CUTS AND PARTITIONS IN GRAPHS/TREES WITH APPLICATIONS

A Dissertation

by

JIA-HAO FAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2013

Major Subject: Computer Science

ABSTRACT

Graph cutting and partitioning problems are among the most interesting and important topics in computational theory. In this thesis, we develop approximation algorithms, parameterized algorithms, and kernelization algorithms for cut and partition problems in graphs and trees, and study their applications. We are focused on three problems in this thesis: the maximum agreement forest problem, the multicut on trees problem, and the protein complex prediction problem.

Both the maximum agreement forest problem and the multicut on trees problem are NP-hard, thus cannot be solved efficiently if P $\neq$ NP. The maximum agreement forest problem was motivated in the study of evolution trees in bioinformatics, in which we are given two leaf-labeled trees and are asked to find a maximum forest that is a subgraph of both trees. The multicut on trees problem has applications in networks, in which we are given a forest and a set of pairs of termianls and are asked to find a cut that separates all pairs of terminals.

We develop combinatorial and algorithmic techniques that lead to improved parameterized algorithms, approximation algorithms, and kernelization algorithms for these problems. For the maximum agreement forest problem, we proceed from the bottommost level of trees and extend solutions to whole trees. With this technique, we show that the maximum agreement forest problem is fixed-parameterized tractable in general trees, resolving an open problem in this area. We also provide the first constant ratio approximation algorithm for the problem in general trees. For the multicut on trees problem, we take a new look at the problem through the eyes of vertex cover problem. This connection allows us to develop an kernelization algorithm for the problem, which gives an upper bound of $O(k^3)$ on the kernel size, significantly improving the previous best upper bound $O(k^6)$. We further exploit this connection to give a parameterized algorithm for the problem that runs in time $O^*(1.62^k)$, thus improving the previous best algorithm of running time $O^*(2^k)$.

In the protein complex prediction problem, which comes directly from the study of bioinformatics, we are given a protein-protein interaction network, and are asked to find dense regions in this graph. We formulate this problem as a graph clustering problem and develop an algorithm to refine the results for identifying protein complexes. We test our algorithm on yeast protein- protein interaction networks, and we show that our algorithm is able to identify complexes more accurately than other existing algorithms.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. First, I would like to thank to Department of Computer Science of Texas A&M for offerring me teaching assistantships that provided me an opportunity to finish my degree without the concerns of finacial issues and offered me various kinds of resources that facilitated the computatioal experiments in this thesis.

I am deeply indebted to my adviser, Jianer Chen whose sincerely suggestions and comments helped me all the time for both doing researches and writing papers. In particular, I am really grateful to his patience for going through the details of proofs with me.

I would also like to thank to my co-adviser, Sign-Hoi Sze who helps me to understand fundamental knowledges in bioinformatics, and leads me to accomplish some projects in this field.

I would also like to thank to my former colleagues, Yang Liu and Iyad whose view of researches and writing skills help me to extend my mind in this field.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION [*]

In theoretical computer science, many practical problems are formulated as theoretical problems on graphs or trees [68] such as *flow network* problem. In particular, with the development of experimental technologies in biology, large amount of data have been collected. These data are usually systematically represented as trees or graphs. For example, a protein-protein interaction network is usually represented as an undirected graph [56] [73] [66], and the evolutionary relationships among groups of species is usually visualized as trees [2, 24, 28, 37, 40]. With the biological data that are presented as graphs or trees, some bioinformatic problems are also formulated as problems on graphs or trees. For example, in the *protein complex prediction* problem, a protein-protein interaction network is given as a graph, and we are asked to find protein complexes, usually the dense regions in the graph. In the *maximum agreement forest* problem, we are given two phylogenetic trees, and we are asked to find a maximum forest that is a subgraph of both trees. For the practical problems that are formulated as theoretical problems on graphs or trees, well-developed graph theories provide a direction to solve them. In particular, some of these problems are formulated as graph cutting or graph partitioning problems. In this thesis, we focus on the algorithms for three cutting and partitioning problems in graphs or trees, *maximum agreement forest* problem, *multicut on trees* problem and *protein complex prediction* problem.

Graph cutting and graph partitioning problems present significant challenges. First, most graph cutting problems and graph partitioning problems are NP-hard. It means that these problems cannot be solved efficiently if P $\neq$ NP. The two theoretical problems that we studied in this thesis, maximum agreement forest problem and multicut on trees problem

[*]Reprinted with permission from "Identifying Complexes from Protein Interaction Networks According to Different Types of Neighborhood Density" by Jia-Hao Fan, Jianer Chen and Sing-Hoi Sze, 2012. Journal of Computational Biology, 19(12): 1284-1294 (2012), Copyright [2012] by Mary Ann Liebert, Inc., and Reprinted with permission from "Multicut in trees viewed through the eyes of vertex cover" by Jianer Chen, Jia-Hao Fan, Iyad A. Kanj, Yang Liu and Fenghui Zhang, 2012. J. Comput. Syst. Sci., 78(5): 1637-1650 (2012), Copyright [2012] by Elsevier.

are NP-hard and MAX SNP-hard [2, 10, 8, 32]. In this thesis, we develop parameterized algorithms, approximation algorithms and kernelization algorithms to these two theoretical problems. Second, the optimal solutions that we obtained to the theoretical problems that we formulated on graphs or trees could not always reflect the true solutions to the corresponding practical problems. In protein complex prediction problem, it can be formulated as a problem in graphs, in which we are given a graph that represents a protein-protein interaction network, and we are asked to find dense regions of the given graph. However, the known protein complexes (solutions to the practical problem) do not always correspond to the dense regions (optimal solutions to the theoretical problem that we formulate) in protein-protein interaction network. In this thesis, we improved our formulation with biological observation. We find that the protein complexes have two trends with respect to the neighborhood density in the protein-protien interaction network. Thus, we devise an algorithm to predict protein complexes based on the observations.

In the following sections, we briefly introduce the three problems that we study in this thesis.

1.1 Maximum agreement forest problem

Evolutionary relationships among a set of species is usually modeled by phylogenetic trees, in which each leaf is labeled by a distinct species, and the taxa in adjacent branches are thought to have descended from a common ancestor. There are several methods to construct phylogenetic trees (i.e. morphology, molecular biology, etc.). For a given set of species, different methods often lead to different trees. Researchers believe that evolutionary relationships of a set of species should be identical. Thus, it makes sense to ask that for a given set of species, how close are the phylogenetic trees that are constructed by different methods.

Several distance metrics have been proposed for measuring the similarity of different phylogenetic trees [2, 24, 28, 37, 40]. In particular, the *tree-bisection-and-reconnection* (*TBR*) and the *subtree-prune-and-regraft* (*SPR*) distances [5, 36, 70] are directly corresponding to the size of the *maximum agreement forest* (abbr. *MAF*) on unrooted trees [2] and on rooted trees [11], respectively.

While most previous work on MAF is restricted to bifurcating (i.e., binary) trees, the problem and related problems on multifurcating (i.e., general) trees have drawn attention recently. Since there are often ambiguities in which the order of more than two branches cannot be reliably resolved by phylogenetic tree construction algorithms, it is important to allow the given trees to be multifurcating. There is also the possibility that some of the nodes are truly multifurcating due to simultaneous divergence that results in multiple descendants in a very narrow time frame [51], such as the case demonstrated in Kliman et al. [46] for the three *Drosophila* species *D. mauritiana*, *D. sechellia*, and *D. simulans*. In addition, the evolutionary mechanism of prokaryotes is different from eukaryotes [7], and it is often more suitable to represent the species tree for prokaryotes as a multifurcating tree. The NCBI taxonomy [69], which is an important source of species trees, has more than half of its branches being multifurcating [51].

In this thesis, we focus on parameterized algorithms and approximation algorithms for the MAF problem on unrooted general trees. Our method is based on a careful study of the graph structures that takes advantage of special relationships among sibling leaves in the given trees. We develop an $O(3^k n)$-time parameterized algorithm for the MAF problem on unrooted general trees, thus showing the fixed-parameter tractability of the problem and resolving the open problem posed in [36, 70]. In fact, our algorithm is even faster than the previous best parameterized algorithm for the problem on binary trees, which runs in time $O(4^k n)$ [71]. We also present a polynomial-time approximation algorithm of ratio 3 for the MAF problem on unrooted general trees. The ratio matches the best known approximation ratio for the problem on unrooted binary trees [70, 71], but our algorithm keeps the same constant ratio and works for general trees. The only previously known approximation algorithm for the MAF problem on general trees [62] is on rooted trees and has a ratio of $d + 1$, where d is the maximum number of children a node in the trees may have. Our algorithm is the first constant-ratio approximation algorithm for the MAF problem on general trees, which is on unrooted trees.

1.2 Multicut on trees problem

In graph theory, given an undirected graph $G = (V, E)$, a *cut* is a partition of vertices V into two disjoint subsets S and $V \setminus S$, which is denoted by $(S, V \setminus S)$. The *cut-set* of a cut $(S, V \setminus S)$ is the set of edges whose two ends are in different subsets S and $V \setminus S$ of the partition. Edges in the cut-set of the cut are said to be *crossing edges*.

In an unweighted undirected graph, the size of a cut is defined by the number of the crossing edges. In a weighted undirected graph, it is defined by the sum of the weights of the crossing edges.

The definition of cuts in directed graphs is usually slightly different from the definition of cuts to the same problems in undirected graphs. Now we give a most well-known and simple graph cutting problem in directed graph as an example.

Minimum *s-t cut* problem is one of the most famous graph cutting problem in graph theory, which is equivalent to *maximum flow* problem by *max-flow min-cut theorem*.

A *flow network*, $G = (V, E)$ is a directed graph in which every edge $e = (u, v) \in E$ has a nonnegative *capacity* $c(u, v) \geq 0$ [20]. A *flow* in a network G is a real-function $f : V \times V \to \mathbb{R}$ with the following two properties:

$$\text{For all } u, v \in V, 0 \leq f(u, v) \leq c(u, v)$$

$$\text{For all } u \in V - \{s, t\}, \sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

In the maximum flow problem, we are given a flow network G, a source s and a sink t, and we are asked to find a flow of maximum volume.

This problem is polynomial time solvable by Edmonds-Karp algorithm [26]. In addition, by the *max-flow min-cut theorem*, the maximum amount of flow passing from the source to the sink is equal to the minimum capacity which when removed in a specific way that there is no path from source to sink in the induced graph. Thus, minimum *s-t cut* problem is also polynomial time solvable.

Undirected graphs are special cases of directed graphs. Given an undirected graph, sometimes, we can obtain its corresponding directed graph by replacing every edge with two opposite arrows. If the problem in the version to undirected graphs has these properties, we always can solve it by the same approach to the problems in the version in directed graphs. Fortunately, for s-t cut problem, we can extend it to undirected graphs.

The *max-flow min-cut theorem* describes the relations of minimum cut size and maximum flow size in directed graph. In undirected graphs, *Menger's theorem* is the corresponding theorem to describe the relations of minimum cut size and maximum number of independent paths. It has two versions, edge-connectivity and vertex-connectivity. Let G be an undirected graph and s and t be two distinct vertices (non-adjacent vertices in the vertex version). The edge(resp. vertex)-connectivity version states that the maximum number of pairwise edge(resp. vertex)-independent paths from s to t is the same as the minimum edge(resp. vertex) cut for s and t. Clearly, the corresponding edge and vertex connectivity problems are also polynomial time solvable using a similar approach.

The *multiway cut* problem generalizes the s-t cut problems with respect to the vertex-connectivity. In the multiway cut problem, we are given a graph $G = (V, E)$ and terminals $T \subseteq V$, and we are asked to find vertices $C \subseteq V$ such that in the induced graph, with C removed from G, there is no path between any pair of terminals. Similarly, the *multiterminal cut* problem generalizes the s-t cut problems with respect to the edge-connectivity.

Although the s-t cut problem is polynomial time solvable, multiterminal cut problem was shown to be NP-hard by Dahlhaus et al. [23] (i.e. multiway cut problem is NP-hard) even when the number of terminals is only 3 or the graph is planar. Additionally, they showed that it is MAX SNP-hard(i.e. there is no polynomial-time approximation scheme(PTAS) unless P=NP), and they gave the first approximation algorithm of constant ratio $2(1 - \frac{1}{p})$ to this problem where p is the number of terminals. Later, Calinescu et al. [16] formulated this problem as a linear program. With the novel geometric relaxation to this problem and a rounding scheme, they obtained an approximation algorithm of ratio $\frac{3}{2} - \frac{1}{p}$. In addition, Karger et al. [43] showed that there exists a rounding scheme with performance ratio equal to the integrality gap to improve the ratio to 1.3438.

In parameterized complexity, Marx [54] conducted a comprehensive study of the graph separation problems, and they introduced the concept of important separators. Then, he showed that the multiway cut problem is FPT when it is parameterized on the size of the cut-set. Later, Chen et al. [18] improved the running time to $O^*(4^k)$ where k is the solution size (size of the cut-set). For the multiterminal cut problem, Xiao [74] had a new measure and further improved the running time to $O^*(2^k)$. Recently, Cygan et al. [22] studied this problem from the perspective of reasonable lower bounds of the cut-set size. More specifically, they looked for FPT algorithms that were parameterized on the difference of the cut-set size and the maximum separating cut or on the difference of the cut-set size and a natural LP-relaxation instead of the solution size (the size of the cut-set). They also obtained an $O^*(2^k)$ algorithm to the multiway cut problem in standard setting. For the kernerlization to this problem, Razgon [60] had a polynomial kernel for the multiway cut problem in the restricted graph. He showed that when the difference between the size of the cut-set k and the size of the smallest isolating cut is at most $\log(k)$, then there exists a polynomial kernel for this problem. Note, the multiway cut problem is the simplest graph separation problem; however, the question of the polynomial kernel for this problem remains unanswered.

The *multicut* problem also generalizes the multiway cut problem. In the multicut problem, we are given a graph $G = (V, E)$, and a set of requests $R \subseteq V(G) \times V(G)$ between pair of vertices in G, and we are asked to find edges $C \subseteq E$ such that in the induced graph, with C removed from G, there is no path between each pair of vertices in R.

Dahlhaus et al. [23] devised the first approximation algorithm to this problem of ratio 2 when the size of the cut-set k is fixed (i.e. the running time of this algorithm is $O(n^{O(k)})$). Later, Klein et al. [45] used their approximation algorithm on the *sparsest cut* problem to devise an approximation algorithm of time $O(\log C \log^2 k)$ where C is the sum of the capacity. In addition, Garg et al. [31] considered this minimization cutting problem in association with the maximization multicommodity flow problem and established the max-flow min-multicut theorem: $\frac{M}{\log k} \leq f \leq M$ where M is the minimum multicut, f is the maximum multi-commodity flow, and k is the number of commodities. Then they gave an $O(\log n)$ approximation algorithm to this problem.

In parameterized complexity, the question about whether the multicut problem is FPT had been unanswered for so long until recently, Marx and Razgon [55] and Bousquet et al. [12] separately provided the positive answer to this question.

If the multicut problem is restricted to trees, it is still NP-hard [8] and MAX-SNP hard [32]. Tree is a special graph with no cycles; thus, problems on trees are easier than problems in general graphs. Because there is no polynomial kernel so far to the above NP-hard cutting problems, it becomes interesting to explore the parameterized complexity and the kernelization of the multicut problem on trees.

In this thesis we take a new look at multicut on trees problem 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 on trees problem, significantly improving the previous $O(k^6)$ upper bound given by Bousquet et al. [13].

To obtain the $O(k^3)$ upper bound on the kernel size, we introduce a novel approach that relies on a grouping of the vertices in the tree. This grouping allows us to derive tighter upper bounds on the number of vertices in the tree after applying some new reduction rules that we introduce in this thesis. Some of the reduction rules we apply exploit a connection between vertex cover problem and multicut on trees problem that is observed in this thesis. The novel approach can be summarized as follows. We first group the vertices in the tree into $O(k)$ groups. We then introduce an ordering that orders the 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 rules is a rule that utilizes the crown kernelization algorithm for vertex cover problem [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 [13]. 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 new structural connections between multicut on trees problem and vertex cover problem that allow us to simplify the instance of multicut on trees problem. We then exploit the simplified structure of the resulting instance to present a simple search-tree algorithm for multicut on trees problem that runs in time $O^*(((\sqrt{5}+1)/2)^k)$. We note that, even though some connection between multicut on trees problem and vertex cover problem was observed in [32, 34], this connection was not developed or utilized in kernelization algorithms, nor in parameterized algorithms for multicut on trees problem.

1.3 Protein complex prediction problem

In biology, large amount of experiments have been conducted to elucidate the protein-protein interactions. These data are collected as protein-protein interaction (PPI) networks [56] [73] [66]. PPI network is usually represented as an undirected graph $G = (V, E)$ where each protein is represented as a node, and each interaction is represented as an edge. This structural data allows bioinformaticists to apply some known theoretical results in graph theory to conduct computational experiments in the corresponding problems in bioinformatics.

In this thesis, we observe that most complexes either reside in very dense regions, in which most algorithms are able to identify, or they reside in regions with low neighborhood density, in which most algorithms are less successful to identify. We investigate the following algorithm to consider these two types of complexes separately. Given a protein interaction network, we first identify all the maximal cliques. For each maximal clique, we count the number of other maximal cliques that overlap significantly with it and use it to define neighborhood density. We subdivide these cliques into two sets, with one containing cliques with low neighborhood density and the other containing cliques with high neighborhood density.

Since the maximal cliques with low neighborhood density are likely to correspond to the core region of a complex, we extend each clique to include more proteins as long as the density remains high. This allows each prediction to become a dense subgraph that is not necessarily a clique. Since the maximal cliques with high neighborhood density have overlap with many other maximal cliques, using these cliques directly, as predicted complexes, will lead to significant overestimate of the number of complexes. We extract the most shared sets of proteins from these cliques. We obtain the set of predicted complexes by collecting the above two types of predictions.

We compare the performance of our algorithm to other complex prediction algorithms on a few protein interaction networks and show that our algorithm is able to identify complexes more accurately with respect to complex agreement, complex accuracy, and protein pair agreement measures.

2. RELATED RESEARCH AND PROBLEM REFORMULATIONS [*]

2.1 Maximum agreement forest problem

In maximum agreement forest problem, we are given two leaf-labeled trees and are asked to find a maximum forest that is a subgraph of both trees.

2.1.1 Review on related research

The problem of constructing an MAF for two unrooted trees is NP-hard and MAX SNP-hard, even when it is restricted to binary trees [2, 10].

Approximation algorithms have been studied for the problem, mainly on binary trees. An approximation algorithm of ratio 3 for the problem on rooted binary trees was claimed by Hein et al. [37], who also claimed that the MAF problem on rooted binary trees correspond to the SPR distance. Allen and Steel [2] showed that the claim in [37] on the relationship between MAF and SPR was not true, and, on the other hand, proved that the MAF problem on unrooted binary trees corresponds to the TBR distance. Rodrigues et al. [61] found a subtle error in [37] and showed that the algorithm in [37] has ratio at least 4. Rodrigues et al. [61] then presented a new approximation algorithm and claimed that their algorithm have ratio 3. Bonet et al. [9] provided a counterexample and showed that for the TBR distance, the algorithm in [37] has approximation ratio at least 5 while the algorithm in [61] has approximation ratio at least 4. Using very different methods, Chataigner [17] developed an approximation algorithm of ratio 8 for the TBR distance for two or more binary trees. Recently, Whidden et al. [70, 71] presented a linear-time approximation algorithm of ratio 3 for the TBR distance on unrooted binary trees. This is the best known approximation algorithm for the TBR distance on binary trees. We note

[*]Reprinted with permission from "Identifying Complexes from Protein Interaction Networks According to Different Types of Neighborhood Density" by Jia-Hao Fan, Jianer Chen and Sing-Hoi Sze, 2012. Journal of Computational Biology, 19(12): 1284-1294 (2012), Copyright [2012] by Mary Ann Liebert, Inc., and Reprinted with permission from "Multicut in trees viewed through the eyes of vertex cover" by Jianer Chen, Jia-Hao Fan, Iyad A. Kanj, Yang Liu and Fenghui Zhang, 2012. J. Comput. Syst. Sci., 78(5): 1637-1650 (2012), Copyright [2012] by Elsevier.

that there is also a line of research on another metric, the *rSPR distance*, on binary trees [9, 71], for which the best approximation algorithm has ratio 3 and runs in linear time [70, 71]. For general trees, to our knowledge, there are currently no known approximation algorithms for the TBR distance on general trees. For the SPR distance on rooted general trees, Rodrigues et al. [62] developed an approximation algorithm of ratio $d + 1$, where d is the maximum number of children a node in the input trees may have. There is also a line of research on the *maximum acyclic agreement forest* problem on general trees (see, for example, [52]).

