows. For every two vertices u and v in Z, add a directed edge from u to v in D if: (1) the outdegree of u in D is 0, v is a descendent of u in a tree T in $\mathcal{F}$ (i.e., v is on the root-leaf path from r_T going through u), and there is a request between u and v (i.e., $(u, v) \in R$). It is clear from the definition that the outdegree of every vertex in D is at most 1. We show next that the indegree of every vertex in D is at most 1 as well. In effect, suppose that there exists a vertex v in D such that the indegree of v is at least 2, and let u and w be two vertices in D that have outgoing edges to v. From the definition of D, v is a descendent of both u and w, and hence, the three vertices u, w, v are on the same root-leaf path from the root of the tree T containing them; without loss of generality, suppose that u appears before w on this path (starting from the root r_T). Since (u, v) and (w, v) are requests in R, and since u, v, w are all internal degree-2 vertices, request (w, v) must dominate request (u, v). This contradicts the fact that the domination reduction rule does not apply to $\mathcal{F}$.

By the definition of D, edges of D go from ancestors to descendants in the trees of $\mathcal{F}$. Therefore, D is acyclic. Since the indegree and outdegree of every vertex in D is at most 1, D consists of a collection of disjoint paths (possibly of length 0, i.e., single vertices). We now bound the number of vertices in D, and hence in Z.[3] For a path P in D, we call the endpoint of P with outdegree 0 the *head* of P. Let $\mathcal{P}$ be the set of all paths in D. Define the function ϕ from $\mathcal{P}$ to Y as follows.

[3]We note that a similar graph to D was defined in [3]. However, the upper bound on the number of vertices in that graph derived in [3] had a multiplicative factor of $O(k^2)$ over the number of vertices in Y, which was upper bounded by $O(k^4)$, thus resulting in an upper bound of $O(k^6)$ on the kernel size. Here we improve the multiplicative factor to $O(k)$.

For every path $P \in \mathcal{P}$, let h_P be the head of P, and let T be the tree in $\mathcal{F}$ containing h_P. Since h_P is an internal degree-2 vertex in T, and since the unique direction reduction rule does not apply, h_P must have a request to at least one of its descendants in T. Moreover, no descendent of h_P could be in D; otherwise, because the outdegree of h_P in D is 0, a directed edge from h_P to such a descendent would exist in D. Therefore, h_P must have a request to some vertex in Y; fix any such vertex $h'_P \in Y$, and define h'_P to be the image of h_P under ϕ. Clearly, ϕ is a well-defined function. We show next that ϕ is injective. First note that for any head h_P of a path P in a tree T of $\mathcal{F}$, h'_P is on the root-leaf path from r_T going through h_P. Let P_1 and P_2 be two distinct paths in $\mathcal{P}$, and let h_{P_1} and h_{P_2} be their head vertices, respectively. Then $h_{P_1} \neq h_{P_2}$ (because D is a collection of disjoint paths). Suppose, to get a contradiction, that $\phi(P_1) = \phi(P_2) = y$. Then y must appear on the root-leaf path (in T) from r_T passing through h_{P_1}, and y must also appear on the root-leaf path from r_T passing through h_{P_2}. It follows that h_{P_1}, h_{P_2}, y all belong to the same root-leaf path in T starting at r_T. Assume, without loss of generality, that h_{P_1} appears before h_{P_2} on this path that starts at r_T. Since (h_{P_1}, y) and (h_{P_2}, y) are both requests in R and h_{P_1}, h_{P_2} are internal degree-2 vertices, request (h_{P_2}, y) dominates request (h_{P_1}, y), contradicting the fact that the domination reduction rule is not applicable to $\mathcal{F}$. We conclude that ϕ is injective, and hence $|\mathcal{P}| \leq |Y| = O(k^2)$. Each path $P \in \mathcal{P}$ has length at most k since its edges correspond to disjoint requests in R. Therefore, the number of vertices on each path in $\mathcal{P}$ is $O(k)$. Since $|\mathcal{P}| = O(k^2)$, the number of vertices on the paths in $\mathcal{P}$, and hence, the number of vertices in Z is $O(k^3)$. This completes the proof. □

Theorem 3.11 *Let $(\mathcal{F}, R, k)$ be a reduced instance of* MULTICUT. *Then the number of vertices in $\mathcal{F}$ is $O(k^3)$.*

PROOF. The statement of the theorem follows directly from Lemma 3.9 and Lemma 3.10. □

Corollary 3.12 *The* MULTICUT *problem has a kernel of at most $O(k^3)$ vertices.*

PROOF. This follows from Theorem 3.11 and the fact that Reduction Rules 3.1 – 3.12 can be implemented to run in polynomial time. □

4 The algorithm

Let $(\mathcal{F}, R, k)$ be a reduced instance of MULTICUT. Since $(\mathcal{F}, R, k)$ is reduced, we can assume that every tree in $\mathcal{F}$ is nontrivial (i.e., contains at least two vertices), and that there is at least one request between the vertices of every tree in $\mathcal{F}$. We shall assume that every tree in $\mathcal{F}$ is rooted at some internal vertex in the tree (chosen arbitrarily). Let T be a tree in $\mathcal{F}$ rooted at a vertex r. A vertex $u \in V(T)$ is *important* if all the children of u are leaves. For a set of vertices $V' \subseteq V(T)$ and a vertex $u \in V'$, u is *farthest* from r w.r.t. V' if $dist_T(u, r) = \max\{dist_T(w, r) \mid w \in V'\}$. The following lemma, which again emphasizes the importance of VERTEX COVER for both kernelization and parameterized algorithms for MULTICUT, will be pivotal:

Lemma 4.1 *Let $(\mathcal{F}, R, k)$ be a reduced instance of* MULTICUT. *Let T be a tree in $\mathcal{F}$ rooted at r. There exists a minimum cut E_{min} for the requests of R in T such that, for every important vertex $u \in V(T)$, the subset of edges in E_{min} that are incident to the children of u (if any) corresponds to a minimum vertex cover of G_u. (Recall that G_u is the subgraph of G induced by the vertices in G that correspond to the good leaves attached to u.)*

PROOF. Among all minimum cuts of T, let E_{min} be one that minimizes the number of important vertices u in T such that the subset of edges in E_{min} between u and children of u does not correspond to a minimum vertex cover of G_u; let n_{min} be the number of such vertices. We claim that $n_{min} = 0$. Suppose not, then there exists an important vertex u in T such that the subset of edges in E_{min} that are incident to the children of u does not correspond to a minimum vertex cover of G_u. Let E' be the subset of edges in E_{min} that are incident to the children of u and note that E' is a vertex cover of G_u, and let E_u be a subset of edges that are incident to the children of u and that corresponds to a minimum vertex cover of G_u. By the choice of u, we have $|E'| \geq |E_u| + 1$.

If $u = r$, then $E' = E_{min}$ is not a minimum cut (since E_u is a smaller cut in this case), contradicting the optimality of E_{min}. On the other hand, if $u \neq r$, then $(E_{min} \setminus E') \cup E_u \cup \{u\pi(u)\}$ is a cut of the same size as E_{min}, contradicting the choice of E_{min} since this cut contains E_u which corresponds to a minimum vertex cover of G_u. □

We introduce the following new reduction rules:

Reduction Rule 4.1 *Let $(\mathcal{F}, R, k)$ be a reduced instance of* MULTICUT*, let T be a tree in $\mathcal{F}$ rooted at r, and let $u \neq r$ be a vertex in T. If there exists no request between a vertex in $V(T_u)$ and a vertex in $V(T_{\pi(u)}) \setminus V(T_u)$ then contract the edge $u\pi(u)$.*