Parameterized algorithms for the MAF problem, parameterized by the number k of trees in the MAF, have also been studied. Allen and Steel [2] showed that the MAF problem on unrooted binary trees, which corresponds to the TBR distance, is fixed-parameter tractable. By branching based on inconsistent structures in quartets, Hallett and McCartin [36] developed an algorithm of time $O(4^k k^5 + n^{O(1)})$ for the problem. Whidden and Zeh [71, 70] further improved the time complexity to $O(4^k n)$, which is currently the best known parameterized algorithm for the MAF problem on unrooted binary trees. For the MAF problem on rooted binary trees, Bordewich et al. [10] proposed a parameterized algorithm of time $O(4^k k^4 + n^3)$, and Whidden et al. [70, 71] improved the time complexity to $O(2.42^k n)$. While there has been significant work that shows the fixed-parameter tractability for the MAF problem and related problems on binary trees, it was unknown whether the MAF problem on general trees is fixed-parameter tractable or not. This had been posed specifically as an open problem by a number of researchers [36, 70].

2.1.2 Definitions and problem reformulations

In this thesis, all graphs are undirected. For a vertex v, an edge e, and an edge subset E' in a graph G, denote by $G - v$, $G - e$, and $G - E'$ the graphs obtained from G with v, e, and the edges in E' removed, respectively. All trees in our discussion are unrooted. A *leaf* of a tree is a vertex of degree less than 2. A *forest* is a collection of disjoint trees. A nonempty forest $\mathcal{F}$ is *leaf-labeled* over a label-set L if there is a one-to-one mapping from the leaves of $\mathcal{F}$ to the elements of L (and if all non-leaf vertices are *unlabeled*). The label for a leaf v is denoted by $\ell(v)$. More generally, for a subforest $\mathcal{F}'$ of $\mathcal{F}$, denote by $\ell(\mathcal{F}')$ the set of labels for the leaves in $\mathcal{F}'$.

Two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L are *isomorphic* if there is an isomorphism between $\mathcal{F}_1$ and $\mathcal{F}_2$ such that the corresponding leaves have the same label. The forests $\mathcal{F}_1$ and $\mathcal{F}_2$ are *homeomorphic* if they become isomorphic after smoothing all degree-2 vertices (*smoothing* a degree-2 vertex v is to replace the vertex v and its incident edges with a new edge connecting the two neighbors of v). Note that if a leaf-labeled forest $\mathcal{F}_1$ is homeomorphic to a subforest of a leaf-labeled forest $\mathcal{F}_2$, then there is a unique subforest of $\mathcal{F}_2$ that is homeomorphic to $\mathcal{F}_1$. Therefore, in this case, without any confusion, we can simply say that the forest $\mathcal{F}_1$ *is a subforest of* $\mathcal{F}_2$. An *agreement forest* for two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L is a leaf-labeled forest $\mathcal{F}'$ over the label-set L such that $\mathcal{F}'$ is a subforest of both $\mathcal{F}_1$ and $\mathcal{F}_2$. A *maximum agreement forest* $\mathcal{F}^*$ (abbr. MAF) for $\mathcal{F}_1$ and $\mathcal{F}_2$ is an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ such that the *size* of (i.e., the number of trees in) $\mathcal{F}^*$ is minimized over all agreement forests for $\mathcal{F}_1$ and $\mathcal{F}_2$ [36].

The two versions of the MAF problem studied in the current thesis are

> PARA-MAF. Given two leaf-labeled trees T_1 and T_2 over the same label-set L, and a parameter k, is there an agreement forest of size at most k for T_1 and T_2?
>
> MAX-MAF. Given two leaf-labeled trees T_1 and T_2 over the same label-set L, construct an MAF for T_1 and T_2.

Our algorithms on a pair (T_1, T_2) of leaf-labeled trees will proceed by removing edges in the tree T_2 to construct a subforest of T_2 that is an agreement forest for T_1 and T_2. Removing edges in T_2 will result in a forest consisting of more than one tree. Therefore, our algorithms will really work on a pair of forests $\mathcal{F}_1$ and $\mathcal{F}_2$. However, the size of an agreement forest $\mathcal{F}'$ for two forests may not properly reflect the complexity of the construction of $\mathcal{F}'$: the size of $\mathcal{F}'$ also heavily depends on the sizes of the given forests $\mathcal{F}_1$

and $\mathcal{F}_2$. Thus, we need a careful reformulation of the problems that allows us to apply a more accurate analysis on the problem complexity.

A partition $\mathcal{P} = \{L_1, \ldots, L_r\}$ of a label-set L, where $L_i \neq \emptyset$, $\bigcup_{i=1}^{r} L_i = L$, and $L_i \cap L_j = \emptyset$, for all i and j, will be called an *L-partition*, in which each L_i is called a *label-subset*. A label-subset is *unit* if it consists of a single label. For a leaf-labeled forest $\mathcal{F} = \{T_1, \ldots, T_h\}$ over a label-set L, where each T_i is a leaf-labeled tree, the L-partition $\{\ell(T_1), \ldots, \ell(T_h)\}$ will be called the *label-partition* for $\mathcal{F}$. For a subset L' of L, denote by $\mathcal{F}[L']$ the subforest of $\mathcal{F}$ *induced* by L', that is, $\mathcal{F}[L']$ consists of all paths in $\mathcal{F}$ that connect pairs of leaves with labels in L'. For an L-partition $\mathcal{P} = \{L_1, \ldots, L_r\}$ where $\mathcal{F}[L_1]$, ..., $\mathcal{F}[L_r]$ are vertex-disjoint trees in $\mathcal{F}$, we say that the L-partition $\mathcal{P}$ *induces* the subforest $\{\mathcal{F}[L_1], \ldots, \mathcal{F}[L_r]\}$ of $\mathcal{F}$. An L-partition $\mathcal{P}$ is a *c-cut label-partition* for the forest $\mathcal{F}$, where $c \geq 0$ is an integer, if there exists a minimum set E_c of c edges in $\mathcal{F}$ such that after removing the c edges in E_c, the label-partition of the resulting forest is $\mathcal{P}$. Note that the label-partition for the forest $\mathcal{F}$ is a 0-cut label-partition for $\mathcal{F}$.

An unlabeled vertex in a leaf-labeled forest $\mathcal{F}$ may have degree 2. Moreover, our algorithms will delete edges that may make an unlabeled vertex to have degree even less than 2. *Contraction* is an operation on an unlabeled vertex v of degree less than 3, defined as follows: (1) if v has degree 2, then the contraction smooths the vertex v; and (2) if v has degree less than 2, then the contraction simply removes the vertex (and the incident edge if there is one). In particular, the contraction enables us in our process to keep the leaves of our forests always labeled.

Contracting an unlabeled vertex of degree less than 2 does not change the label-partition for a forest. For contracting a degree-2 vertex v, which replaces v and its two incident edges by a new edge e', it is easy to verify that the forest obtained by removing any one of the two edges incident to v in the old forest and the forest obtained by removing the edge e' in the new forest have the same label-partition. These observations give immediately the following lemma.

Lemma 2.1.1. *Let $\mathcal{F}$ be a leaf-labeled forest over a label-set L, and let $\mathcal{F}'$ be the forest* *obtained from $\mathcal{F}$ by applying an arbitrary sequence of contractions on $\mathcal{F}$. For any integer $c \geq 0$, an L-partition $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}$ if and only if $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}'$.*

The following lemma provides a simple relation between a c-cut label-partition for a leaf-labeled forest $\mathcal{F}$ and the number of trees in $\mathcal{F}$.

Lemma 2.1.2. *Suppose that $\mathcal{P} = \{L_1, \ldots, L_r\}$ is a c-cut label-partition for a leaf-labeled forest $\mathcal{F}$ consisting of h trees. Then $r = h + c$.*

Proof. We prove the lemma by induction on c. The lemma obviously holds true for the case $c = 0$. Consider the case $c > 0$. Let E_c be a set of c edges in $\mathcal{F}$ whose removal results in a forest whose label-partition is $\mathcal{P}$. Let e be any edge in E_c. Then $\mathcal{P}$ is a $(c-1)$-cut label-partition for the forest $\mathcal{F} - e$. The forest $\mathcal{F} - e$ consists of exactly $h + 1$ trees. By the inductive hypothesis, $r = (h + 1) + (c - 1) = h + c$. This completes the proof of the lemma. $\square$

Lemma 2.1.2 directly implies the following corollary, which allows us to characterize agreement forests for two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ in terms of c-cut label-partitions for $\mathcal{F}_2$.

Corollary 2.1.3. *For an agreement forest $\mathcal{F}^* = \{T_1, \ldots, T_k\}$ for two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, where $\mathcal{F}_2$ consists of c_2 trees, the L-partition $\{\ell(T_1), \ldots, \ell(T_k)\}$ is a $(k - c_2)$-cut label-partition for $\mathcal{F}_2$.*

When a c-cut label-partition $\mathcal{P}$ for a forest is given, it is easy to find the c edges whose removal results in a forest whose label-partition is $\mathcal{P}$, as shown in the following lemma.

Lemma 2.1.4. *Let L be the label-set for a forest $\mathcal{F}$, and let $\mathcal{P} = \{L_1, \ldots, L_r\}$ be an L-partition. Let e be an edge in $\mathcal{F}$ whose removal splits a leaf-labeled tree in $\mathcal{F}$ into two leaf-labeled trees T_1 and T_2 such that no label-subset in $\mathcal{P}$ has labels in both T_1 and T_2. Then for any integer $c \geq 1$, $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}$ if and only if $\mathcal{P}$ is a $(c-1)$-cut label-partition for $\mathcal{F} - e$.*

Proof. Suppose that $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}$. Then $\mathcal{F}[L_1]$, ..., $\mathcal{F}[L_r]$ are vertex-disjoint trees in $\mathcal{F}$. By the given condition, the edge e is not on a path connecting any two leaves whose labels are in the same label-subset in $\mathcal{P}$. Thus, $\mathcal{F}[L_1]$, ..., $\mathcal{F}[L_r]$ are still vertex-disjoint trees in the forest $\mathcal{F} - e$, so $\mathcal{P}$ is a d-cut label-partition for $\mathcal{F} - e$ for some integer d. On the other hand, if $\mathcal{P}$ is a d-cut label-partition for $\mathcal{F} - e$, then obviously $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}$ for some integer $c \leq d + 1$. The equality $d = c - 1$ follows directly from Lemma 2.1.2 because the number of trees in $\mathcal{F} - e$ is exactly one larger than that in $\mathcal{F}$. □

Corollary 2.1.3 and Lemma 2.1.4 suggest a formulation of the MAF problem in terms of c-cut label-partitions. We say that an L-partition $\mathcal{P}$ *induces* an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ if the subforest induced by $\mathcal{P}$ in $\mathcal{F}_1$ and the subforest induced by $\mathcal{P}$ in $\mathcal{F}_2$ are homeomorphic. By definition, if an L-partition $\mathcal{P} = \{L_1, \ldots, L_k\}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, and if $\mathcal{F}_2$ consists of c_2 trees, then $\mathcal{P}$ is a $(k - c_2)$-cut label-partition for $\mathcal{F}_2$. This characterization and Lemma 2.1.4 also provide a convenient way for a branch-and-search process: once we know that an edge e is not on the path connecting any two leaves whose labels are in the same label-subset in the desired $(k - c_2)$-cut label-partition $\mathcal{P}$ for the forest $\mathcal{F}_2$, we can simply remove e from $\mathcal{F}_2$, and recursively construct a $(k - c_2 - 1)$-cut label-partition in the forest $\mathcal{F}_2 - e$.

Our study will be based on the following problem formulations using the above characterization.

> PARA-MAF'. Given two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, and a parameter k, is there a k'-cut label-partition for the forest $\mathcal{F}_2$ that induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, where $k' \leq k$?
>
> MAX-MAF'. Given two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, construct an L-partition $\mathcal{P}$ such that $\mathcal{P}$ is a k-cut label-partition for the forest $\mathcal{F}_2$ and that $\mathcal{P}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, with k minimized.

An L-partition $\mathcal{P}$ will be called a *solution* for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests over the label-set L if $\mathcal{P}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. The *value* of the solution $\mathcal{P}$ is c if $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}_2$. By this definition, constructing a solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$ is to find c edges in $\mathcal{F}_2$ whose removal results in a subforest that is an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2)$. In particular, by Lemma 2.1.2, a minimum-value solution for $(\mathcal{F}_1, \mathcal{F}_2)$ induces an MAF for $\mathcal{F}_1$ and $\mathcal{F}_2$.

2.2 Multicut on trees problem

In the multicut on trees problem 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).

2.2.1 Review on related research

The multicut on trees problem has applications in networking [21]. The problem is known to be NP-hard, and its optimization version can be approximated to within ratio 2 [32]. We consider the multicut on trees problem from the parameterized complexity perspective. We mention that the parameterized complexity of several graph separation problems, including variants of the multicut on trees problem, was studied with respect to different parameters by Marx in [54]. Guo and Niedermeier [34] showed that the multicut on trees 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 $f(k) \cdot p(n)$, where $p(n)$ is a polynomial in the input length n.) They also showed that multicut on trees has an exponential-size kernel. Recently, Bousquet, Daligault, Thomassé, and Yeo, improved the upper bound on the kernel size for multicut on trees problem to $O(k^6)$ [13].

2.2.2 Definitions and problem reformulations

We assume familiarity with basic graph theory and parameterized complexity notation and terminology. For more information, we refer the reader to [25, 29, 58, 68].

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 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.

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 of size at most k for R. Since some of the folklore operations performed on the tree end up cutting edges from the tree, researchers who work on the multicut in trees problem consider a generalization of the problem to forests. We follow suit and define the following problem:

> MULTICUT. Given a forest $\mathcal{F}$, a set of requests $R \subseteq V(\mathcal{F}) \times V(\mathcal{F})$ and a parameter k, is there a cut of size at most k for R?

Let $(\mathcal{F}, R, k)$ be an instance of multicut on trees problem, 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 $u\pi(u)$. 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

[1]We note that we differ from the terminology used in [13]. What we call good leaves are called bad

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 C is a vertex cover for G then the edge-set $E_C = \{uw \in E(\mathcal{F}) \mid w \in C\}$, which has the same cardinality as C, cuts every request between a pair of good leaves attached to the same vertex in $\mathcal{F}$. On the other hand, for any cut K for R, the vertices in G corresponding to the leaves in $\mathcal{F}$ that are incident to the edges in K form a vertex cover for G. It follows that the number of edges in any cut K for $\mathcal{F}$ that are incident to the leaves corresponding to the vertices in G is at least the size of a minimum vertex cover for G.

2.3 Protein complex prediction problem

2.3.1 Review on related research

In the cell, proteins usually do not act in isolation but rather work together with other proteins to form *protein complexes*. Protein complexes are functional modulars in biology which are the basis of biological processes and together they form molecular machinery that perform a vast array of biological functions [48]. It is important to develop mechanisms to identify protein complexes; however to identify protein complexes through experiments remains expensive and difficult. Thus, computational techniques are used to predict protein

leaves in [13], 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.

complexes. While a common strategy is to predict complexes from given protein-protein interaction networks [6] [44] [63] [4] [38] [50] [19] [59] [53] [57] [42] [64] [67], recent combined experimental-computational strategies utilize these techniques to construct protein complexes from purification data [33] [39] [47].

Apart from the general agreement that protein complexes form dense subgraphs in an interaction network, Spirin and Mirny [65], which leads to the strategy of first generating small dense subgraphs and either extending or merging these subgraphs to construct protein complexes [6] [50], detailed understanding of the organization of protein complexes remains inadequate. To improve the modeling of complexes, recent approaches separate the tasks of predicting a core complex and its attachment proteins [49] [72].

2.3.2 Definitions and problem reformulations

Let G be a graph which represents a protein-protein interaction network. In the protein complex prediction problem, we are supposed to find dense regions in graph G. We may formulate the protein complex prediction problem (PCPrediction) as a graph clustering problem.

PCPrediction. Given a graph G, find a set of dense subgraphs $\mathcal{C}'$ of G.

The solutions to above problem depends on the definition to the dense subgraphs of G. In fact, protein complex prediction problem is a practical problem in bioinformatics. The performance to the predicted protein complexes depends on the measures of the similarity between the known protein complexes and the predicted protein complexes. Thus, the dense subgraphs that we mention in above problem is with respect to a kind of measure to the similarity between two sets of subgraphs in G. Several measures (complex agreement measure, complex accuracy measure and protein pair agreement measure) have been proposed to evaluate the performances of predicted protein complexes. In this problem, we are given a set of protein-protein interaction networks (MIPS [56], DIP [73] and BioGRID [66]) , and for each protein-protein interaction network, we are asked to predict protein complexes such that the prediction has good performances with respect to the measures.

3. ALGORITHMS TO MAXIMUM AGREEMENT FOREST PROBLEM

In this chapter, we present our reduction rules, parameterized algorithms and approximation algorithms to maximum agreement forest problem.

3.1 Bottommost sibling sets and reduction rules

Because of Lemma 2.1.1, we will assume that there are no unlabeled vertices of degree less than 3 in a leaf-labeled forest. Moreover, if our algorithms create unlabeled vertices of degree less than 3 during their processing, then we will immediately contract these vertices and work on the resulting forests without unlabeled vertices of degree less than 3.

A tree is a *single-vertex tree* if it consists of a single vertex, which is a leaf of the tree. A tree is a *single-edge tree* if it consists of a single edge, which contains two leaves and no non-leaf vertices. The *parent* of a leaf v in a tree with at least three vertices is the unique non-leaf vertex adjacent to v.

Two leaves in a forest are *siblings* if they either share the same parent, or are the two leaves of a single-edge tree. A *sibling set* is a set of leaves that are all siblings. A *bottommost sibling set* (abbr. *BSS*) is a maximal sibling set X such that either the degree of their parent is at most $|X| + 1$, or X is the leaf set of a single-edge tree. By definition, the leaf of a single-vertex tree is *not* a BSS. Moreover, by our assumption, an unlabeled vertex has degree at least 3. Thus, a BSS contains at least two leaves, and a leaf-labeled tree that is not single-vertex must contain a BSS.

In the rest of this section, we fix two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, and consider their agreement forests. Note that if the contraction is not applicable on an agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$, then each vertex in $\mathcal{F}^*$ corresponds to a unique vertex in $\mathcal{F}_1$ as well as to a unique vertex in $\mathcal{F}_2$. Therefore, in this case, it makes sense to say that two vertices in $\mathcal{F}^*$ are "adjacent in $\mathcal{F}_1$," or that a vertex v in $\mathcal{F}^*$ is "the parent of a leaf w in $\mathcal{F}_2$." Moreover, because of the one-to-one mapping between the leaf set and the label-set of a leaf-labeled forest, we can conveniently refer to a leaf by its label, without any confusions. For example, we may say that "a label ℓ is in the tree T in the forest $\mathcal{F}$," or that "the parent of the label ℓ is the vertex v."

If $\mathcal{F}_1$ consists of only single-vertex trees, then $\mathcal{F}_1$ itself is the only agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. Therefore, in the following discussion, we assume that $\mathcal{F}_1$ contains at least one tree that is not a single-vertex tree. Thus, $\mathcal{F}_1$ always contains a BSS.

Lemma 3.1.1. *Let X_1 be a BSS in $\mathcal{F}_1$, and let $\mathcal{P}$ be a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$. Then $\mathcal{P}$ has at most one label-subset L_i that intersects $\ell(X_1)$ and $|L_i| > 1$.*

Proof. Let $\mathcal{F}^*$ be the agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ that is induced by $\mathcal{P}$. Let $x \in X_1$ such that $\ell(x) \in L_i$ and $|L_i| > 1$ for a label-subset L_i in $\mathcal{P}$. If X_1 is the leaf set of a single-edge tree in $\mathcal{F}_1$, then we must have $\ell(X_1) = L_i$ and the lemma is proved. If X_1 is not the leaf set of a single-edge tree in $\mathcal{F}_1$, then since $|L_i| > 1$, the subset $\mathcal{F}_1[L_i]$ must contain the parent of x in $\mathcal{F}_1$. Thus, the only way to make a sibling $x' \in X_1$ of x in $\mathcal{F}_1$ to be in a disjoint subtree $\mathcal{F}_1[L_j]$, where $j \neq i$, is to cut the edge between x' and its parent, i.e., $L_j = \{\ell(x')\}$ and $|L_j| = 1$. □

Let $\mathcal{P}$ be an L-partition, and let Y be a set of leaves in $\mathcal{F}_2$. Denote by $\mathcal{P}_Y$ the L-partition that consists of all label-subsets in $\mathcal{P}$ that do not intersect $\ell(Y)$, plus a label-subset that is the union of all label-subsets in $\mathcal{P}$ that intersect $\ell(Y)$.

Lemma 3.1.2. *Let X_1 be a BSS in the forest $\mathcal{F}_1$, and let Y be a sibling set in the forest $\mathcal{F}_2$ with $\ell(Y) \subseteq \ell(X_1)$. Let $\mathcal{P}$ be a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ such that a label-subset in $\mathcal{P}$ contains at least two labels in $\ell(Y)$. Then, the L-partition $\mathcal{P}_Y$ is also a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$.*

Proof. If $|Y| \leq 2$, then $\mathcal{P} = \mathcal{P}_Y$ and the lemma is obvious. Thus, we can assume that neither X_1 nor Y is the leaf set of a single-edge tree. Suppose that the agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ induced by $\mathcal{P} = \{L_1, \ldots, L_k\}$ is $\mathcal{F}^* = \{T_1, \ldots, T_k\}$, where $\ell(T_i) = L_i$ for all i, and the tree T_1 contains two labels ℓ_0 and ℓ_1 in $\ell(Y)$. Then the parent of ℓ_0 and ℓ_1 in $\mathcal{F}_1$ is in the subtree $\mathcal{F}_1[L_1]$, and the parent of ℓ_0 and ℓ_1 in $\mathcal{F}_2$ is in the subtree $\mathcal{F}_2[L_1]$. By

Lemma 3.1.1, any other tree in $\mathcal{F}^*$, say T_2, that contains labels in $\ell(Y)$ must be a single-vertex tree whose unique label ℓ_2 is in $\ell(Y)$. Thus, the label subset $L_1 \cup L_2 = L_1 \cup \{\ell_2\}$ induces a subtree T_{1+2}, in both $\mathcal{F}_1$ and $\mathcal{F}_2$, which is the tree T_1 plus a new leaf labeled ℓ_2 attached to the parent of ℓ_0 and ℓ_1. The tree T_{1+2} intersects no other trees $\mathcal{F}_1[L_i]$ and $\mathcal{F}_2[L_j]$ in $\mathcal{F}_1$ and $\mathcal{F}_2$, for $i, j \neq 1, 2$. Therefore, the L-partition $\mathcal{P}' = \{L_1 \cup L_2, L_3, \ldots, L_k\}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. Repeating this process shows that the L-partition $\mathcal{P}_Y$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. $\square$

The following lemma shows that a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ is symmetric with respect to two labels when certain conditions are enforced.

Lemma 3.1.3. *Let X_1 be a BSS in $\mathcal{F}_1$, and let Y be a sibling set in $\mathcal{F}_2$ with $\ell(Y) \subseteq \ell(X_1)$. For any solution $\mathcal{P}$ for $(\mathcal{F}_1, \mathcal{F}_2)$, swapping two labels of $\ell(Y)$ in $\mathcal{P}$ also results in a solution for $(\mathcal{F}_1, \mathcal{F}_2)$.*

Proof. Let $\mathcal{F}^*$ be the agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ induced by $\mathcal{P}$. Let y and y' be in Y whose labels are in different label-subsets in $\mathcal{P}$. By Lemma 3.1.1, at least one of $\ell(y)$ and $\ell(y')$, say $\ell(y)$, is in a single-vertex tree T_i in $\mathcal{F}^*$, where the tree T_i is obtained from $\mathcal{F}_1$ and $\mathcal{F}_2$ by removing the edge incident to $\ell(y)$. Since the labels $\ell(y)$ and $\ell(y')$ are symmetric in the forests $\mathcal{F}_1$ and $\mathcal{F}_2$ ($\ell(y)$ and $\ell(y')$ are siblings in both $\mathcal{F}_1$ and $\mathcal{F}_2$), removing the edge incident to $\ell(y)$ and removing the edge incident to $\ell(y')$ in $\mathcal{F}_1$ or in $\mathcal{F}_2$ result in exactly the same forest structure, with only the labels $\ell(y)$ and $\ell(y')$ swapped. Therefore, if the L-partition $\mathcal{P}$ induces the agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$, then by swapping $\ell(y)$ and $\ell(y')$ in $\mathcal{P}$, which corresponds to replacing the removal of the edge incident to $\ell(y)$ with the removal of the edge incident to $\ell(y')$, we get an L-partition that still induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. $\square$

Now we are able to state our main result in this section, which will play an important role in our algorithms for the maximum agreement forest problems.

Theorem 3.1.4. *Let X_1 be a BSS in $\mathcal{F}_1$ and let Y be a sibling set in $\mathcal{F}_2$ with $\ell(Y) \subseteq \ell(X_1)$. For any $u_0 \in Y$, there is a maximum agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$ such that either (1) all labels in $\ell(Y)$ are in a single tree in $\mathcal{F}^*$, or (2) each label $\ell(u)$ in $\ell(Y)$, where $u \neq u_0$, is in a single-vertex tree in $\mathcal{F}^*$.*

Proof. Let $\mathcal{P}$ be the label-partition for a maximum agreement forest $\mathcal{F}'$ for $\mathcal{F}_1$ and $\mathcal{F}_2$. If a tree in $\mathcal{F}'$ contains more than one label in $\ell(Y)$, then by Lemma 3.1.2, the L-partition $\mathcal{P}_Y$ also induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, where $\mathcal{P}_Y$ is the L-partition that consists of all label-subsets in $\mathcal{P}$ that do not intersect $\ell(Y)$, plus a label-subset that is the union of all label-subsets in $\mathcal{P}$ that intersect $\ell(Y)$. The agreement forest induced by $\mathcal{P}_Y$ is obviously maximum and satisfies the condition (1) in the theorem.

If each tree in $\mathcal{F}'$ contains at most one label in $\ell(Y)$, then by Lemma 3.1.1, at least $|Y| - 1$ labels in $\ell(Y)$ are contained in unit label-subsets in $\mathcal{P}$. If a leaf $u \neq u_0$ in Y has its label $\ell(u)$ in a label-subset in $\mathcal{P}$ that is not a unit label-subset, then we simply swap the labels $\ell(u)$ and $\ell(u_0)$ in $\mathcal{P}$. By Lemma 3.1.3, the resulting L-partition $\mathcal{P}'$ also induces an agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$, which is obviously also maximum, and satisfies the condition (2) in the theorem. □

In the following, we present two simple reduction rules on a given pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests over the same label-set L.

Reduction Rule 1. If a label ℓ is in a single-vertex tree in one of the forests $\mathcal{F}_1$ and $\mathcal{F}_2$, then remove the edge (if any) incident to the label ℓ in the other forest.

Lemma 3.1.5. *Let $(\mathcal{F}_1', \mathcal{F}_2')$ be the pair produced by Reduction Rule 1 on the pair $(\mathcal{F}_1, \mathcal{F}_2)$, then an L-partition $\mathcal{P}$ is a solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$ if and only if $\mathcal{P}$ is a solution of value c' for $(\mathcal{F}_1', \mathcal{F}_2')$, where $c' = c - 1$ if an edge in $\mathcal{F}_2$ is removed, and $c' = c$ otherwise.*

Proof. That the pairs $(\mathcal{F}_1, \mathcal{F}_2)$ and $(\mathcal{F}_1', \mathcal{F}_2')$ have the same collection of solutions follows directly from the fact that if the label ℓ is in a single-vertex tree in one of the forests $\mathcal{F}_1$

and $\mathcal{F}_2$, then the label ℓ must be in a single-vertex tree in *every* agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. The relation between the values c and c' follows from Lemma 2.1.4. $\square$

Let X be a sibling set in a forest. By *shrinking* X, we mean we delete all leaves in X, and introduce a new leaf v_X with a new label ℓ_X (e.g., we can use $\ell(X)$ for ℓ_X), and let v_X be adjacent to the common neighbor of the leaves in X if such a common neighbor exists.

Note that if the sibling set X is the leaf set of a single-edge tree, then shrinking X gives a single-vertex tree with the vertex v_X. If X is the set of all leaves of a tree with a single non-leaf (and unlabeled) vertex v, then shrinking X makes v a degree-1 vertex adjacent to the new leaf v_X, and v will be contracted so that the resulting tree becomes a single-vertex tree with the leaf v_X.

Reduction Rule 2. Let X_1 be a BSS in $\mathcal{F}_1$, and let X_2 be a set of leaves in $\mathcal{F}_2$ such that $\ell(X_1) = \ell(X_2)$. If X_2 is the leaf set of a single-edge tree in $\mathcal{F}_2$, or if X_2 is a sibling set whose parent has degree at most $|X_2| + 1$, then shrink X_1 in $\mathcal{F}_1$, and shrink X_2 in $\mathcal{F}_2$.

Note that after applying Reduction Rule 2, if a vertex v in either $\mathcal{F}_1$ or $\mathcal{F}_2$ is adjacent to the new leaf (there is at most one such vertex), then the degree of v is at most 2. In particular, if v is unlabeled, then v will be contracted.

Lemma 3.1.6. *Let $(\mathcal{F}_1', \mathcal{F}_2')$ be the pair produced by Reduction Rule 2 on the pair $(\mathcal{F}_1, \mathcal{F}_2)$, then the pair $(\mathcal{F}_1, \mathcal{F}_2)$ has a solution of value at most c if and only if the pair $(\mathcal{F}_1', \mathcal{F}_2')$ has a solution of value at most c.*

Proof. Suppose that the pair $(\mathcal{F}_1', \mathcal{F}_2')$ is obtained from the pair $(\mathcal{F}_1, \mathcal{F}_2)$ by shrinking a BSS X_1 in $\mathcal{F}_1$ and the corresponding leaf set X_2 in $\mathcal{F}_2$ into a single leaf labeled ℓ_{X_2} $(= \ell_{X_1})$.

If the pair $(\mathcal{F}_1', \mathcal{F}_2')$ has a solution $\mathcal{P}' = \{L_1', L_2', \ldots, L_r'\}$ of value c, in which the label ℓ_{X_2} is in the label-subset L_1', then obviously $\mathcal{P} = \{L_1, L_2, \ldots, L_r\}$ is a solution of value c for the pair $(\mathcal{F}_1, \mathcal{F}_2)$, where $L_i = L_i'$ for $i \neq 1$, and $L_1 = (L_1' \setminus \{\ell_{X_2}\}) \cup \ell(X_2)$.

For the other direction, suppose that the pair $(\mathcal{F}_1, \mathcal{F}_2)$ has a solution of value c. Let $\mathcal{P} = \{L_1, L_2, \ldots, L_r\}$ be a solution of minimum value c' for $(\mathcal{F}_1, \mathcal{F}_2)$, where $c' \leq c$ and $\mathcal{P}$

induces a maximum agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$. By Theorem 3.1.4, we can assume that either (1) all labels in $\ell(X_2)$ are in the same tree in $\mathcal{F}^*$, or (2) each tree in $\mathcal{F}^*$ contains at most one label in $\ell(X_2)$ and at most one tree T_0 in $\mathcal{F}^*$ containing a label ℓ_0 in $\ell(X_2)$ is not a single-vertex tree. However, case (2) cannot happen: since $|X_2| \geq 2$, by adding all labels except ℓ_0 in $\ell(X_2)$ to the tree T_0, we would get an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ that consists of fewer trees than $\mathcal{F}^*$ does (note that this can always be done, no matter if the tree T_0 is a single-vertex tree or not), contradicting the assumption that $\mathcal{F}^*$ is a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$.

Thus, all labels in $\ell(X_2)$ are in the same tree T in $\mathcal{F}^*$. Moreover, by the conditions enforced on the sets X_1 and X_2, all leaves in T with labels in $\ell(X_2)$ are siblings, and if they have a common neighbor v, then v has degree at most $|X_2|+1$. With all these observations, we derive that if we shrink the leaves with labels in $\ell(X_2)$ in $\mathcal{F}^*$, we will get a forest $\mathcal{F}^{**}$ that is an agreement forest for $\mathcal{F}_1'$ and $\mathcal{F}_2'$, and the label-partition $\mathcal{P}''$ of $\mathcal{F}^{**}$ is a solution of value $c' \leq c$ for the pair $(\mathcal{F}_1', \mathcal{F}_2')$. $\square$

The following theorem follows directly from the definitions of Reduction Rules 1-2.

Theorem 3.1.7. *For a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests on which Reduction Rules 1-2 are not applicable, (1) a label is in a single-vertex tree in $\mathcal{F}_1$ if and only if it is in a single-vertex tree in $\mathcal{F}_2$; and (2) for any sibling set X_2 in $\mathcal{F}_2$ such that the set X_1 in $\mathcal{F}_1$ with $\ell(X_1) = \ell(X_2)$ is a BSS, the siblings in X_2 must have a parent and the parent has degree at least $|X_2| + 2$.*

3.2 MAF is fixed-parameter tractable

In this section, we present a parameterized algorithm for the PARA-MAF' problem, and show that the problem, thus also the original PARA-MAF problem, are fixed-parameter tractable.

Let $(\mathcal{F}_1, \mathcal{F}_2; k)$ be an instance of the PARA-MAF' problem, for which we look for a solution of value not larger than k for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests. Because of Lemmas 3.1.5 and 3.1.6, during our process, we will exhaustively apply Reduction Rules 1-

2 on the pair of forests whenever the rules are applicable, and work on the reduced instance (note that by Lemma 3.1.5, in case we apply Reduction Rule 1 that removes an edge in the forest $\mathcal{F}_2$, we will also decrease the parameter k by 1). An instance is *strongly reduced* if these reduction rules are not applicable on the corresponding pair of forests. Therefore, throughout the discussion, we will assume that our instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ is always a strongly reduced instance.

If all trees in $\mathcal{F}_1$ are single-vertex, then by Theorem 3.1.7, all trees in $\mathcal{F}_2$ are also single-vertex, and the problem becomes trivial: $\mathcal{F}_1 = \mathcal{F}_2$ is the unique agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$.

Thus, we can assume that $\mathcal{F}_1$ contains a tree that is not single-vertex. As a consequence, there is a BSS X_1 in $\mathcal{F}_1$ such that $|X_1| \geq 2$. Let X_2 be the leaf set in $\mathcal{F}_2$ such that $\ell(X_2) = \ell(X_1)$.

Our algorithm is based on a branch-and-bound process. We will introduce a set of branching rules. A branching rule is *safe* if the branching rule applied on an instance I for PARA-MAF' produces a set $\mathcal{S}$ of instances for PARA-MAF' such that I is a yes-instance if and only if at least one of the instances in $\mathcal{S}$ is a yes-instance. We say that a branching rule satisfies the recurrence relation $T(k) = T(k_1) + \cdots + T(k_r)$ if on an instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ of PARA-MAF', the branching rule produces r instances $(\mathcal{F}_{1,1}, \mathcal{F}_{1,2}; k_1)$, …, $(\mathcal{F}_{r,1}, \mathcal{F}_{r,2}; k_r)$ for the problem. Moreover, we assume that the function $T(k)$ is non-decreasing. Therefore, if $k \leq k'$ then $T(k) \leq T(k')$.

Case 1. Leaves in X_2 are not in the same tree in $\mathcal{F}_2$.

Branching Rule 1. Let u_2 and v_2 be two leaves in X_2 that are in different trees in $\mathcal{F}_2$, then decrease k by 1, and branch into two ways: $[W1]$ cut u_2 in $\mathcal{F}_2$; and $[W2]$ cut v_2 in $\mathcal{F}_2$.

Lemma 3.2.1. *Branching Rule 1 is safe, and satisfies the recurrence relation $T(k) = 2T(k-1)$.*

Proof. That the branching rule satisfies the recurrence relation $T(k) = 2T(k-1)$ follows directly from the rule. Let u_1 and v_1 be two leaves in $\mathcal{F}_1$ such that $\ell(u_1) = \ell(u_2)$ and

$\ell(v_1) = \ell(v_2)$. Since u_2 and v_2 are in different trees in $\mathcal{F}_2$, the two labels $\ell(u_2)$ and $\ell(v_2)$ must be in different trees in an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. Therefore, in order to construct an agreement forest from $\mathcal{F}_1$ for $\mathcal{F}_1$ and $\mathcal{F}_2$, we must remove edges in $\mathcal{F}_1$ to separate the leaves u_1 and v_1 into two different trees. Because u_1 and v_1 are siblings, the only way to separate the leaves u_1 and v_1 into two different trees is to either remove the edge incident to u_1 or remove the edge incident to v_1. Each of these edge removals makes one of u_1 and v_1 in a single-vertex tree. Therefore, in an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, either u_2 or v_2 must be in a single-vertex tree. As a consequence, at least one of the branching ways $[W1]$ and $[W2]$ in the branching rule must correctly remove an edge. Moreover, by Lemma 2.1.4, a k'-cut label-partition for $\mathcal{F}_2$ that induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, where $k' \leq k$, will become a $(k'-1)$-cut label-partition for the forest obtained from $\mathcal{F}_2$ by the correct branch of the branching rule. □

Case 2. X_2 is a sibling set in $\mathcal{F}_2$.

Because the instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ is strongly reduced, by Theorem 3.1.7, X_2 has a parent p_2 and the degree of p_2 is at least $|X_2| + 2$.

Branching Rule 2. If X_2 is a sibling set, fix a leaf u_2 in X_2, and let $E_2 = \{e_1, \ldots, e_h\}$ be the set of edges that are incident to the parent p_2 of X_2 but not to any leaf in X_2, $h \geq 2$. Then branch into $h+1$ ways: $[W(0)]$ remove all edges incident to X_2 except the one incident to u_2 and decrease k by $|X_2| - 1$; and, for each $1 \leq i \leq h$, $[W(i)]$ remove all edges in E_2 except e_i and decrease k by $h-1$.

Lemma 3.2.2. *Branching Rule 2 is safe, and satisfies the recurrence relation $T(k) = T(k - (|X_2| - 1)) + hT(k - (h-1))$.*

Proof. Again that Branching Rule 2 satisfies the recurrence relation $T(k) = T(k - (|X_2| - 1)) + hT(k - (h-1))$ follows directly from the branching rule.

By Theorem 3.1.4, we can assume that for a maximum agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$, either all labels in $\ell(X_2)$ are in the same tree in $\mathcal{F}^*$, or each leaf in $X_2 \setminus \{u_2\}$

has its label in a single-vertex tree in $\mathcal{F}^*$. In case each leaf in $X_2 \setminus \{u_2\}$ has its label in a single-vertex tree in $\mathcal{F}^*$, the branching way $[W(0)]$ correctly removes the edges.

Suppose that all labels in $\ell(X_2)$ are in the same tree in $\mathcal{F}^*$. Then all labels in $\ell(X_2)$ must be siblings in $\mathcal{F}^*$. Because the parent p_1 of X_1 has degree at most $|X_1|+1$, the parent p^* of the labels in $\ell(X_2)$ in the forest $\mathcal{F}^*$ has its degree at most $|X_2|+1$. Therefore, in the forest $\mathcal{F}_2$, all edges in E_2 except at most one must be removed in order to construct $\mathcal{F}^*$ from $\mathcal{F}_2$. Thus, in this case, one of the branching ways $[W(i)]$ correctly removes the edges. Finally, it is easy to verify that Branching Rule 2 decreases the parameter value k correctly. □

Case 3. X_2 contains a sibling set Y_2 with $|Y_2| \geq 2$.

We consider this case when Cases 1-2 do not apply. Therefore, the set X_2 contains a leaf z_2 that is not a sibling of the leaves in Y_2, but z_2 and Y_2 are in the same tree. Fix a leaf u_2 in Y_2. Let P be the path in $\mathcal{F}_2$ that connects the parent p_2 of Y_2 and the leaf z_2. Let E_2 be the set of edges that are not on the path P but are adjacent to a vertex $w \neq p_2$ on the path P. See Figure 3.1(A) for an illustration of this situation. Note that the edge set E_2 cannot be empty because the path P consists of at least three vertices and by our assumption of the contraction operation, no unlabeled vertex has degree less than 3.