PROOF. By the hypothesis, there is no request between any vertex in $V(T_u)$ and a vertex in $V(T_{\pi(u)}) \setminus V(T_u)$. If $\pi(u) = r$, then no minimum cut can contain $u\pi(u)$, since removing $u\pi(u)$ from the cut would still yield a cut. On the other hand, if $\pi(u) \neq r$, then since there is no request between any vertex in $V(T_u)$ and a vertex in $V(T_{\pi(u)}) \setminus V(T_u)$, from any edge cut containing $u\pi(u)$ we can obtain an edge cut of the same size by replacing edge $u\pi(u)$ with edge $\pi(u)\pi(\pi(u))$. □

Reduction Rule 4.2 *Let $(\mathcal{F}, R, k)$ be a reduced instance of* MULTICUT*, let T be a tree in $\mathcal{F}$ rooted at r, and let u be an important vertex in T such that $\Delta(G_u) \leq 2$. If there exists a (leaf) child l of u that is not in any minimum vertex cover of G_u, then contract the edge ul.*

PROOF. First note that the existence of such a child can be determined in polynomial time since $\Delta(G_u) \leq 2$. By Lemma 4.1, there exists a minimum cut that does not include the edge ul. Therefore, edge ul can be contracted. □

Noting that a path of even length in a graph has a unique minimum vertex cover, we have the following reduction rule:

Reduction Rule 4.3 *Let $(\mathcal{F}, R, k)$ be a reduced instance of* MULTICUT*, let T be a tree of $\mathcal{F}$ rooted at r, and let w be an important vertex in T such that $\Delta(G_w) \leq 2$. For every path in G_w of even length, cut the leaves in $children(w)$ that correspond to the unique minimum vertex cover of P.*

PROOF. Since a path of even length has a unique minimum vertex cover, which can be computed in in polynomial time since $\Delta(G_w) \leq 2$, if G_w contains a path P of even length, then by Lemma 4.1, we can cut the vertices in $children(w)$ that correspond to the minimum vertex cover of P. □

Definition 4.1 Let $(\mathcal{F}, R, k)$ be a reduced instance of MULTICUT, let T be a tree of $\mathcal{F}$ rooted at r, and let $w \neq r$ be an important vertex in T. A request between a vertex in $V(T_w)$ and a vertex in $V(T_{\pi(w)}) \setminus V(T_w)$ is called a *cross request*.

Reduction Rule 4.4 *Let $(\mathcal{F}, R, k)$ be a reduced instance of* MULTICUT*, let T be a tree rooted at r in $\mathcal{F}$, and let $w \neq r$ be an important vertex in T such that $\Delta(G_w) \leq 2$. If there is a minimum vertex cover of G_w such that cutting the leaves in this minimum vertex cover cuts all the cross requests from the vertices in $V(T_w)$ then contract $w\pi(w)$.*

PROOF. First note that the existence of such a minimum vertex cover can be determined in polynomial time since $\Delta(G_w) \leq 2$. Suppose that there exists a minimum vertex cover C_{min} of G_w such that cutting the leaves in C_{min} cuts all the cross requests from the vertices in $V(T_w)$. Consider a minimum cut C, and suppose that C contains $w\pi(w)$. If $\pi(w) = r$, then since there is no cross request from $V(T_w)$, $C - w\pi(w)$ is still a cut, contradicting the optimality of C. On the other hand, if $\pi(w) \neq r$, then replacing $w\pi(w)$ in C with the edge between $\pi(w)$ and its parent (i.e., $\pi(\pi(w))$), and replacing the edges in C that are incident to the children of w with the edges that are incident to the leaves in C_{min}, yields a cut of size at most $|C|$, and hence, a minimum cut, containing the edges that are incident to the leaves in C_{min}. □

Definition 4.2 The instance $(\mathcal{F}, R, k)$ of MULTICUT is said to be *strongly reduced* if $(\mathcal{F}, R, k)$ is reduced and none of Reduction Rules 4.1 – 4.4 is applicable to the instance.

Proposition 4.2 *Let $(\mathcal{F}, R, k)$ be a strongly reduced instance, and let T be a tree in $\mathcal{F}$ rooted at a vertex r. Then the following are true:*

(i) For any vertex $u \in V(T)$, there exists no request between u and $\pi(u)$.

(ii) For any vertex $u \neq r$ in $V(T)$, there exists a request between some vertex in $V(T_u)$ and some vertex in $V(T_{\pi(u)}) \setminus V(T_u)$.

(iii) For any internal vertex $u \in V(T)$, there exists at least one request between the vertices in $V(T_u) - u$.

(iv) For any important vertex $w \in V(T)$ such that $\Delta(G_w) \leq 2$ and any child u of w, there exists a request between u and a sibling of u, and hence all the children of an important vertex are good leaves.

(v) For any important vertex $w \in V(T)$ such that $\Delta(G_w) \leq 2$, G_w contains no path of even length.

(vi) For any important vertex $w \neq r$ in $V(T)$ such that $\Delta(G_w) \leq 2$, there is no minimum vertex cover of G_w such that cutting the leaves in this minimum vertex cover cuts all the cross requests from the vertices in $V(T_w)$.

PROOF.

(i) This follows directly from the fact that the unit request reduction rule (Reduction Rule 3.3) is not applicable to T.

(ii) This follows directly from the fact that Reduction Rule 4.1 is not applicable to T.

(iii) Proceed by contradiction. Let u be an internal vertex in $V(T)$, and assume that there is no request between any two vertices in $V(T_u) - u$. Since Reduction Rule 4.1 is not applicable, u does not have any grandchildren (otherwise the reduction rule would apply to any grandchild of u), and hence all the children of u must be leaves. Moreover, u cannot have any children either, otherwise, since all the children of u must be leaves, by Reduction Rule 4.1, if u has a child x then there must exist a request between x and u. This, however, contradicts part (i) above.

(iv) This follows directly from the fact that Reduction Rule 4.2 is not applicable to T since such a child of w would be an isolated vertex in G_w.

(v) This follows directly from the fact that Reduction Rule 4.3 is not applicable to T.

(vi) This follows directly from the fact that Reduction Rule 4.4 is not applicable to T.

□

We are now ready to present the algorithm. Let $(\mathcal{F}, R, k)$ be an instance of MULTICUT. Clearly, in polynomial time we can either reject the instance or transform it into an equivalent strongly reduced instance. Therefore, we shall assume that $(\mathcal{F}, R, k)$ is strongly reduced. The algorithm is a branch-and-search algorithm, and its execution can be depicted by a search tree. The running time of the algorithm is proportional to the number of root-leaf paths, or equivalently, to the number of leaves in the search tree, multiplied by the time spent along each such path, which will be polynomial in k. Therefore, the main step in the analysis of the algorithm is to derive an upper bound on the number of leaves $L(k)$ in the search tree. We shall assume that the instance $(\mathcal{F}, R, k)$ is strongly reduced before every branch of the algorithm. We shall also assume that the branches are considered in the listed order. In particular, when a branch is considered, $(\mathcal{F}, R, k)$ is strongly reduced and none of the preceding branches applies. We first make the following observations.

Observation 4.3 Let T be a tree in $\mathcal{F}$ rooted at r, let $w \neq r$ be an important vertex in T, and let $S \subseteq \mathit{children}(w)$ be such that S is contained in some minimum vertex cover of G_w. If edge $w\pi(w)$ is in some minimum cut of T, then the edges incident to the leaves of any minimum vertex cover of G_w are contained in some minimum cut: simply replace all the edges that are incident to the children of w in a minimum cut that contains $w\pi(w)$ with the edges incident to the leaves corresponding to the desired minimum vertex cover of G_w. Since S is contained in some minimum vertex cover of G_w, there is a minimum cut that contains the edges that are incident to the (leaf) children of w that are in S. Therefore, if we choose edge $w\pi(w)$ to be in the solution, then we can choose the edges in $\{wu \mid u \in S\}$ to be in the solution as well. If we choose to cut the children of w that are in S when we cut edge $w\pi(w)$ in a branch, then we say that we *favor* the vertices in S in this branch; this observation will be very useful when branching. If S consists of a single vertex u, we simply say that we *favor* vertex u. Note that if we favor the vertices in S, then using the contrapositive of the statement "if w is cut then the vertices in S are cut", if we decide not to cut a *certain* vertex in S in a branch, then we can assume that w will not be cut as well in the same branch. If we decide not to cut an edge in a certain branch, we say that the edge is *kept* and we can contract it.