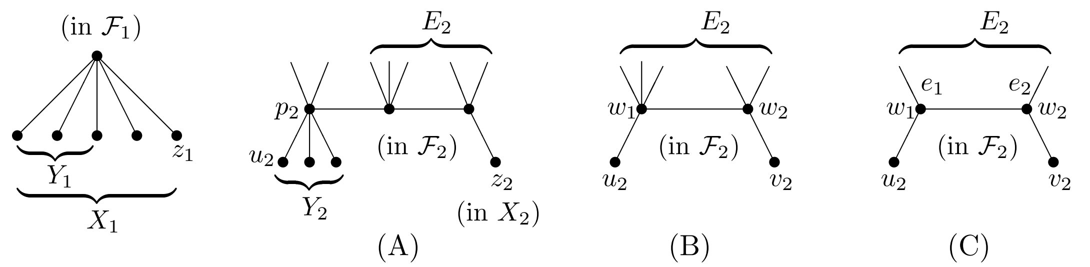


Figure 3.1: (A) Situation for case 3, (B) Situation for subcase 4.2, (C) Situation for subcase 4.3.

Branching Rule 3. In the situation of Figure 3.1(A), branch into three ways: $[W1]$ cut all leaves in Y_2 except u_2 in $\mathcal{F}_2$ and decrease k by $|Y_2| - 1$; $[W2]$ cut z_2 in $\mathcal{F}_2$ and decrease k by 1; and $[W3]$ cut all edges in E_2 in $\mathcal{F}_2$ and decrease k by $|E_2|$.

Lemma 3.2.3. *Branching Rule 3 is safe, and satisfies the recurrence relation $T(k) \leq 3T(k-1)$.*

Proof. Let Y_1 be the subset of the leaf set X_1 in $\mathcal{F}_1$ such that $\ell(Y_1) = \ell(Y_2)$, and let z_1 be the leaf in X_1 such that $\ell(z_1) = \ell(z_2)$. See Figure 3.1.

By Theorem 3.1.4, there is a maximum agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$ such that either (3.1) all labels in $\ell(Y_2)$ are in the same tree in $\mathcal{F}^*$, or (3.2) each leaf in $Y_2 \setminus \{u_2\}$ has its label in a single-vertex tree in $\mathcal{F}^*$. The branching way $[W1]$ correctly cuts edges for subcase (3.2).

There are two different possibilities for subcase (3.1).

Subsubcase 3.1.1: The label $\ell(z_2)$ and the label set $\ell(Y_2)$ are not in the same tree in $\mathcal{F}^*$. Since the leaf z_1 with $\ell(z_1) = \ell(z_2)$ and the set Y_1 with $\ell(Y_1) = \ell(Y_2)$ in $\mathcal{F}_1$ are siblings, in order to construct the agreement forest $\mathcal{F}^*$ from the forest $\mathcal{F}_1$, we must separate the leaf z_1 from the set Y_1 and keep the set Y_1 in the same tree. The only way to do this is to cut the leaf z_1 (note that $|Y_1| \geq 2$), which makes the label $\ell(z_1) = \ell(z_2)$ in a single-vertex tree in $\mathcal{F}^*$. Therefore, in this case, the branching way $[W2]$ correctly cuts the edge.

Subsubcase 3.1.2: The label $\ell(z_2)$ and $\ell(Y_2)$ are in the same tree in $\mathcal{F}^*$. Since Y_1 and z_1 are siblings in $\mathcal{F}_1$, if we want to keep the label $\ell(z_1) = \ell(z_2)$ and the label set $\ell(Y_1) = \ell(Y_2)$ in the same tree in $\mathcal{F}^*$, then $\ell(z_1)$ and $\ell(Y_1)$ must be siblings in $\mathcal{F}^*$. Therefore, in order to construct the agreement forest $\mathcal{F}^*$ from the forest $\mathcal{F}_2$ and make $\ell(z_2)$ and $\ell(Y_2)$ become siblings, we must cut all edges in E_2. Thus, the branching way $[W3]$ correctly handles this case.

The recurrence relation for Branching Rule 3 is $T(k) = T(k-(|Y_2|-1)) + T(k-1) + T(k-|E_2|)$. Since $|Y_2| \geq 2$, $|E_2| \geq 1$, and $T(k)$ is a non-decreasing, we have $T(k) \leq 3T(k-1)$. $\square$

Case 4. No two leaves in X_2 are siblings in $\mathcal{F}_2$.

If none of Cases 1-3 apply, then all leaves in the set X_2 are in the same tree in the forest $\mathcal{F}_2$ and no two leaves in X_2 are siblings. We split this case into three subcases depending on the size $|X_1|$ and the number of unlabeled vertices on a path connecting two leaves in X_2. Let u_2 and v_2 be two arbitrary leaves in X_2, and let P be the path in $\mathcal{F}_2$ that connects u_2 and v_2. Denote by $int(P) = \{w_1, \ldots, w_h\}$ the set of vertices on P that are unlabeled, $h \geq 2$.

Subcase 4.1: The path P consists of at least five vertices, i.e., $h \geq 3$.

Branching Rule 4.1 Let u_2 and v_2 be leaves in X_2 such that the path $P = \{u_2, w_1, \ldots, w_h, v_2\}$ in $\mathcal{F}_2$ satisfies $h \geq 3$. Then branch into $(h+2)$ ways: $[W1]$ cut u_2 in $\mathcal{F}_2$ and decrease k by 1; $[W2]$ cut v_2 in $\mathcal{F}_2$ and decrease k by 1; and, for each $1 \leq i \leq h$, $[W(2+i)]$ cut the edges incident to $int(P) \setminus \{w_i\}$ but not on the path P, and decrease k by the number of edges cut.

Lemma 3.2.4. *Branching Rule 4.1 is safe, and satisfies the recurrence relation $T(k) \leq 2T(k-1) + hT(k-(h-1))$.*

Proof. Let u_1 and v_1 be the leaves in the set X_1 in the forest $\mathcal{F}_1$, with $\ell(u_1) = \ell(u_2)$ and $\ell(v_1) = \ell(v_2)$. Let $\mathcal{F}^*$ be a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$.

If $\ell(u_2)$ and $\ell(v_2)$ are not in the same tree in $\mathcal{F}^*$, then to construct $\mathcal{F}^*$ from the forest $\mathcal{F}_1$, we need to separate the leaves u_1 and v_1 in X_1 into different trees. Since u_1 and v_1 are siblings, the only way to separate u_1 and v_1 is to either cut the edge incident to u_1 or cut the edge incident to v_1. Each of these makes one of u_1 and v_1 a single-vertex tree. Thus, in this case, either label $\ell(u_2) = \ell(u_1)$ or label $\ell(v_2) = \ell(v_1)$ is in a single-vertex tree in $\mathcal{F}^*$. The branching ways $[W1]$ and $[W2]$ correctly handle these two cases in $\mathcal{F}_2$.

If the labels $\ell(u_2)$ and $\ell(v_2)$ are in the same tree in $\mathcal{F}^*$, then since u_1 and v_1 are siblings in $\mathcal{F}_1$, the labels $\ell(u_2)$ and $\ell(v_2)$ must also be siblings in $\mathcal{F}^*$. Therefore, in this case, in order to construct the agreement forest $\mathcal{F}^*$ from the forest $\mathcal{F}_2$, for all unlabeled vertices w' in $int(P)$, except at most one w_i, we must cut all edges incident to w' but not on the path P, so that the labels $\ell(u_2)$ and $\ell(v_2)$ can become siblings in $\mathcal{F}^*$. This case is correctly handled by branching way $[W(2+i)]$ in the branching rule.

For each i for the branching way $W(2+i)$, we cut $h_i = \sum_{j \neq i} |E'_j|$ edges, where E'_j is the set of edges in $\mathcal{F}_2$ that are incident to w_j but not on the path P. Thus, the recurrence relation for the branching rule is $T(k) = 2T(k-1) + \sum_i T(k-h_i)$. Since no unlabeled vertex has degree less than 3, $|E'_j| \geq 1$ for all j, and $h_i \geq h-1$ for all i. Since $T(k)$ is non-decreasing, we have $T(k) \leq 2T(k-1) + hT(k-(h-1))$. □

Note that if $|X_2| \geq 3$ and no two leaves in X_2 are siblings, then there are always two leaves u_2 and v_2 in X_2 such that the path connecting u_2 and v_2 in $\mathcal{F}_2$ consists of at least 5 vertices. In this case, Subcase 4.1 is always applicable. Therefore, in the following, we will assume that the set X_2 contains exactly two leaves u_2 and v_2, the path P connecting u_2 and v_2 consists of exactly 4 vertices, of which two are unlabeled, and $int(P) = \{w_1, w_2\}$. Let E_2 be the set of edges that are incident to either w_1 or w_2 but not on the path P.

Subcase 4.2: $E_2 = \{e_1, \ldots, e_h\}$ **with** $h \geq 3$**, see Figure 3.1(B).**

Branching Rule 4.2 Under the conditions of Subcase 4.2, branch into $(2+h)$ ways: $[W1]$ cut u_2 in $\mathcal{F}_2$ and decrease k by 1; $[W2]$ cut v_2 in $\mathcal{F}_2$ and decrease k by 1; and, for $1 \leq i \leq h$, $[W(2+i)]$ for the edge e_i in E_2, cut all edges in E_2 except e_i and decrease k by $h-1$.

Lemma 3.2.5. *Branching Rule 4.2 is safe, and satisfies the recurrence relation* $T(k) = 2T(k-1) + hT(k-(h-1))$.

Proof. Let u_1 and v_1 be the leaves in the set X_1 in the forest $\mathcal{F}_1$, with $\ell(u_1) = \ell(u_2)$ and $\ell(v_1) = \ell(v_2)$. Let $\mathcal{F}^*$ be a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$.

If the labels $\ell(u_2)$ and $\ell(v_2)$ are not in the same tree in $\mathcal{F}^*$, then, similar to Subcase 4.1, since u_1 and v_1 are siblings in $\mathcal{F}_1$, at least one of the labels $\ell(u_2)$ and $\ell(v_2)$ must be in a single-vertex tree in $\mathcal{F}^*$. The branching ways $[W1]$ and $[W2]$ correctly handle these cases in $\mathcal{F}_2$.

Suppose that $\ell(u_2)$ and $\ell(v_2)$ are in the same tree T in $\mathcal{F}^*$. If T is a single-edge tree, then all edges in E_2 should be removed in $\mathcal{F}_2$ to make $\ell(u_2)$ and $\ell(v_2)$ induce a single-edge tree in $\mathcal{F}^*$. If $\ell(u_2)$ and $\ell(v_2)$ have a parent p^* in $\mathcal{F}^*$, then p^* must correspond to the parent p_1 of u_1 and v_1 in $\mathcal{F}_1$. Since u_1 and v_1 are the only children of p_1 and since X_1 is a BSS, the vertex p_1 has degree at most 3 in $\mathcal{F}_1$. As a consequence, the vertex p^* has degree at most 3 in $\mathcal{F}^*$. Therefore, all edges in E_2 except at most one must be removed when we construct the agreement forest $\mathcal{F}^*$ from $\mathcal{F}_2$. These cases are correctly handled by branching ways $[W(2+i)]$.

Branching Rule 4.2 obviously satisfies the recurrence relation $T(k) = 2T(k-1)+hT(k-(h-1))$. □

Since the leaves u_2 and v_2 in $\mathcal{F}_2$ are not siblings, the path P connecting u_2 and v_2 has at least two unlabeled vertices. Since each unlabeled vertex has degree at least 3, we must have $|E_2| \geq 2$. As a consequence, only the case where $|E_2| = 2$ is not covered by the above cases.

Subcase 4.3: $E_2 = \{e_1, e_2\}$**, see Figure 3.1(C).**

Branching Rule 4.3 Under the conditions of Subcase 4.3, decrease k by 1, and branch into three ways: $[W0]$ cut u_2 in $\mathcal{F}_2$; $[W1]$ cut e_1 in E_2; and $[W2]$ cut e_2 in E_2.

Lemma 3.2.6. *Branching Rule 4.3 is safe, and satisfies the recurrence relation* $T(k) = 3T(k-1)$.

Proof. Let u_1 and v_1 be the leaves in the set X_1 in the forest $\mathcal{F}_1$, with $\ell(u_1) = \ell(u_2)$ and $\ell(v_1) = \ell(v_2)$. Let $\mathcal{F}^*$ be a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$.

If $\ell(u_2)$ and $\ell(v_2)$ are in the same tree in $\mathcal{F}^*$, then, as for Subcase 4.2, at least one of the two edges e_1 and e_2 in E_2 should be removed in order to construct the agreement forest $\mathcal{F}^*$ from $\mathcal{F}_2$. These cases are handled correctly by the branching ways $[W1]$ and $[W2]$.

If $\ell(u_2)$ and $\ell(v_2)$ are in different trees in $\mathcal{F}^*$, then since u_1 and v_1 are siblings in $\mathcal{F}_1$, at least one of $\ell(u_2)$ and $\ell(v_2)$ is in a single-vertex tree in $\mathcal{F}^*$. If u_2 is in a single-vertex tree in $\mathcal{F}^*$, then branching way $[W0]$ removes the correct edge. If v_2 is in a single-vertex tree in $\mathcal{F}^*$, then since the two internal vertices of the path P have degree exactly 3 (see Figure 3.1(C)), cutting v_2 and cutting u_2 give two leaf-labeled forests that are almost homeomorphic except that the labels $\ell(u_2)$ and $\ell(v_2)$ are swapped. Therefore, swapping the labels $\ell(u_2)$ and $\ell(v_2)$ in the label-partition for $\mathcal{F}^*$ gives an L-partition that also induces a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, in which the label $\ell(u_2)$ is in a single-vertex tree. In summary, in case $\ell(u_2)$ and $\ell(v_2)$ are in different trees in $\mathcal{F}^*$, it is always safe to cut the leaf u_2 in $\mathcal{F}_2$ when we construct a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. The branching way $[W0]$ correctly handles this case.

Branching Rule 4.3 obviously satisfies the recurrence relation $T(k) = 3T(k-1)$. $\square$

An instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ of PARA-MAF' with $k \leq 0$ can be easily handled: if $k < 0$ then it is a no-instance; and if $k = 0$ then it is a yes-instance if and only if $\mathcal{F}_2$ is a subforest of $\mathcal{F}_1$. Also, as we remarked before, if the forest $\mathcal{F}_1$ consists of only single-vertex trees, then the maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ is just $\mathcal{F}_1$ itself, and it is again easy to decide if $(\mathcal{F}_1, \mathcal{F}_2; k)$ is a yes-instance. On the other hand, if $\mathcal{F}_1$ contains a tree that is not a single-vertex tree, then $\mathcal{F}_1$ contains a BSS X_1 of at least two leaves. Under the assumptions that the forest $\mathcal{F}_1$ contains a BSS X_1 and that the contraction operation and reduction rules are applied whenever they are applicable, Cases 1-4 above cover all possible cases for a given instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ for the PARA-MAF' problem. Therefore, our parameterized algorithm just proceeds each of the cases and applies the corresponding branching rule, as given in Figure 3.2.

We need the following lemma for the recurrence relations for the analysis of our algorithm.

Lemma 3.2.7. *Let $T_h(k)$ be a non-decreasing and positive-valued function satisfying the recurrence relation $T_h(k) = 2T_h(k-1) + hT_h(k-(h-1))$, where $h \geq 3$ is a constant. Then $2^k \leq T_h(k) \leq 3^k$.*

Proof. Because $hT_h(k-(h-1)) \geq 0$, we have $T_h(k) \geq 2T_h(k-1)$, which gives $T_h(k) \geq 2^k$.

```
Parameterized Algorithm Para-MAF
INPUT: two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$, and a parameter $k$
OUTPUT: a solution of value at most $k$ for $(\mathcal{F}_1, \mathcal{F}_2)$.
/** assume $(\mathcal{F}_1, \mathcal{F}_2; k)$ is strongly reduced, and contraction is not applicable on $\mathcal{F}_1$ and $\mathcal{F}_2$.
1   if $k \leq 0$ or $\mathcal{F}_1$ consists of only single-vertex trees then solve the problem directly;
2   Let $X_1$ be a BSS in $\mathcal{F}_1$, and let $X_2$ be the leaf set in $\mathcal{F}_2$ such that $\ell(X_1) = \ell(X_2)$;
3   switch
3.1     Case 1:    apply Branching Rule 1, and recursively work on the resulting instances;
3.2     Case 2:    apply Branching Rule 2, and recursively work on the resulting instances;
3.3     Case 3:    apply Branching Rule 3, and recursively work on the resulting instances;
3.4     Case 4.1:  apply Branching Rule 4.1, and recursively work on the resulting instances;
3.5     Case 4.2:  apply Branching Rule 4.2, and recursively work on the resulting instances;
3.6     Case 4.3:  apply Branching Rule 4.3, and recursively work on the resulting instances.
```

Figure 3.2: A parameterized algorithm for PARA-MAF'

We prove the other direction by induction on h. For the base case $h = 3$, the recurrence relation becomes $T_3(k) = 2T_3(k-1) + 3T_3(k-2)$. Using the standard technique in parameterized computation [25], we get $T_3(k) = 3^k$.

For general $h \geq 3$, let $T_h(k) = c_h^k$. By the recurrence relation, $c_h^k = 2c_h^{k-1} + hc_h^{k-(h-1)}$. Simplifying it gives $(c_h - 2)c_h^{h-2} = h$. Replacing h by $h+1$, we get $(c_{h+1} - 2)c_{h+1}^{h-1} = h + 1$. From $(c_h - 2)c_h^{h-2} = h$ and the inductive hypothesis $2 \leq c_h \leq 3$, we get $(c_h - 2)c_h^{h-1} = hc_h \geq h + 1$. Therefore, $(c_h - 2)c_h^{h-1} \geq (c_{h+1} - 2)c_{h+1}^{h-1}$, which implies $c_h \geq c_{h+1}$. Now the inductive hypothesis $c_h \leq 3$ implies $c_{h+1} \leq 3$, which shows $T_{h+1}(k) \leq 3^k$ and the induction goes through. □

Now we are ready to analyze the algorithm **Para-MAF**. For an instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ of PARA-MAF', the execution of the algorithm is depicted by a search tree whose leaves correspond to the conclusions of the algorithm. Each internal node of the search tree corresponds to a branch based on the branching rules. Let $T(k)$ be the number of leaves in the search tree of the algorithm on the input $(\mathcal{F}_1, \mathcal{F}_2; k)$. Then the function $T(k)$ satisfies the recurrence relations given for the branching rules (note that we can assume that the function $T(k)$ is non-decreasing):

(1) By Lemma 3.2.1, Branching Rule 1 satisfies the recurrence relation $T(k) = 2T(k-1) \leq 3T(k-1)$, which satisfies $T(k) \leq 3^k$;

(2) By Lemma 3.2.2, Branching Rule 2 satisfies the recurrence relation

$$T(k) = T(k - (|X_2| - 1)) + hT(k - (h-1)) \leq T(k-1) + hT(k-(h-1)),$$

where $|X_2| \geq 2$ and $h \geq 2$. For $h = 2$, the recurrence relation becomes $T(k) \leq 3T(k-1)$, which satisfies $T(k) \leq 3^k$. For $h \geq 3$, the recurrence relation satisfies:

$$T(k) \leq T(k-1) + hT(k-(h-1)) \leq 2T(k-1) + hT(k-(h-1)).$$

By Lemma 3.2.7, the function $T(k)$ also satisfies $T(k) \leq 3^k$;

(3) By Lemma 3.2.3, Branching Rule 3 satisfies the recurrence relation $T(k) \leq 3T(k-1)$, which gives $T(k) \leq 3^k$;

(4.1-4.2) By Lemmas 3.2.4-3.2.5, Branching Rules 4.1 and 4.2 satisfy the recurrence relation $T(k) \leq 2T(k-1) + hT(k-(h-1))$, where $h \geq 3$. By Lemma 3.2.7, $T(k)$ satisfies $T(k) \leq 3^k$;

(4.3) By Lemma 3.2.6, Branching Rule 4.3 satisfies the recurrence relation $T(k) \leq 3T(k-1)$, which again gives $T(k) \leq 3^k$.