Observation 4.4 Let T be a tree in $\mathcal{F}$ and let $w \in V(T)$ be an important vertex. Let $v \in G_w$, and recall that $\mathit{deg}_G(v)$ denotes the degree of v in G_w. By Lemma 4.1, we can assume that the set of edges in T_w that are contained in the solution that we are looking for corresponds to a minimum vertex cover of G_w. Since any minimum vertex cover of G_w either contains v, or excludes v and contains its neighbors, we can branch by cutting v in the first side of the branch, and by cutting the neighbors of v in G_w in the second side of the branch. Note that by part (iv) of Proposition 4.2, and the fact that there is no request between a child and its parent (unit request rule), there must be at least one request between v and another child of w, and hence, $\mathit{deg}_G(v) \geq 1$.

Observation 4.4 leads to the following branching rule:

BranchRule 4.5 *Let T be a tree in $\mathcal{F}$, and let $w \in V(T)$ be an important vertex. If there exists a vertex $v \in G_w$ such that $\mathit{deg}_G(v) \geq 3$, then branch by cutting v in the first side of the branch, and by cutting the neighbors of v in G_w in the second side of the branch. Cutting v reduces the parameter k by 1, and cutting the neighbors of v in G_w reduces k by at least 3. Therefore, the number of leaves in the search tree of the algorithm, $L(k)$, satisfies the recurrence relation: $L(k) \leq L(k-1) + L(k-3)$.*

After BranchRule 4.5, we can now assume that for any important vertex w in a tree of $\mathcal{F}$, we have $\Delta(G_w) \leq 2$, and hence, G_w consists of a collection of disjoint paths and cycles.

Let T be a tree in $\mathcal{F}$ rooted at r. Among all important vertices in T, let w be a vertex that is farthest from r. Since every subtree of T contains an important vertex, w must be a farthest vertex among all internal vertices of T. By part (ii) of Proposition 4.2, there exists a cross request between a vertex in $V(T_w)$ and a vertex in $V(T_{\pi(w)}) \setminus V(T_w)$. Since w is farthest from r, the cross request between a vertex in $V(T_w)$ and a vertex in $V(T_{\pi(w)}) \setminus V(T_w)$ can be either a request: (1) between w and a sibling of w, (2) between w and a nephew of w, (3) between a child of w and its grandparent $\pi(w)$, (4) between a child of w and an uncle, or (5) between a child of w and a cousin. By symmetry (and by the choice of w), the case when there is a request between w and a nephew is identical to the case when there is a request between a child of w and an uncle. Therefore, we shall only treat the latter case.

Case 4.6 *Vertex w has a cross request to a sibling w'.*

In this case at least one of w, w' must be cut. We branch by cutting w in the first side of the branch, and cutting w' in the second side of the branch. Note that by part (iii) of Proposition 4.2, the size of a minimum vertex cover in G_w is at least 1. Moreover, a minimum vertex cover for G_w can be computed in polynomial time since $\Delta(G_w) \leq 2$. Therefore, in the first side of the branch we end up cutting the edges corresponding to a minimum vertex cover of G_w, which reduces the parameter further by at least 1. Therefore, we have $L(k) \leq L(k-2) + L(k-1)$ in this case.