Thus, each of the branching rules in the algorithm **Para-MAF** satisfies the relation $T(k) \leq 3^k$, which implies, inductively, that the search tree of the algorithm has at most 3^k leaves. Finally, it is easy to verify that each of the contraction operation, Reduction Rules 1-2, and Branching Rules 1-3, 4.1-4.3 takes time $O(n)$, where n is the number of leaves in the input forests. This concludes our main result in this section:

Theorem 3.2.8. *The parameterized problem* PARA-MAF' *can be solved in time $O(3^k n)$, so the problem is fixed-parameter tractable.*

It is straightforward to see how the original problem PARA-MAF is solved using the algorithm **Para-MAF**, where two leaf-labeled trees T_1 and T_2 and a parameter k are given, asking if there is an agreement forest of at most k trees for T_1 and T_2. The problem can be simply regarded as an instance $(T_1, T_2; k-1)$ of the PARA-MAF' problem where we are looking for a solution of value at most $k-1$ for (T_1, T_2), i.e., a k'-cut label-partition for the forest (i.e., the tree) T_2, where $k' \leq k-1$, which induces an agreement forest (of $k'+1 \leq k$ trees) for T_1 and T_2.

Corollary 3.2.9. *The parameterized problem* PARA-MAF *is fixed-parameter tractable.*

Corollary 3.2.9 resolves an open problem posed in the literature [36, 71].

3.3 A constant-ratio approximation algorithm for MAX-MAF

The analysis in previous sections based on BSS also motivates an approximation algorithm for the MAX-MAF' problem, which is presented in this section.

Recall that the MAX-MAF' problem on a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests over the same label-set L looks for a solution of minimum value, i.e., an L-partition $\mathcal{P}$ that induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ and is a c-cut label-partition for $\mathcal{F}_2$ with the value c minimized. A solution of the minimum value will be called an *optimal solution*, whose value will be called the *optimal value* for the instance $(\mathcal{F}_1, \mathcal{F}_2)$. Recall that a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ is induced by an optimal solution for $(\mathcal{F}_1, \mathcal{F}_2)$.

Similarly, we will say that an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of MAX-MAF' is *strongly reduced* if none of the contraction operation and Reduction Rules 1-2 is applicable on the instance. We will always assume that the instances in our discussion are strongly reduced.

An *edge-removal meta-step* of an algorithm is a collection of consecutive computational steps in the algorithm that on an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of MAX-MAF' removes a set of edges in $\mathcal{F}_2$.

Definition 3.3.1. *An edge-removal meta-step M keeps ratio r if on an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of* MAX-MAF'*, M removes a set E_M of edges in $\mathcal{F}_2$ such that $|E_M| \leq r(c - c')$, where c and c' are the optimal values for the instances $(\mathcal{F}_1, \mathcal{F}_2)$ and $(\mathcal{F}_1, \mathcal{F}_2 - E_M)$, respectively.*

For example, an application of Reduction Rule 1 that removes an edge e in $\mathcal{F}_2$ is an edge-removal meta-step that keeps ratio 1 because $|E_M| = |\{e\}| = 1$, and by Lemma 3.1.5, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e)$ is one less than that for $(\mathcal{F}_1, \mathcal{F}_2)$. Also note that by definition if a meta-step neither removes edges in $\mathcal{F}_2$ nor change the optimal value for the forest pair (such as an application of Reduction Rule 1 that removes an edge in $\mathcal{F}_1$) keeps ratio r for any $r \geq 0$.

Before we present our algorithm, we observe the following simple fact:

Lemma 3.3.2. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be an instance of* MAX-MAF' *and let e_2 be any edge in $\mathcal{F}_2$. Then the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e_2)$ is not larger than the optimal value for $(\mathcal{F}_1, \mathcal{F}_2)$.*

Proof. Let $\mathcal{P}$ be an optimal solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$, where $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}_2$. Then there is a set E_2 of c edges in $\mathcal{F}_2$ such that after contraction, $\mathcal{F}_2 - E_2$ (whose label-partition is $\mathcal{P}$) is an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2)$. If $e_2 \in E_2$, then $E_2' = E_2 \setminus \{e_2\}$ is a set of $c - 1$ edges such that $(\mathcal{F}_2 - e_2) - E_2'$ is an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2 - e_2)$. That is, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e_2)$ is not larger than $|E_2'| = c - 1$. On the other hand, if $e_2 \notin E_2$, then since $\mathcal{F}_2 - E_2$ is an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2)$, removing the edge e_2 in $\mathcal{F}_2 - E_2$ also results in an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2)$. That is, $(\mathcal{F}_2 - e_2) - E_2$ is an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2 - e_2)$, and the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e_2)$ is then not larger than $|E_2| = c$. $\square$

In the rest of this section, we fix an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of the MAX-MAF' problem. As we examined in previous sections, we can assume that the instance is strongly reduced, and that the forest $\mathcal{F}_1$ contains at least one BSS X_1. Let X_2 be the leaf set in $\mathcal{F}_2$ with $\ell(X_2) = \ell(X_1)$.

As we did in the parameterized algorithm, we consider different cases based on the structure of the leaf set X_2 in $\mathcal{F}_2$. For each case, we apply a meta-step that removes a set of edges in $\mathcal{F}_2$ and we verify that the meta-step keeps a ratio bounded by 3.

Case 1. Leaves in X_2 are not in the same tree in $\mathcal{F}_2$.

Meta-Step 1. Let u_2 and v_2 be two leaves in X_2 that are in different trees in $\mathcal{F}_2$, then remove the edges incident to u_2 and v_2.

Lemma 3.3.3. *Meta-Step 1 keeps ratio 2.*

Proof. Let $\mathcal{P}$ be an optimal solution of value c for the instance $(\mathcal{F}_1, \mathcal{F}_2)$, which is a c-cut label-partition for $\mathcal{F}_2$. As examined in Lemma 3.2.1, one of the labels $\ell(u_2)$ and $\ell(v_2)$ is in a unit label-subset in $\mathcal{P}$. Without loss of generality, suppose $\ell(u_2)$ is such a label, and let e_u and e_v be the edges incident to u_2 and v_2 in $\mathcal{F}_2$, respectively. By Lemma 2.1.4, $\mathcal{P}$ is a $(c-1)$-cut label-partition for $\mathcal{F}_2 - e_u$. Also, $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - e_u)$ because u_2 is in a unit label-subset in $\mathcal{P}$. Therefore, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e_u)$ is at most $c-1$. Finally, by Lemma 3.3.2, the optimal vale c' for $(\mathcal{F}_1, \mathcal{F}_2 - \{e_u, e_v\})$ is at most $c-1$. Therefore, Meta-Step 1 removes the edge set $E_1 = \{e_u, e_v\}$ that satisfies $|E_1| \leq 2(c - c')$, where c and c' are the optimal values for $(\mathcal{F}_1, \mathcal{F}_2)$ and $(\mathcal{F}_1, \mathcal{F}_2 - E_1)$, respectively. That is, the meta-step keeps ratio 2. □

Case 2. X_2 is a sibling set in $\mathcal{F}_2$.

Because the instance $(\mathcal{F}_1, \mathcal{F}_2)$ is strongly reduced, by Theorem 3.1.7, X_2 has a parent p_2 and the degree of p_2 is at least $|X_2| + 2$. In the following two cases, let E'' be the set of edges that are incident to the parent p_2 of X_2 but not incident to the leaves in X_2. We apply the following meta-steps based on the difference between the sizes of X_2 and E''. See Figure 3.3(A).

Subcase 2.1: $|E''| > |X_2|$.

Meta-Step 2.1. If $|E''| > |X_2|$, then pick any set E_1 of $|X_2| - 1$ edges incident to the leaves in X_2, and pick any set E_2 of $|X_2|$ edges in E'', remove all edges in $E_1 \cup E_2$.

Lemma 3.3.4. *Meta-Step 2.1 keeps ratio* 3.

Proof. Let $\mathcal{P}$ be an optimal solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$, which is a c-cut label-partition for $\mathcal{F}_2$. Let $E_{1,2} = E_1 \cup E_2$. Meta-Step 2.1 totally removes $|E_{1,2}| = 2|X_2| - 1$ edges from $\mathcal{F}_2$.

By Theorem 3.1.4, we can assume that either each leaf incident to an edge in E_1 is in a unit label-subset in $\mathcal{P}$ or all labels in $\ell(X_2)$ are in the same label-subset in $\mathcal{P}$.

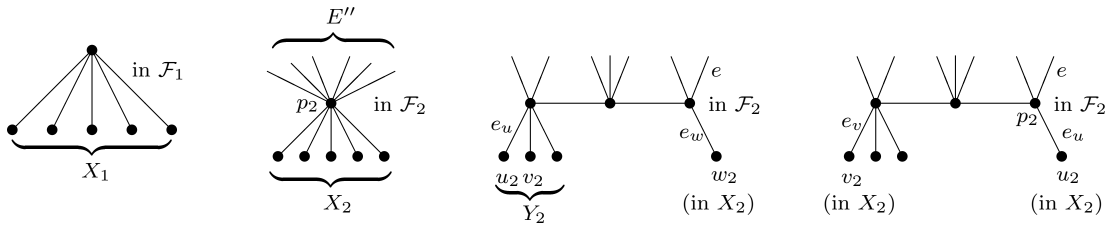
If each of the $|X_2|-1$ leaves incident to the edges in E_1 is in a unit label-subset in $\mathcal{P}$, then $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - E_1)$. Since removing each edge in E_1 does not split a label-subset in $\mathcal{P}$ into two trees, by Lemma 2.1.4, $\mathcal{P}$ is a $(c-(|X_2|-1))$-cut label-partition for $\mathcal{F}_2 - E_1$. Therefore, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_1)$ is at most $(c-(|X_2|-1))$. By Lemma 3.3.2, the optimal value c' for $(\mathcal{F}_1, \mathcal{F}_2 - E_{1,2})$ is also not larger than $(c-(|X_2|-1))$.

If all labels in $\ell(X_2)$ are in the same label-subset L_i in $\mathcal{P}$, then the degree of the parent of X_2 in the subtree $\mathcal{F}_2[L_i]$ is at most $|X_2|+1$ (because X_1 is a BSS). This implies that among the $|X_2|$ edges in E_2, at least $|X_2|-1$ must be removed in order to obtain the subtree $\mathcal{F}_2[L_i]$. Let the set of these $|X_2|-1$ edges be E_2', then $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - E_2')$. Note that removing any edge in E_2' would not split a label-subset in $\mathcal{P}$ into two trees. By Lemma 2.1.4, $\mathcal{P}$ is a $(c-(|X_2|-1))$-cut label-partition for $\mathcal{F}_2 - E_2'$. Therefore, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_2')$ is at most $(c-(|X_2|-1))$. By Lemma 3.3.2, the optimal value c' for $(\mathcal{F}_1, \mathcal{F}_2 - E_{1,2})$ is also not larger than $(c-(|X_2|-1))$.

This shows that Meta-Step 2.1 keeps ratio

$$|E_{1,2}|/(c-c') \leq (2|X_2|-1)/(c-(c-(|X_2|-1))) = (2|X_2|-1)/(|X_2|-1) \leq 3,$$

where we have used the fact $|X_2| \geq 2$. □



(A) Meta-Step 2 (B) Meta-Step 3 (C) Meta-Step 4

Figure 3.3: Situations for Meta-Steps 2-4

Subcase 2.2: $|E''| \leq |X_2|$.

Meta-Step 2.2. If $|E''| \leq |X_2|$, then pick any set E'_1 of $|E''| - 1$ edges incident to the leaves in X_2, and remove all edges in $E'_1 \cup E''$.

Lemma 3.3.5. *Meta-Step 2.2 keeps ratio* 3.

Proof. The proof is similar to that for Lemma 3.3.4. Let $E'_{1,2} = E'_1 \cup E''$. Meta-Step 2.2 totally removes $2|E''| - 1$ edges from $\mathcal{F}_2$. Let $\mathcal{P}$ be an optimal solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$. By Theorem 3.1.4, we can assume that either every leaf incident to an edge in E'_1 is in a unit label-subset in $\mathcal{P}$ (note $|E'_1| \leq |X_2| - 1$) or all labels in $\ell(X_2)$ are in the same label-subset in $\mathcal{P}$.

If every leaf incident to an edge in E'_1 is in a unit label-subset in $\mathcal{P}$, then $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - E'_1)$, and $\mathcal{P}$ is a $(c - (|E''| - 1))$-cut label-partition for $\mathcal{F}_2 - E'_1$. Thus, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E'_1)$ is at most $(c - (|E''| - 1))$, which implies that the optimal value c' for $(\mathcal{F}_1, \mathcal{F}_2 - E'_{1,2})$ is not larger than $(c - (|E''| - 1))$.

If all labels in $\ell(X_2)$ are in the same label-subset L_i in $\mathcal{P}$, then the degree of the parent of X_2 in the subtree $\mathcal{F}_2[L_i]$ is at most $|X_2| + 1$. Thus, in order to obtain the subtree $\mathcal{F}_2[L_i]$, at least $|E''| - 1$ of the edges in E'' must be removed. Suppose E''_2 is the set of $|E''| - 1$ edges in E'' that must be removed in order to obtain $\mathcal{F}_2[L_i]$. Then $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - E''_2)$, and is a $(c - (|E''| - 1))$-cut label-partition for $\mathcal{F}_2 - E''_2$. Therefore, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E''_2)$ is at most $(c - (|E''| - 1))$, which implies that the optimal value c' for $(\mathcal{F}_1, \mathcal{F}_2 - E'_{1,2})$ is also not larger than $(c - (|E''| - 1))$.

For the above analysis, we derive immediately that Meta-Step 2.2 keeps a ratio not larger than $(2|E''| - 1)/(|E''| - 1) \leq 3$, using the fact $|E''| \geq 2$ because the instance is strongly reduced. □

Case 3. X_2 contains a sibling set Y_2 with $|Y_2| \geq 2$.

Because of Case 2, here we assume that X_2 itself is not a sibling set.

Meta-Step 3 Let $u_2, v_2 \in Y_2$, and let w_2 be a leaf in X_2 that is not a sibling of u_2 and v_2. Let e be an edge incident to the parent of w_2 but not on the path between u_2 and w_2. Then remove the edge e, the edge e_u incident to u_2, and the edge e_w incident to w_2. See Figure 3.3(B).

Lemma 3.3.6. *Meta-Step 3 keeps ratio* 3.

Proof. Let $E_3 = \{e, e_u, e_w\}$. Let $\mathcal{P}$ be an optimal solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$. By Theorem 3.1.4, we can assume that either the label $\ell(u_2)$ is in a unit label-subset in $\mathcal{P}$ or both labels $\ell(u_2)$ and $\ell(v_2)$ are in the same label-subset in $\mathcal{P}$.

If $\ell(u_2)$ is in a unit label-subset in $\mathcal{P}$, then $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - e_u)$. By Lemma 2.1.4, $\mathcal{P}$ is a $(c-1)$-cut label-partition for $\mathcal{F}_2 - e_u$. This combined with Lemma 3.3.2 shows that the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_3)$ is at most $c - 1$.

Now suppose that $\ell(u_2)$ and $\ell(v_2)$ are in the same label-subset L_i in $\mathcal{P}$. If $\ell(w_2)$ is also in L_i then the edge e must be removed because $\ell(u_2)$, $\ell(v_2)$, and $\ell(w_2)$ are siblings in $\mathcal{F}_1$, which implies that the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e)$ (thus the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_3)$) is at most $c-1$. On the other hand, if $\ell(w_2)$ is not in L_i, then $\ell(w_2)$ must be in a unit label-subset in $\mathcal{P}$ because the only way to separate $\ell(w_2)$ from the tree containing both $\ell(u_2)$ and $\ell(v_2)$ in the forest $\mathcal{F}_1$ is to remove the edge incident to w_2. Therefore, in this case, $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - e_w)$, and is a $(c-1)$-cut label-partition for $\mathcal{F}_2 - e_w$, which implies again that the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_3)$ is at most $c - 1$.

Since Meta-Step 3 totally removes three edges, the above discussion implies directly that the meta-step keeps ratio 3. □

When none of the cases 1-3 hold true, all leaves in X_2 are in the same tree in $\mathcal{F}_2$ but no two are siblings. For this last case, we first prove the following lemma.

Lemma 3.3.7. *If all leaves in X_2 are in the same tree in $\mathcal{F}_2$ but no two are siblings, then there is a leaf in X_2 whose parent has degree 2 in the induced subforest $\mathcal{F}_2[\ell(X_2)]$.*

Proof. The lemma obviously holds true if $|X_2| = 2$ since the two leaves in X_2 are not siblings. For $|X_2| > 2$, the tree $\mathcal{F}_2[\ell(X_2)]$ has at least three unlabeled vertices. Removing all leaves in $\mathcal{F}_2[\ell(X_2)]$ results in a non-empty tree T'. Any leaf in T' is a degree-2 vertex in $\mathcal{F}_2[\ell(X_2)]$ that is a parent of a leaf in X_2. $\square$

Case 4. All leaves in X_2 are in the same tree in $\mathcal{F}_2$ but no two are siblings.

Meta-Step 4 Let $u_2, v_2 \in X_2$ such that the parent p_2 of u_2 has degree 2 in $\mathcal{F}_2[\ell(X_2)]$. Let e be an edge in $\mathcal{F}_2$ that is incident to p_2 but not on the path P_{uv} between u_2 and v_2. Then, cut the edge e, the edge e_u incident to u_2, and the edge e_v incident to v_2. See Figure 3.3(C).

Lemma 3.3.8. *Meta-Step 4 keeps ratio* 3.

Proof. Let $E_4 = \{e, e_u, e_v\}$. Let $\mathcal{P}$ be an optimal solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$. If removing any e' of the edges in E_4 does not split any label-subset in $\mathcal{P}$ into two trees, then $\mathcal{P}$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - e')$. By Lemma 2.1.4, $\mathcal{P}$ is a $(c-1)$-cut label-partition for $\mathcal{F}_2 - e'$. This combined with Lemma 3.3.2 shows that the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_4)$ is at most $c - 1$.

So we assume removing any edge in E_4 will split some label-subsets in $\mathcal{P}$. In particular, $\ell(u_2)$ and $\ell(v_2)$ are not in unit label-subsets in $\mathcal{P}$. Let $\mathcal{F}^*$ be the maximum agreement forest induced by $\mathcal{P}$. Since $\ell(u_2)$ and $\ell(v_2)$ are siblings in $\mathcal{F}_1$, $\ell(u_2)$ and $\ell(v_2)$ must be siblings in $\mathcal{F}^*$. Since the edge e must remain in $\mathcal{F}^*$, the vertex p_2, which is the parent of u_2 in $\mathcal{F}_2$, must be the parent of $\ell(u_2)$ and $\ell(v_2)$ in $\mathcal{F}^*$. By Lemma 3.3.7, p_2 has degree 2 in $\mathcal{F}_2[\ell(X_2)]$. Since p_2 corresponds to the parent of $\ell(u_2)$ and $\ell(v_2)$ in $\mathcal{F}_1$, which is incident to at most one edge whose other end is not in X_1, p_2 must have degree exactly 3 in $\mathcal{F}^*$ (the edge e does not lead to another vertex in X_2).

Thus, if we let E_{uv} be the set of all edges, except e, that are incident to a vertex in P_{uv} but not on P_{uv}, then in order to make $\mathcal{F}^*$ from $\mathcal{F}_2$, all edges in E_{uv} must be removed. Let E_c be a set of c edges in $\mathcal{F}_2$ such that the label-partition for $\mathcal{F}_2 - E_c$ is $\mathcal{P}$. Since removing any edge in E_{uv} does not split a label-subset in $\mathcal{P}$, we can assume $E_{uv} \subseteq E_c$. Now removing the edge e in $\mathcal{F}_2 - E_c$ makes the path P_{uv} become a component of the forest, and removing the edge e_u further makes $\ell(u_2)$ and $\ell(v_2)$ become single-vertex trees. Let the resulting forest be $\mathcal{F}'$ whose label-partition is $\mathcal{P}'$. Then both $\ell(u_2)$ and $\ell(v_2)$ are in unit label-subsets in $\mathcal{P}'$, and $\mathcal{P}'$ is a solution for $(\mathcal{F}_1, \mathcal{F}_2 - E_4)$, as well as a solution for $(\mathcal{F}_1, \mathcal{F}_2)$. Moreover, $\mathcal{P}'$ is a $(c+2)$-cut label-partition for $\mathcal{F}_2$.

On the other hand, if we cut the edges e, e_u, and e_v in $\mathcal{F}_2$, each cutting certainly does not split a label-subset in $\mathcal{P}'$. Moreover, since $\ell(u_1)$ and $\ell(v_2)$ are not siblings in $\mathcal{F}_2$, cutting each of these edges increases the number of leaf-labeled trees in the forest. Therefore, $\mathcal{P}'$ is a $((c+2)-3)$-cut, i.e., a $(c-1)$-cut label-partition for $\mathcal{F}_2 - E_4$. This shows that the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - E_4)$ is at most $c-1$.

Since $|E_4| = 3$, we conclude that Meta-Step 4 keeps ratio 3. □

Now we are ready to present our approximation algorithm for the MAX-MAF' problem in Figure 3.4.

```
Apx-MAF
INPUT: two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set $L$
OUTPUT: an $L$-partition $\mathcal{P}$ that is a solution for $(\mathcal{F}_1, \mathcal{F}_2)$
1.   apply Reduction Rules 1-2 and contractions on $(\mathcal{F}_1, \mathcal{F}_2)$ until they are not applicable;
2.   **repeat** until $\mathcal{F}_2$ becomes a subgraph of $\mathcal{F}_1$
2.1    Let $X_1$ be a BSS in $\mathcal{F}_1$, and let $X_2$ be the leaf set in $\mathcal{F}_2$ with $\ell(X_1) = \ell(X_2)$;
2.2    **switch**
         case 1:     apply Meta-Step 1;
         case 2.1:   apply Meta-Step 2.1;
         case 2.2:   apply Meta-Step 2.2;
         case 3:     apply Meta-Step 3;
         case 4:     apply Meta-Step 4;
2.3    apply Reduction Rules 1-2 and contractions on $(\mathcal{F}_1, \mathcal{F}_2)$ until they are not applicable;
3.   return the $L$-partition $\mathcal{P}$ constructed from the label-partition for $\mathcal{F}_2$.
```

Figure 3.4: An approximation algorithm for MAX-MAF'

Theorem 3.3.9. *Algorithm* **Apx-MAF** *is an approximation algorithm for the* MAX-MAF' *problem that runs in time $O(n^2)$ and has an approximation ratio at most* 3.

Proof. We provide some explanations for the algorithm. Because of step 1 and step 2.3, the pair $(\mathcal{F}_1, \mathcal{F}_2)$ on step 2.1 is strongly reduced on which contraction is not applicable. In particular, $\mathcal{F}_1$ cannot consist of only single-vertex trees: otherwise by Reduction Rule 1, $\mathcal{F}_2$ should have also consisted of single-vertex trees and $\mathcal{F}_2$ would have been a subgraph of $\mathcal{F}_1$, contradicting the condition given in step 2. Therefore, at step 2.1, the forest $\mathcal{F}_1$ must has a BSS X_1.

We also need to give some explanations on step 3. During the process of the algorithm, each shrinking operation replaces a label subset $\ell(X)$ with a "combined" new label ℓ_X. Therefore, in order to get the L-partition $\mathcal{P}$ for the input instance, we need to restore each label subset $\ell(X)$ from the corresponding combined label ℓ_X in a straightforward way, in reverse order.

The algorithm consists of a sequence of meta-steps, in which each meta-step is either one of those given in step 2.2 of the algorithm, or a contraction operation, or an application of Reduction Rules 1-2. Suppose that on an instance $I = (\mathcal{F}_1, \mathcal{F}_2)$, a meta-step M produces an instance $I' = (\mathcal{F}'_1, \mathcal{F}'_2)$. Let c and c' be the optimal values for the instances I and I', respectively. Moreover, let $\mathcal{P}_2$ be the label-partition for the forest $\mathcal{F}_2$ and have d_2 label-subsets, and let $\mathcal{P}'_2$ be the label-partition for the forest $\mathcal{F}'_2$ and have d'_2 label-subsets.

If the meta-step M is one of those in step 2.2 or is an application of Reduction Rule 1, then $\mathcal{F}'_1$ and $\mathcal{F}'_2$ are obtained from $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively, by removing certain edges. Thus, every solution for the instance I' is also a solution for the instance I. Suppose that the meta-step M removes a set E_2 of edges in $\mathcal{F}_2$, i.e., $\mathcal{F}'_2 = \mathcal{F}_2 - E_2$. Since removing an edge in a leaf-labeled forest can increase the number of label-subsets in its corresponding label-partition by at most one, the number d'_2 of label-subsets in $\mathcal{P}'_2$ is at most $|E_2|$ more than that in $\mathcal{P}_2$, i.e., $d'_2 \leq d_2 + |E_2|$. By Lemmas 3.3.3-3.3.6 and 3.3.8, also noticing the remark given before Lemma 3.3.2, we have $|E_2| \leq 3(c - c')$. Therefore, $d'_2 \leq d_2 + 3(c - c')$.

If the meta-step M is a contraction or an application of Reduction Rule 2, then $d_2 = d'_2$, and by Lemma 2.1.1 and Lemma 3.1.6, we also have $c = c'$. Therefore, the condition $d_2' \leq d_2 + 3(c - c')$ also holds true. Note that because of the shrinking operation, a solution for I' may no longer be a solution for I. However, as explained above, from a solution with c label-subsets for the instance I', we can easily construct a solution with c label-subsets for the instance I.

Now suppose that the sequence of the meta-steps of the algorithm is $\{M_1, M_2, \dots, M_h\}$, where for each i, $1 \leq i \leq h$, the meta-step M_i on an instance $I_i = (\mathcal{F}_{1,i}, \mathcal{F}_{2,i})$ produces an instance $I_{i+1} = (\mathcal{F}_{1,i+1}, \mathcal{F}_{2,i+1})$, and let c_i be the optimal value for the instances I_i, and let $\mathcal{P}_i$ be the label-partition for the forest $\mathcal{F}_{2,i}$ that consists of d_i label-subsets. By the above, analysis, we have $d_{i+1} \leq d_i + 3(c_i - c_{i+1})$ for all i. From this, we get immediately $d_{h+1} \leq d_1 + 3(c_1 - c_{h+1})$.

The instance $I_1 = (\mathcal{F}_{1,1}, \mathcal{F}_{2,1})$ is the input $(\mathcal{F}_1, \mathcal{F}_2)$ to the algorithm **Apx-MAF**, whose optimal value is c_1 and the label-partition $\mathcal{P}_1$ for $\mathcal{F}_{2,1} = \mathcal{F}_2$ has d_1 label-subsets. By the condition in step 2, in the final instance $I_{h+1} = (\mathcal{F}_{1,h+1}, \mathcal{F}_{2,h+1})$, $\mathcal{F}_{2,h+1}$ is a subgraph of $\mathcal{F}_{1,h+1}$. Therefore, the optimal value c_{h+1} for I_{h+1} is 0. This gives $d_{h+1} \leq d_1 + 3c_1$. Moreover, by step 3 and the above discussion, the solution $\mathcal{P}$ returned by the algorithm is constructed from the solution for the instance I_{h+1} (i.e., the label-partition for $\mathcal{F}_{2,h+1}$), which consists of d_{h+1} label-subsets. Therefore, the solution $\mathcal{P}$ also consists of d_{h+1} label-subsets. Since the label-partition $\mathcal{P}_1$ for $\mathcal{F}_2$ consists of d_1 label-subsets, by Lemma 2.1.2, the solution $\mathcal{P}$ is a $(d_{h+1} - d_1)$-cut label-partition for $\mathcal{F}_2$, i.e., $\mathcal{P}$ is a solution of value $d_{h+1} - d_1 \leq 3c_1$ for the input $(\mathcal{F}_1, \mathcal{F}_2)$. This shows that the approximation ratio of the algorithm **Apx-MAF** is not larger than $(d_{h+1} - d_1)/c_1 \leq 3$.

Finally, it is easy to see that the running time of the algorithm **Apx-MAF** is $O(n^2)$ because each meta-step runs in time $O(n)$ and decreases the number of edges in the instance by at least one. $\square$

The original MAX-MAF problem on two given leaf-labeled trees T_1 and T_2 over the label-set L asks to construct a maximum agreement forest for T_1 and T_2. Suppose that a maximum agreement forest for T_1 and T_2 consists of c leaf-labeled trees. Then the L-

partition that is an optimal solution for (T_1, T_2) consists of c label-subsets, i.e., the optimal value for (T_1, T_2) is $c-1$. We can apply the **Apx-MAF** algorithm on the instance (T_1, T_2), which will return an L-partition $\mathcal{P}$ that is a solution of value at most $3(c-1)$ for (T_1, T_2). Therefore, the solution $\mathcal{P}$ induces an agreement forest of at most $3c-2$ trees for (T_1, T_2). The ratio $(3c-2)/c < 3$ shows that **Apx-MAF** can also be used as an approximation algorithm for the problem MAX-MAF that has an approximation ratio bounded by 3.

When applied on binary trees, the algorithm **Apx-MAF** has its ratio of 3 matches the best previous known ratio for the problem on binary trees [71]. The only previously known approximation algorithm for the MAF problem on general trees is on rooted trees and has ratio $d+1$, where d is the maximum number of children a node in the forests may have [62]. Our algorithm is the first constant-ratio approximation algorithm for the MAF problem on general trees, where the trees are unrooted.

3.4 Conclusion

We presented a parameterized algorithm and an approximation algorithm for the MAF problem on unrooted general trees, which corresponds to the TBR distance on multifurcating phylogenetic trees. For general trees, our parameterized algorithm is the first fixed-parameter tractable algorithm and our approximation algorithm is the first constant-ratio approximation algorithm. Our algorithms are based on the concept of a bottommost sibling set in one tree and the structure of the corresponding leaf set in the other tree. The methods based on sibling sets have been used by other researchers [70] but the special structure of the bottommost sibling set enables us to deal with operations on general trees more effectively.

4. ALGORITHMS TO MULTICUT ON TREES PROBLEM [*]

In this chapter, we present our kernelization algorithm and parameterized algorithm to multicut on trees problem.

4.1 A kernelization algorithm to multicut on trees problem

In this section we prove an upper bound of $O(k^3)$ on the kernel size for multicut on trees problem. The approach can be summarized as follows. We group the vertices in the forest into $O(k)$ groups such that no request exists within the vertices of the same group. We then define an order among the bad leaves of a group with respect to the bad leaves and the internal vertices of another group. This ordering allows us to introduce a set of reduction rules that replace the requests from the bad leaves of a group (except few), going to the internal vertices and bad leaves of another group, allowing us to derive an $O(k^2)$ upper bound on the total number of bad leaves that have requests to other bad leaves or internal vertices. We then use the crown kernelization algorithm [1] for vertex cover problem to upper bound the number of bad leaves in a group that have requests to good leaves in another group, again obtaining an $O(k^2)$ upper bound on the total number of bad leaves that have requests to good leaves. Combining the above allows us to upper bound the number of leaves in the reduced instance by $O(k^2)$, improving on the $O(k^4)$ upper bound obtained in [13]. 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 reduced instance. We now proceed to the details. Let $(\mathcal{F}, R, k)$ be an instance of multicut on trees problem, and let T be a 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

[*]Reprinted with permission from "Multicut in trees viewed through the eyes of vertex cover" by Jianer Chen, Jia-Hao Fan, Iyad A. Kanj, Yang Liu and Fenghui Zhang, 2012. J. Comput. Syst. Sci., 78(5): 1637-1650 (2012), Copyright [2012] by Elsevier.

following reduction rules for multicut on trees problem are folklore, easy to verify, and can be implemented to run in polynomial time (see [13, 34] for proofs). Therefore, we omit their proofs.

4.1.1 Basic reduction rules

Reduction Rule 4.1.1 (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 4.1.2 (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 4.1.3 (Unit request). *If $(u, v) \in R$ and $uv \in E(\mathcal{F})$, then cut uv (i.e., remove uv from $\mathcal{F}$ and decrement k by 1).*

Reduction Rule 4.1.4 (Disjoint requests). *If there are $k+1$ pairwise disjoint requests in R, then reject the instance $(\mathcal{F}, R, k)$.*

Reduction Rule 4.1.5 (Unique direction). *Let x be a leaf or an internal degree-2 vertex in $\mathcal{F}$. Suppose that all the requests from x have the same direction, i.e., can be disconnected by the removal of a single edge from $\mathcal{F}$ that is not incident to x in case x is a leaf, and can be disconnected by the removal of a single edge in case x is an internal degree-2 vertex. 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.*

Reduction Rule 4.1.6 (Domination/Inclusion). *If a request (p, q) dominates another request (u, v) then remove (u, v) from R.*

It was shown in [13] 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 [15] (this reduction rule was implicitly observed in [34]). (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 problem in kernelization algorithms for multicut on trees problem, 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 4.1.7 (**Bound on good leaves**). *Apply Buss' kernelization algorithm for vertex cover problem [15] 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 good leaves in $\mathcal{F}$ after Buss' algorithm is applied is more than $2k^2$, then reject the input instance $(\mathcal{F}, R, k)$. (Note that a good leaf may become bad after cutting some leaves in $\mathcal{F}$.)*

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 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 on trees problem. □

We shall assume henceforth that none of Reduction Rules 4.1.1 – 4.1.7 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 [13]:

Lemma 4.1.1 ([13]). *In the forest $\mathcal{F}$, both the number of internal vertices of internal degree 1 and the number of internal vertices of internal degree at least 3 are at most k.*

(The number of internal vertices of internal degree 1 is at most k because at least two good leaves must be attached to each such vertex by the unique direction reduction rule, and by the disjoint requests rule, there can be at most k such vertices. The number of internal vertices of internal degree at least 3 is at most k because the number of such vertices is not more than the number of internal vertices of internal degree 1.)

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

4.1.2 Three types of groups in forest

Type-I group. 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. Note that by the unique direction rule, every vertex in $\mathcal{F}$ of internal degree 1 forms a type-I group.

Type-II 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.

Type-III group. After removing all the vertices in the type-I and type-II groups, each connected component of the resulting forest is a caterpillar in which each internal vertex u has internal degree 2 in $\mathcal{F}$, and all the leaves attached to u (if any) are bad leaves. Now we further partition each caterpillar in a greedy fashion into vertex-disjoint subcaterpillars such that the following condition is satisfied: There is no request between any two vertices (internal-internal, leaf-internal, nor leaf-leaf) of the same subcaterpillar. To partition a caterpillar, we start from an internal vertex that is an endpoint of the caterpillar, and traverse the caterpillar towards the other endpoint as long as the above condition is not violated. The first time a vertex v is reached such that the condition is violated at v, or at one of the leaves attached to v, the subcaterpillar traversed so far up to the vertex before v forms a *type-III group*, and the process is repeated starting from v; the process stops when the other endpoint of the caterpillar is reached. Note that there is no request between any two vertices of a type-III group, and that, for any two type-III groups in the same caterpillar 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 is clear that the above partitioning can be carried out in polynomial time.

Lemma 4.1.2. *The number of groups obtained from the above partitioning of $V(\mathcal{F})$ is $O(k)$.*

Proof. 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. The fact that the number of type-II groups is $O(k)$ follows from Lemma 4.1.1.

Now we consider the type-III groups. For each tree T in the forest $\mathcal{F}$, if we remove the vertices in the type-I and type-II groups from T, we obtain a set of caterpillars. It is fairly easy to see that the number of such caterpillars in T is bounded by the number of type-I and type-II groups in T: if we remove from T all the leaves and replace each caterpillar by an edge, we obtain a new tree in which each vertex corresponds to a group of type-I or type-II, and each edge corresponds to a caterpillar. This, combined with the fact that there are totally $O(k)$ type-I and type-II groups in $\mathcal{F}$, implies that there are $O(k)$ caterpillars after the vertices in the type-I and type-II groups are removed from $\mathcal{F}$. For two type-III groups that are consecutive subcaterpillars in the same caterpillar, there exists a request between some vertex of the first group and another vertex in the other group. Therefore, if a caterpillar C is partitioned into 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. This, in addition to the fact that the number of caterpillars is $O(k)$, implies that the total number of type-III groups is $O(k)$. $\square$

Definition 4.1.3. Let γ_i be a type-I, type-II, or a type-III group. The *intergroup edges* of γ_i are the edges in $\mathcal{F}$ with exactly one endpoint in γ_i; the *intergroup degree* of γ_i, denoted d_i, is the number of intergroup edges of γ_i. Note that if γ_i is a type-I or a type-II group, where u is the internal vertex in $\mathcal{F}$ that forms γ_i, then d_i is the internal degree of u in $\mathcal{F}$. On the other hand, if γ_i is a type-III group then $d_i = 2$ (each of the two endpoints of a caterpillar forming a type-III group has internal degree 2, and has exactly one neighbor that is not in the caterpillar). The *internal vertices of* γ_i are the internal vertices of $\mathcal{F}$ that are in γ_i. The *internal edges of* γ_i are the edges between the internal vertices of γ_i. Note that only type-III groups can have internal edges. The *leaves (resp. good/bad leaves) of* γ_i are the leaves (resp. good/bad leaves) attached to the internal vertices of γ_i.

Lemma 4.1.4.

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

Proof. Since there are $O(k)$ type-III groups by Lemma 4.1.2, 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 Lemma 4.1.1. □

4.1.3 Reduction rules to bad leaves, good leaves and internal nodes

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 problem. 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 4.1.8 (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, the subgraph of G induced by the good leaves that are attached to 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 4.1.5. *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 4.1.6. *Let u be a vertex in $\mathcal{F}$ that forms a type-I group γ_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 by Lemma 4.1.5, S cuts at least $\tau(G_u) + d_u$ edges that are incident to the leaves in γ_i. We can then remove all the edges in S that are incident to the leaves in γ_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 γ_i. □

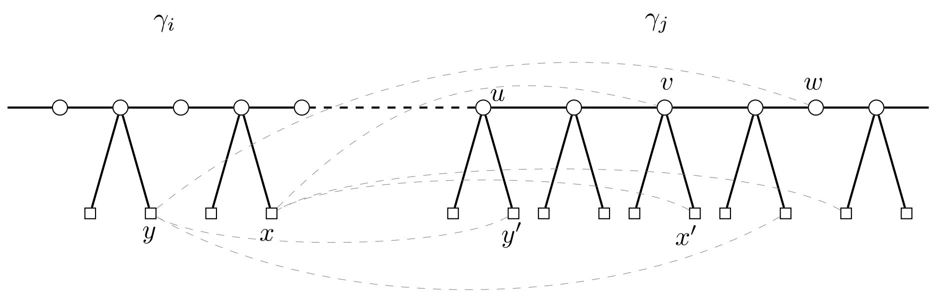


Figure 4.1: Illustration for Definition 4.1.8 and Definition 4.1.9: v-offset$_j(y)$ is w; v-offset$_j(x)$ is v; l-offset$_j(y)$ is y'; l-offset$_j(x)$ is x'. Therefore, we have $x \preceq_j^v y$ and $y \preceq_j^l x$.

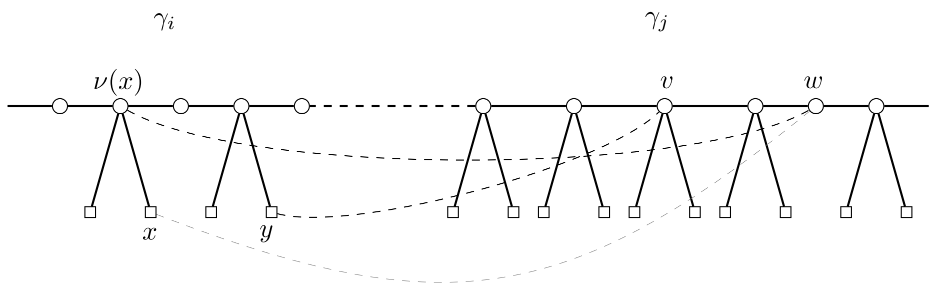


Figure 4.2: Illustration for Reduction Rule 4.1.10 and the replacement of requests: v-offset$_j(x)$ is w; v-offset$_j(y)$ is v. Replace the request (x, v-offset$_j(x)$) with ($\nu(x)$, v-offset$_j(x)$).