Case 4.7 *There exists a child u of w such that u has a cross request to its grandparent $\pi(w)$.*

In this case we can cut u. This can be justified as follows. Any minimum cut of T either cuts $w\pi(w)$ or does not cut it. If the minimum cut cuts $w\pi(w)$, then we can assume that it cuts edge wu as well because by Reduction Rule 4.2, u is in some minimum vertex cover of G_w. On the other hand, if the minimum cut does not cut $w\pi(w)$, then it must cut edge wu since $(u, \pi(w)) \in R$. It follows that in both cases there is a minimum cut that cuts wu. We have $L(k) \leq L(k-1)$ in this case.

Now we can assume that the cross request is between a child u of w and either an uncle or a cousin of u.

Case 4.8 *There exists a child u of w such that u has a cross request to an uncle w'.*

We favor u and branch as follows. In the first side of the branch we cut u. In the second side of the branch we keep edge uw, and cut the neighbor(s) of u in G_w. Since u is not cut in the second side of the branch and u is favored, w is not cut as well, and hence w' must be cut. Noting that u has at least one neighbor in G_w, $L(k)$ satisfies the recurrence relation $L(k) \leq L(k-1) + L(k-2)$.

Case 4.9 *There exists a child u of w such that u has a cross request to a cousin u'.*

Let $w' = \pi(u')$ and note that $\pi(w) = \pi(w')$. We favor u and u' (thus if u' is not cut then w' is not cut as well). We branch as follows. In the first side of the branch we cut u. In the second side of the branch uw is kept and we cut the neighbor(s) of u in G_w. Since in the second side of the branch uw is kept and u is favored, $w\pi(w)$ is kept as well, and u' must be cut (otherwise, w' is not cut as well because u' is favored) since $(u, u') \in R$. Therefore, $L(k)$ in this case satisfies the recurrence relation $L(k) \leq L(k-1) + L(k-2)$.

Theorem 4.10 *The* MULTICUT *problem can be solved in time* $O^*(\rho^k)$*, where* $\rho = (\sqrt{5}+1)/2 \approx 1.618$ *is the positive root of the polynomial* $x^2 - x - 1$*.*

PROOF. From the above branching it follows that the number of leaves in the search tree corresponding to the algorithm satisfies the recurrence relation $L(k) \leq L(k-1) + L(k-2)$, whose characteristic polynomial is $x^2 - x - 1$. It follows that $L(k) \in O^*(\rho^k)$, where $\rho = (\sqrt{5}+1)/2 \approx 1.618$ is the positive root of the polynomial $x^2 - x - 1$. Since the time spent by the algorithm along each root-leaf path in the search tree is polynomial, the theorem follows. □

5 Acknowledgment

The third coauthor would like to thank Jiong Guo and Stefan Kratsch for several discussions about the MULTICUT problem.

References

[1] F. A. Abu-Khzam, R. Collins, M. Fellows, M. Langston, W. Suters, and C. Symons. Kernelization algorithms for the vertex cover problem: theory and experiments. In *Proceedings of 6th Workshop on Algorithm Engineering and Experiments*, pages 62–69, 2004.

[2] N. Bousquet, J. Daligault, and S. Thomassé. Multicut is fpt. *CoRR*, abs/1010.5197, 2010.

[3] N. Bousquet, J. Daligault, S. Thomassé, and A. Yeo. A polynomial kernel for multicut in trees. In *Proceedings of the 26th Symposium on Theoretical Aspects of Computer Science*, pages 183–194, 2009.

[4] J. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993.

[5] R. Downey and M. Fellows. *Parameterized Complexity.* Springer, New York, 1999.

[6] J. Flüm and M. Grohe. *Parameterized Complexity Theory.* Springer-verlag, Berlin, Germany, 2010.

[7] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[8] J. Guo and R. Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Networks*, 46(3):124–135, 2005.

[9] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

[10] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *CoRR*, abs/1010.3633, 2010.

[11] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, USA, 2006.

[12] Douglas B. West. *Introduction to graph theory.* Prentice Hall Inc., Upper Saddle River, NJ, 1996.