Lemma 4.1.7. *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 γ_i: if γ_i is a type-I or a type-II group then S cuts at most $d_i - 1$ bad leaves of γ_i, and if γ_i is a type-III group then the number of bad leaves and internal edges of γ_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 γ_i be a group. If γ_i is a type-I or a type-II group such that S cuts at least d_i bad leaves from γ_i, then the edges in S that are incident to all the bad leaves of γ_i can be replaced with the intergroup edges of γ_i. Similarly, if γ_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 γ_i, is at least $d_i = 2$, then those edges in S can be replaced with the two intergroup edges of γ_i. By performing the above replacement for every group γ_i, we obtain a solution S that satisfies the statement of the lemma. $\square$

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 of bad leaves of a group γ_i with respect to another group γ_j, to limit the number of bad leaves of γ_i that have requests to bad leaves or vertices of γ_j to at most $d_i \times d_j$. For a leaf x of a group γ_i, we shall refer to the internal vertex in γ_i that x is attached to by $\nu(x)$.

Reduction Rule 4.1.9. (**Bound on the number of bad leaves in a group that have requests to a certain vertex**)

Let x be a vertex, and let γ_i be a group. If there are at least d_i bad leaves in γ_i that have requests to x, then let L_x be the list containing the bad leaves in γ_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 γ_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 γ_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 4.1.7, we can assume that there is a solution S that cuts at most $d_i - 1$ bad leaves from γ_i. Let z be a bad leaf in L_x whose rank is larger than $d_i - 1$. If S does not cut z, then S must cut an edge on the path from $\nu(z)$ to x. On the other hand, if S cuts z, then because S cuts at most $d_i - 1$ bad leaves in γ_i, there is a bad leaf z' in L_x whose rank is smaller than that of z such that S does not cut z' but cuts an edge e' on the path from $\nu(z')$ to x. By the way we rank the bad leaves in L_x, cutting the edge e' will also cut the path from $\nu(z)$ to x. Therefore, in any case, the solution S will also cut the request $(\nu(z), x)$ in $(\mathcal{F}, R', k)$ that replaces the request (z, x) in $(\mathcal{F}, R, k)$. As a consequence, S is also a solution to the instance $(\mathcal{F}, R', k)$. □

Definition 4.1.8. Let γ_i and γ_j be two distinct groups, and let x be a bad leaf in γ_i.

If x has a request to an internal vertex w in γ_j (by the domination reduction rule, such internal vertex is unique), then call w the *vertex-offset of x with respect to γ_j*, denoted v-offset$_j(x)$.

If x has a request to bad leaves in γ_j, then define the *leaf-offset of x with respect to γ_j*, denoted l-offset$_j(x)$, to be *any* such leaf with the minimum distance to x.

Definition 4.1.9. (**orders** $\preceq_j^v$ and $\preceq_j^l$ in γ_i) Let γ_i and γ_j be two distinct groups, and let u be the vertex in γ_j that has the minimum distance to the vertices in γ_i. For any two bad leaves x and y in γ_i we say:

$x \preceq_j^v y$ if the distance from v-offset$_j(x)$ to u is not larger than that from v-offset$_j(y)$ to u; and

$x \preceq_j^l y$ if the distance from l-offset$_j(x)$ to u is not larger than that from l-offset$_j(y)$ to u.[1]

See Figure 4.1 for an illustration of vertex-offsets, leaf-offsets, and the orders $\preceq_j^l, \preceq_j^v$.

Reduction Rule 4.1.10. (**Bound on the number of bad leaves in a group that have requests to internal vertices in another group**)

Let γ_i and γ_j be two distinct groups. If there are at least d_i bad leaves in γ_i that have

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

requests to internal vertices of γ_j, then consider all bad leaves in γ_i that have requests to internal vertices of γ_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 γ_j with the request $(\nu(x), p)$. Refer to Figure 4.2 for illustration.

Proof. Suppose that the above reduction rule applies to two groups γ_i and γ_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 4.1.7, we can assume that there is a solution S such that if γ_i is a type-I or a type-II group then S cuts at most $d_i - 1$ bad leaves of γ_i, and if γ_i is a type-III group then the number of bad leaves and internal edges of γ_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 p of x with respect to γ_j, $p = \text{v-offset}_j(x)$, and hence, the request from x to p that was replaced with $(\nu(x), p)$ is cut by S. Suppose now that S cuts x, and note that if γ_i is a type-III group then because S cuts x, S does not cut *any* bad leaf of γ_i other than x, nor does it cut an internal edge of γ_i. Therefore, if γ_i is a type-III group, then we may contract all the internal edges of γ_i to obtain a single internal vertex, say w, such that the leaves attached to w are precisely the leaves of γ_i. If γ_i is a type-I or a type-II group, also denote by w the internal vertex that forms group γ_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 p. 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 with respect to γ_j appear no later than p. Since S cuts x, it follows that S cuts d_i bad leaves of γ_i, contradicting our assumption. It follows that S must cut an edge on the path between w and p. Since S does not cut any internal edge from γ_i in case γ_i is a type-III group, it follows that the request between $(\nu(x), p)$ in $(\mathcal{F}, R', k)$ that replaces the request (x, p) in $(\mathcal{F}, R', k)$ is also cut by S. Since x was arbitrarily chosen to be a leaf whose rank in L_i is at least d_i, this shows that S is also a solution to $(\mathcal{F}, R', k)$. $\square$

Reduction Rule 4.1.11. (Bound on the number of bad leaves in a group that have requests to bad leaves in another group)
Suppose that Reduction Rule 4.1.9 does not apply to $(\mathcal{F}, R, k)$. Let γ_i and γ_j be two distinct groups. If there are at least $(d_i - 1) \times d_j + 1$ bad leaves in γ_i that have requests to bad leaves in γ_j, then consider all the bad leaves in γ_i that have requests to bad leaves of γ_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 γ_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 γ_j with the request $(\nu(x), y)$.

Proof. Suppose that the above reduction rule applies to two groups γ_i and γ_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 S to $(\mathcal{F}, R, k)$. By Lemma 4.1.7, we can assume that S cuts at most $d_i - 1$ bad leaves from γ_i and at most $d_j - 1$ bad leaves from γ_j, and that if γ_i is a type-III group then the number of bad leaves and internal edges of γ_i that are cut by S is at most 1.

Therefore, by cutting bad leaves in γ_i, S can cut requests from at most $d_i - 1$ bad leaves in γ_i to bad leaves in γ_j. Since Reduction Rule 4.1.9 is not applicable, each bad leaf z in γ_j has at most $d_i - 1$ requests to bad leaves in γ_i. Thus, cutting z can cut requests from at most $d_i - 1$ bad leaves in γ_i to z. Since S cuts at most $d_j - 1$ bad leaves in γ_j, by cutting bad leaves in γ_j, S can cut requests from at most $(d_i - 1) \times (d_j - 1)$ bad leaves in γ_i to bad leaves in γ_j. Putting these together, we conclude that by cutting bad leaves in γ_i and γ_j, S can cut requests from at most $(d_i - 1) + (d_i - 1) \times (d_j - 1) = (d_i - 1) \times d_j$ bad leaves in γ_i to bad leaves in γ_j.

Now let x be a bad leaf in γ_i whose rank in L_i is at least $(d_i - 1) \times d_j + 1$. If S does not cut x, then for any request (x, y) in $(\mathcal{F}, R, k)$ where y is a bad leaf in γ_j, S must cut an edge on the path between $\nu(x)$ and y. Hence, the request $(\nu(x), y)$ in $(\mathcal{F}, R', k)$ that replaces the request (x, y) in $(\mathcal{F}, R, k)$ is also cut by S. If S cuts x, then by the analysis in the previous paragraph, there must be a bad leaf x' in L_i whose rank is smaller than x, and a request (x', y'), where $y' = \text{l-offset}_j(x')$, such that S neither cuts x' nor y'. Since $x' \preceq_j^l x$, the edge in S that cuts the request (x', y') must also cut the request from $\nu(x)$ to bad leaves in γ_j (note that in case γ_i is a type-III group, since S cuts x, S would not cut any internal edges in γ_i). Thus, again S cuts any request $(\nu(x), y)$ in $(\mathcal{F}, R', k)$ that replaces the request (x, y) in $(\mathcal{F}, R, k)$, i.e., S is also a solution to $(\mathcal{F}, R', k)$. □

Reduction Rule 4.1.12. (Bound on the number of bad leaves in a group that have requests to good leaves in OUT_u for a type-I group γ_j formed by vertex u) *Suppose that Reduction Rule 4.1.9 does not apply to $(\mathcal{F}, R, k)$. Let u be a vertex such that u forms a type-I group γ_j, and let $\gamma_i \neq \gamma_j$ be a group. If there are at least $d_j \times (d_i - 1) + 1$ many bad leaves in γ_i that have requests to leaves in OUT_u, let L_i be the list of bad leaves in γ_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 γ_i and a type-I group γ_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 S to $(\mathcal{F}, R, k)$. By Corollary 4.1.6, we can assume that S cuts at most $d_u - 1 = d_j - 1$ leaves from OUT_u. By Lemma 4.1.7, we can also assume that S cuts at most $d_i - 1$ bad leaves from γ_i. Since Reduction Rule 4.1.9 is not applicable, every leaf in OUT_u has requests to at most $d_i - 1$ bad leaves in γ_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 γ_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 γ_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 4.1.10. The instance $(\mathcal{F}, R, k)$ is said to be *reduced* if none of Reduction Rules 4.1.1 – 4.1.12 is applicable to it.

Lemma 4.1.11. *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 γ_i be a group with bad leaves. By Reduction Rule 4.1.9, every good leaf in G has requests to at most $d_i - 1$ bad leaves in γ_i. Therefore, the number of bad leaves in γ_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 4.1.8). Thus, the number of bad leaves in $\mathcal{F}$ that have requests to good leaves in $O \cup V(M)$ is at most $5k \times \sum_{\gamma_i} d_i = O(k^2)$ (by Lemma 4.1.4). 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 γ_j formed by some vertex u, is $O(k^2)$.

In effect, let γ_j be a type-I group formed by a vertex u. For every group $\gamma_i \neq \gamma_j$, by Reduction Rule 4.1.12, the number of bad leaves in γ_i that have requests to good leaves in OUT_u is at most $(d_i - 1) \times d_j < d_i \times d_j$. Therefore, the total number of bad leaves in $\mathcal{F}$ that have requests to good leaves in OUT_u is at most $d_j \times \sum_{\gamma_i} d_i = d_j \times O(k)$ (by Lemma 4.1.4). By summing over all type-I groups γ_j, the number of bad leaves that have requests to good leaves is $O(k) \times \sum_{\gamma_j \text{ is of type-I}} d_j = O(k) \times O(k) = O(k^2)$. □

Lemma 4.1.12. *Let $(\mathcal{F}, R, k)$ be a reduced instance. The number of leaves in $\mathcal{F}$ is $O(k^2)$.*

Proof. By Reduction Rule 4.1.7, 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 γ_i, and must have a request to a good leaf, an internal vertex, or a bad leaf of another group γ_j. By Lemma 4.1.11, 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 γ_i that have requests to an internal vertex or to a bad leaf in another group γ_j.

Fix a group $\gamma_j \neq \gamma_i$. By Reduction Rule 4.1.10, the number of bad leaves in γ_i that have requests to internal vertices of γ_j is less than d_i. Therefore, the total number of bad leaves in γ_i that have requests to internal vertices in $\mathcal{F}$ is $O(k) \times d_i$ since by Lemma 4.1.2 the number of groups is $O(k)$. Summing over all groups γ_i, we obtain that the total number of bad leaves in $\mathcal{F}$ that have requests to internal vertices in $\mathcal{F}$ is $O(k) \times \sum_{\gamma_i} d_i = O(k) \times O(k) = O(k^2)$. By Reduction Rule 4.1.11, the number of bad leaves in γ_i that have requests to bad leaves in γ_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 γ_i is at most $d_i \times \sum_{\gamma_j \neq \gamma_i} d_j = d_i \times O(k)$. Summing over all groups γ_i, we obtain that the number of bad leaves in $\mathcal{F}$ that have requests to bad leaves in $\mathcal{F}$ is $O(k) \times \sum_{\gamma_i} d_i = O(k) \times O(k) = O(k^2)$. □

Lemma 4.1.13. *Let $(\mathcal{F}, R, k)$ be a reduced instance. The number of vertices in $\mathcal{F}$ whose internal degree is not equal to 2 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 internal degree at least 3 (note that every vertex in $\mathcal{F}$ with internal degree 1 must have leaves attached to it). By Lemma 4.1.12, 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 Lemma 4.1.1, 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 4.1.14. *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 4.1.13, 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 ancestor/descendant relationship in the 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 D, add a directed edge from u to v in D if: (1) the current outdegree of u in D is 0, v is a descendant of u in the tree T in $\mathcal{F}$, and there is a request between u and v. 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 descendant 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.[2] 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. 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 descendant of h_P that h_P has a request to could be in D; otherwise, because the outdegree of h_P in D is 0, a directed edge from h_P to such a descendant would exist in D. Therefore, h_P must have a request to some descendant 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

[2]We note that a similar graph to D was defined in [13]. However, the upper bound on the number of vertices in that graph derived in [13] 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 $O(k)$.

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 4.1.15. *Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem. Then the number of vertices in $\mathcal{F}$ is $O(k^3)$.*

Proof. The theorem follows directly from Lemma 4.1.13 and Lemma 4.1.14. □

Corollary 4.1.16. *The multicut on trees problem has a kernel of at most $O(k^3)$ vertices.*

Proof. This follows from Theorem 4.1.15 and the fact that Reduction Rules 4.1.1 – 4.1.12 can be implemented to run in polynomial time. □

4.2 A parameterized algorithm to multicut on trees problem

In this section, we present our parameterized algorithm to multicut on trees problem.

Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem. Since $(\mathcal{F}, R, k)$ is reduced, we can assume that every tree in $\mathcal{F}$ is nontrivial (contains at least three vertices). 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 with respect to V' if $dist_T(u,r) = \max\{dist_T(w,r) \mid w \in V'\}$.

Before we proceed further, we try to give the reader an intuitive idea about how the algorithm works. The algorithm uses a branch-and-search strategy. Before the branching is performed, more reduction rules, which further exploit the connection between multicut on trees problem and vertex cover problem, are applied. The algorithm then applies a general branching rule (**BranchRule 4.2.5**) to simplify the structure further. After this branching rule is applied, it can be assumed that, for any important vertex w in a tree of $\mathcal{F}$, the degree of every leaf in G_w is at most 2. (Recall that G_w is the subgraph of G induced by the vertices in G that correspond to the good leaves attached to w.) This aforementioned property allows for efficient branching. By choosing an important vertex w properly, it is shown that there must be a request from w, or from a child of w, to a vertex in the subtree of $\mathcal{F}$ rooted at the parent of w. Based on this request, the algorithm distinguishes four possible branching cases (the different cases are illustrated in Figure 4.3), and branches accordingly.

The following lemma, which again emphasizes the importance of vertex cover problem for both kernelization and parameterized algorithms for multicut on trees problem, will be pivotal:

Lemma 4.2.1. *Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem. 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 corresponds to a minimum vertex cover of G_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 whose size is not larger than that of 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 4.2.1. *Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem, 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 4.2.2. *Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem, 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 4.2.1, 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 4.2.3. *Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem, 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 polynomial time since $\Delta(G_w) \leq 2$, if G_w contains a path P of even length, then by Lemma 4.2.1, we can cut the vertices in *children*(w) that correspond to the minimum vertex cover of P. □

Definition 4.2.2. Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem, 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 4.2.4. *Let $(\mathcal{F}, R, k)$ be a reduced instance of multicut on trees problem, 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 that does not contain the edge $w\pi(w)$. □

Definition 4.2.3. The instance $(\mathcal{F}, R, k)$ of multicut on trees problem is said to be *strongly reduced* if $(\mathcal{F}, R, k)$ is reduced and none of Reduction Rules 4.2.1 – 4.2.4 is applicable to the instance.

Proposition 4.2.4. *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 4.1.3) is not applicable to T.

(ii) This follows directly from the fact that Reduction Rule 4.2.1 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 4.2.1 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 4.2.1, 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 4.2.2 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 4.2.3 is not applicable to T.

(vi) This follows directly from the fact that Reduction Rule 4.2.4 is not applicable to T.

□

We are now ready to present the algorithm. Let $(\mathcal{F}, R, k)$ be an instance of multicut on trees problem. 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.

Observations. Let T be a tree in $\mathcal{F}$ rooted at r, let $w \neq r$ be an important vertex in T, and let u be a child of w such that u 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 u is contained in some minimum vertex cover of G_w, there is a minimum cut that contains the edge wu. Therefore, if we choose edge $w\pi(w)$ to be in the solution, then we can choose the edge wu to be in the solution as well. If when we branch we choose to cut uw whenever we cut $w\pi(w)$ then we say that we *favor* vertex u. Note that if we favor a vertex u, then by contrapositivity, if we decide not to cut u in a branch, then we can assume that w will not be cut as well in the same branch. This observation will be very useful when branching.

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 $deg_G(v)$ denotes the degree of v in G_w. By Lemma 4.2.1, 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 4.2.4, 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, $deg_G(v) \geq 1$.

The above observations lead to the following branching rule:

BranchRule 4.2.5. *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 $deg_G(v) \geq 3$, 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: $L(k) \leq L(k-1) + L(k-3)$.*[3]

[3]For the sake of obtaining an algorithm with the running time described in this thesis, it is sufficient to branch on vertices of degree ≥ 2 (rather than ≥ 3). However, we would like to present Reduction Rules 4.2.1 – 4.2.4 and Proposition 4.2.4 in a more general form to possibly make it easier to obtain algorithms with improved running time. As a matter of fact, as mentioned in section 4.3, we can obtain an algorithm with an improved running time over the algorithm presented in the current thesis at the expense of performing a complicated case-by-case analysis. The improved algorithm first branches on vertices of degree at least 3, and makes use of the reduction rules in the current thesis in addition to other reduction

After BranchRule 4.2.5, 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. We can assume that $w \neq r$; otherwise, by Branching Rule 4.2.5, we have $\Delta(G_w) \leq 2$, and the problem can be solved in polynomial time. 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 4.2.4, 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 a child of w and its grandparent $\pi(w)$, (3) between a child of w and an uncle, (4) between a child of w and a cousin, or (5) between w and a nephew of w. 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. We refer the reader to Figure 4.3 for an illustration of the different cases.

Case 1. 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 4.2.4, 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.

rules.

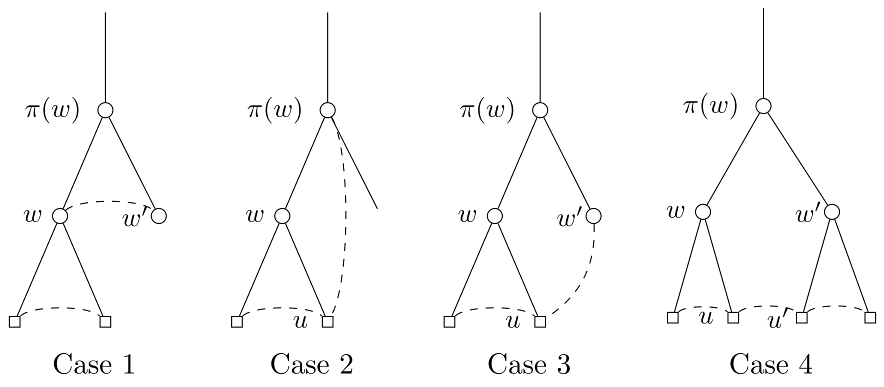


Figure 4.3: Illustration of the different cases treated by the algorithm. The dashed curved lines represent requests.

Case 2. 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 4.2.2, 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.

Case 3. 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. 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 4.2.6. *The multicut on trees 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 $x^2 - x - 1$ (for example, see [25, 29, 58]). Since the time spent by the algorithm along each root-leaf path in the search tree is polynomial, the theorem follows. $\square$

4.3 Conclusion

In this chapter, we presented a kernelization algorithm for the multicut on trees problem that computes a kernel of size at most $O(k^3)$ for the problem, improving the upper bound of $O(k^6)$ on the kernel size given in [13]. More specifically, in the analysis of our algorithm, we partition the nodes in the forest into three types of groups, and we bound the number of groups for each type in the forest. Then, we also bound the number of bottommost leaves through the connection with vertex cover problem. Eventually, with carefully analysis on the number of bad leaves, good leaves and internal nodes, we obtain a kernel of size $O(k^3)$ for the problem.

We also presented a parameterized algorithm that solves the multicut on trees problem in time $O^*(\rho^k)$, where $\rho = (\sqrt{5}+1)/2 \approx 1.618$ is the positive root of the polynomial $x^2 - x - 1$; this improves the $O^*(2^k)$-time algorithm given in [34]. The presented algorithm itself is a simple search-tree algorithm that distinguishes very few cases, and exploits the connection between multicut on trees problem and vertex cover problem. It is possible to obtain an algorithm with a slightly improved running time by distinguishing more cases based on the tree structure around the chosen important vertex. In fact, at the expense of a sophisticated case-by-case analysis, we can obtain an algorithm running in time $O^*(\rho'^k)$, where $\rho' = \sqrt{\sqrt{2}+1} \approx 1.553$ is the positive root of the polynomial $x^4 - 2x^2 - 1$. While the improved algorithm still uses the structural connection between multicut on trees problem and vertex cover problem, the numerous cases that need to be considered make the analysis very cumbersome, which makes it unworthy to present the improved algorithm. It is interesting to see if we can exploit the structural properties of the problem further to obtain improved algorithms without overcomplicating the analysis and case distinction; we leave this as an open problem.

5. METHODS AND RESULTS FOR PROTEIN COMPLEX PREDICTION *

In this chapter, we present our algorithms to predict protein complexes, and then we test our algorithms on three well known protein-protein interaction networks in yeast and evaluate the performances of our results with respect to several measures.

5.1 Methods to predict protein complex

5.1.1 Neighborhood density for protein complex prediction

Given a protein interaction network represented by a graph $G = (V, E)$, in which each vertex represents a protein and each edge represents interaction between two proteins, we first obtain the set C of all maximal cliques with at least three vertices by using a branch-and-bound algorithm to add one vertex at a time until no more vertices can be added. Since the degree of most vertices in G is small, the number of maximal cliques is not large and this step is feasible, with time complexity $O(|C||V|^2)$. Given two sets C_1 and C_2, we define their similarity to be $S(C_1, C_2) = |C_1 \cap C_2|^2/(|C_1||C_2|)$ [6] [4] [50] [19] [57] [72] [42] [64]. For each maximal clique C, we count the number of other maximal cliques C' with $S(C, C') \geq t$, where t is a given threshold. We define a clique C to have low neighborhood density if this number is below a given threshold c, and it has high neighborhood density otherwise. The worst case time complexity of this step is $O(|C|^2|V|)$.

5.1.2 Cliques with low neighborhood density

Given an induced subgraph $G_0 = (V_0, E_0)$ of a graph $G = (V, E)$, we define its density to be $D(G_0) = 2|E_0|/(|V_0|(V_0| - 1))$ [6], [4] [50] [57] [72] [64]. For each maximal clique with low neighborhood density, we iteratively identify the best vertex to add so that the density of the enlarged subgraph remains high and the length of the shortest path between two vertices in the enlarged subgraph remains small. We repeat the procedure until no

*Reprinted with permission from "Identifying Complexes from Protein Interaction Networks According to Different Types of Neighborhood Density" by Jia-Hao Fan, Jianer Chen and Sing-Hoi Sze, 2012. Journal of Computational Biology, 19(12): 1284-1294 (2012), Copyright [2012] by Mary Ann Liebert, Inc.

more changes can be made, and use the resulting dense subgraphs as predicted complexes (Figure 5.1). Since these subgraphs reside in regions with low neighborhood density, they are likely to function as an independent unit. For each potential vertex to add, it takes $O(|V|)$ time to compute the density of the enlarged subgraph. By precomputing all vertex pairs that have shortest paths of length at most two, the worst case time complexity of the procedure is $O(|C||V|^2)$, although it should terminate quickly when the density threshold is large.

```
NDComplexL
INPUT: a graph $G = (V, E)$, a set of cliques $\mathcal{C}$ in $G$ and density threshold $d$
OUTPUT: a set of $\mathcal{C}'$ of predicted complexes by adding vertices to cliques in $\mathcal{C}$.
1       $\mathcal{C}' \leftarrow \emptyset$
2       for each clique $C \in \mathcal{C}$
2.1       $C' \leftarrow C$
2.2       $V' \leftarrow V \setminus C$
2.3       while there exists a vertex $v \in V'$ with density $\mathcal{D}(C' \cup \{v\}) \geq d$
2.3.1       $v \leftarrow$ vertex in $V'$ with highest density $\mathcal{D}(C' \cup \{v\})$
2.3.2       if all shortest path between vertices in $C' \cup \{v\}$ are of length at most two
2.3.2.1       $C' \leftarrow C' \cup \{v\}$
2.3.3       $V' \leftarrow V' \setminus \{v\}$
2.4       $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{C'\}$
3       return $\mathcal{C}'$
```

Figure 5.1: Algorithm NDComplexL is used to obtain predicted complexes from maximal cliques with low neighborhood density.

5.1.3 Cliques with high neighborhood density

For each maximal clique with high neighborhood density, if its most similar maximal clique overlaps significantly with it, we replace it by the intersection of the two cliques. We repeat the procedure until no more changes can be made, and use the remaining cliques as predicted complexes (Figure 5.2). This procedure reduces the number of highly overlapping predictions significantly, since many cliques become identical after the intersections. By labeling each clique with a positive integer and picking the one with the lowest label when resolving ties in similarity, we can guarantee that at least two cliques become identical after each iteration, and the procedure takes at most $|C|$ iterations. The worst case time complexity of the procedure is $O(|C|^3|V|)$, although it should terminate quickly when the similarity threshold is large.

```
NDComplexH
INPUT: a graph $G = (V, E)$, a set of cliques $\mathcal{C}$ in $G$ and similarity threshold $s$
OUTPUT: a set of $\mathcal{C}'$ of predicted complexes by intersecting cliques $\mathcal{C}$.
1 while there exist cliques $C_1$ and $C_2$ in $\mathcal{C}$ with similarity $S(C_1, C_2) \geq s$
1.1   $\mathcal{C}' \leftarrow \emptyset$
1.2   for each clique $C \in \mathcal{C}$
1.2.1     $C' \leftarrow$ cliques in $\mathcal{C} \setminus \{C\}$ with hightest similarity $S(C, C')$
1.2.2     if $S(C, C') > s$
1.2.2.1       $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{C \cap C'\}$
1.2.3     else
1.2.3.1       $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{C\}$
1.3   $\mathcal{C} \leftarrow \mathcal{C}'$
2 return $\mathcal{C}'$
```

Figure 5.2: Algorithm NDComplexH is used to obtain predicted complexes from maximal cliques with high neighborhood density.

5.2 Experimental results

5.2.1 Performance evaluation

We use a combined set of true complexes that contain at least two proteins to which we compare the predicted complexes, including 214 curated complexes from the Munich Information Center for Protein Sequences (MIPS) database [56], 101 curated complexes from Aloy et al.[3], and 363 complexes extracted from the Saccharomyces Genome Database [41], according to GO slim complex annotations [30]. To reduce evaluation bias, we have removed complexes that contain more than 100 proteins from the SGD complexes. We removed the duplicates from the combined set, resulting in a total of 574 true complexes. Most of these complexes are small, with the average number of proteins within a complex being 9.4 (Table 5.1). We used the yeast protein interaction network from the MIPS database [56], the yeast protein interaction network from the Database of Interacting Proteins (DIP database [73]), and the yeast protein interaction network from the Biological General Repository for Interaction Datasets (BioGRID database [66]) to obtain separate sets of predicted complexes from each network. For the BioGRID network, only physical interactions are included. These networks have different densities, with the BioGRID network being the densest (Table 5.2).

Table 5.1: Distribution of True Complexes Used in the Evaluation

Protein	Complex	Protein	Complex	Protein	Complex
2	116	5	47	8	24
3	96	6	32	9	13
4	67	7	32	$\leq$ 10	147

[a] Complex denotes the number of complexes that have number of proteins specified by Protein.

5.2.2 Performance comparisons

We compared the performance of our algorithm NDComplex to the Markov cluster algorithm (MCL) [27], which subdivides a given graph into clusters by Markov clustering,

Table 5.2: Protein Interaction Networks Used in the Evaluation

	Protein	Interaction	Avg_deg	Max_deg	Density
MIPS	4546	12319	5.4	286	0.0012
DIP	4945	21639	8.8	281	0.0018
BioGRID	5727	51319	17.9	2553	0.0031

[a] Protein, interaction, avg_deg, max_deg, and density denote the number of proteins, the number of interactions, the average vertex degree, the maximum vertex degree, and the density of the network, respectively.

to the Molecular complex detection algorithm (MCODE) [6], which uses a seed-extension algorithm to identify complexes in dense regions, to the Dense-neighborhood extraction using connectivity and confidence features algorithm (DECAFF) [50], which is based on the identification and merging of dense subgraphs, and to the Core-attachment based method (COACH) [72], which is based on the prediction of a core complex and its attachment proteins.We also compared the performance of our algorithm to the Maximal clique algorithm (CLIQUE), which uses the set of maximal cliques with at least three vertices as predicted complexes. For MCL, we set inflation to 3.5. For MCODE, we set depth to 100, haircut to true, fluff to false, and fluff density threshold to 0.2. For DECAFF, we implement steps 1 to 4 of Algorithm 1 in Li et al.[50], including the local clique detection step, the hub removal step, and the merging step, and set the density threshold to 0.8 and the neighborhood affinity threshold to 0.5. Since the other algorithms do not use functional information, we skip the filtering steps 5 to 9 in DECAFF that use the MIPS functional catalog [56]. For COACH, we set the neighborhood affinity threshold to 0.1. For NDComplex, we set the similarity threshold t and the occurrence threshold c during the computation of neighborhood density to 0.3 and 3 respectively. We set the density threshold d to 0.7 during the computation of predicted complexes in regions with low neighborhood density, and the similarity threshold s to 0.2 during the computation of predicted complexes in regions with high neighborhood density. These parameters are determined by testing a few combinations and choosing the one that gives the best overall performance on the test sets. We collect the predicted complexes from both types of regions for performance evaluation, with each distinct prediction counted once.

5.2.3 Complex agreement measure

Given a similarity threshold u, we evaluate the agreement between a set of true complexes and a set of predicted complexes by defining the precision (PC) to be the ratio of the number of predicted complexes P that have a true complex C with similarity $S(C, P) \geq u$ to the total number of predicted complexes, and the recall (RC) to be the ratio of the number of true complexes C that have a predicted complex P with similarity $S(C, P) \geq u$ to the total number of true complexes [19] [57] [72] [42] [64]. We compute the F-measure $= 2 \times (PC \times RC)/(PC + RC)$. Figure 5.3 shows that NDComplex has the best overall performance when the similarity threshold u is low, while MCODE and COACH perform better when the similarity threshold u is high, which corresponds to a small number of almost perfect predictions. DECAFF has the next best performance, followed by CLIQUE and MCL.

5.2.4 Complex accuracy measure

In addition to using the complex agreement measure, we use the complex accuracy measure in Brohée and van Helden [14], Friedel et al.[30], and Tu et al.[67] that does not rely on the use of a similarity threshold. Given a set of true complexes C and a set of predicted complexes P, we compute the sensitivity $S_n = \sum_{C \in \mathcal{C}} max_{P \in \mathcal{P}} |C \cap P| / \sum_{C \in \mathcal{C}} |C|$, which is the ratio of the sum of the maximum overlap of each true complex with a predicted complex to the total size of true complexes, the positive predictive value $PPV = \sum_{P \in \mathcal{P}} max_{C \in \mathcal{C}} |C \cap P| / \sum_{P \in \mathcal{P}} |P|$, which is the ratio of the sum of the maximum overlap of each predicted complex with a true complex to the total size of predicted complexes, and the accuracy $Acc = \sqrt{S_n \times PPV}$, which is the geometric mean of sensitivity and PPV. Figure 5.4 shows that NDComplex has the best accuracy in almost all cases, followed by

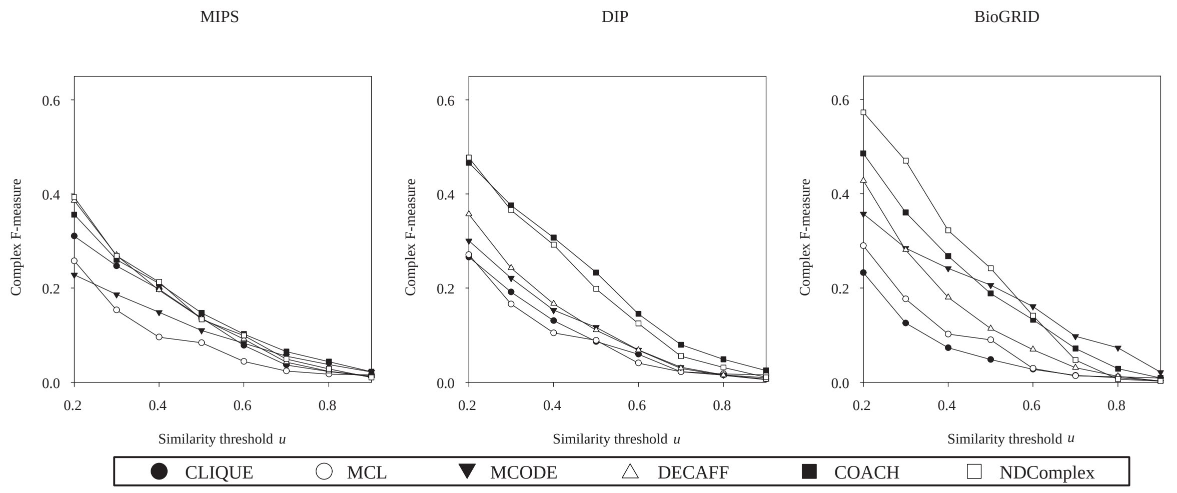


Figure 5.3: Performance of complex prediction algorithms on each protein interaction network with respect to the complex agreement measure over different similarity thresholds u between a true complex and a predicted complex. For DECAFF, only steps 1 to 4 of Algorithm 1 in Li et al. [50] are included, while the filtering steps 5 to 9 that are based on the use of functional information are skipped.

CLIQUE, DECAFF, and COACH. The performance differences between NDComplex and MCL, MCODE, DECAFF, or COACH are the largest on the BioGRID network. DECAFF puts emphasis on sensitivity, while MCODE puts emphasis on PPV.

5.2.5 Protein pair agreement measure

We evaluate the protein pair agreement by defining a true positive (TP) to be a protein pair that is within the same true complex and within the same predicted complex, a false positive (FP) to be a protein pair that is within the same predicted complex but not in the same true complex, and a false negative (FN) to be a protein pair that is within the same true complex but not in the same predicted complex. We compute the precision $PC = |TP|/(|TP| + |FP|)$, the recall $RC = |TP|/(|TP| + |FN|)$, and the F-measure $= 2 \times (PC \times RC)/(PC + RC)$. When the true complexes and the predicted complexes both consist of disjoint sets of proteins, this measure evaluates the resemblance of the two

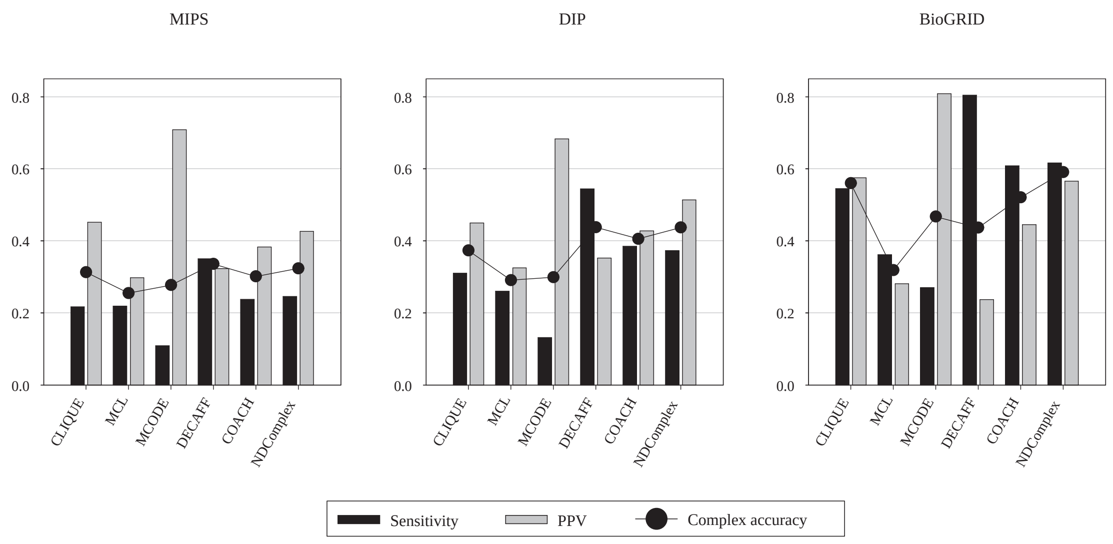


Figure 5.4: Performance of complex prediction algorithms on each protein interaction network with respect to the complex accuracy measure.

sets. Since a protein can appear in more than one complex, this measure evaluates how accurately an algorithm can predict whether a given pair of proteins are within the same complex or not, although this is only an approximation since the set of true complexes may not be complete. Figure 5.5 shows that COACH and NDComplex have the best performance with respect to protein pairs. The performance differences between these algorithms and MCL, MCODE, or DECAFF are especially large on the BioGRID network, with CLIQUE performing better than MCL, MCODE, and DECAFF. MCL and DECAFF have low performance on the BioGRID network due to a large number of false positive protein pairs. MCODE puts emphasis on precision, while DECAFF puts emphasis on recall.

5.3 Discussion

We have developed an algorithm to identify complexes from protein interaction networks based on using different strategies for regions with different neighborhood densities.

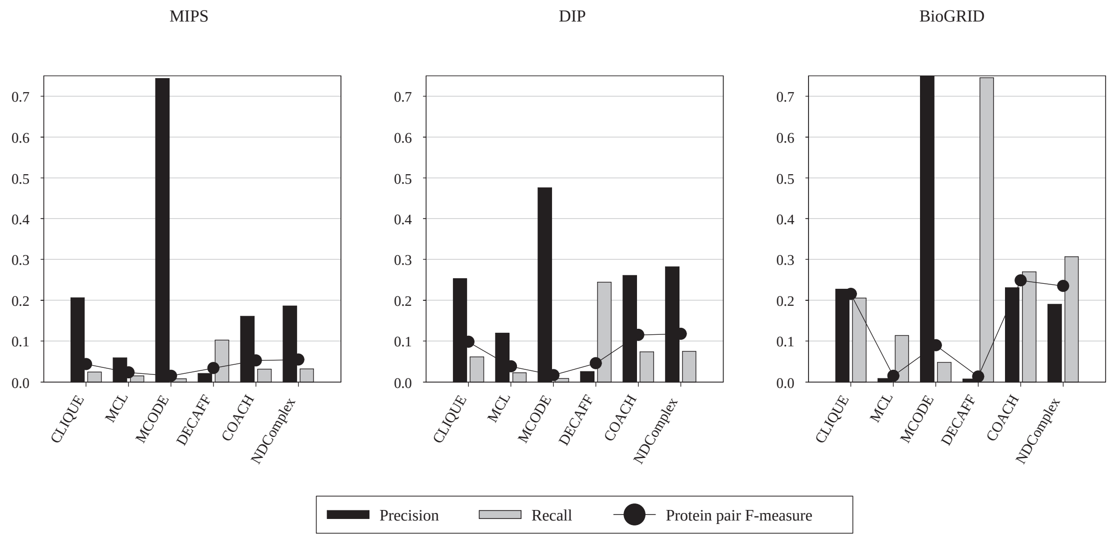


Figure 5.5: Performance of complex prediction algorithms on each protein interaction network with respect to the protein pair agreement measure.

We have shown that this approach is very effective and it achieves the best performance with respect to complex agreement, complex accuracy, and protein pair agreement measures. Among the three networks that we have tested, we found that each algorithm performs the best on the BioGRID network, which has the highest density, with weaker performance on the DIP network, followed by the MIPS network, which has the lowest density. On the BioGRID network, our algorithm NDComplex performs the best with respect to the complex agreement measure when the similarity threshold u is low, and with respect to the complex accuracy measure, while COACH and NDComplex have the best and comparable performance with respect to the protein pair agreement measure.

Ideally, the correspondences between the true complexes and the predicted complexes should be one-to-one. To evaluate the degree of success of each algorithm in reaching this goal, we use the separation measure in Brohée and van Helden [14], and Tu et al.[67]. Given a true complex C and a predicted complex P, define their separation $Sep(C, P) = |C \cap P|^2/(\sum_{C \in \mathcal{C}} |C \cap P| \sum_{P \in \mathcal{P}} |C \cap P|)$. Given a set of true complexes $\mathcal{C}$ and a set of predicted complexes $\mathcal{P}$, we compute the average separation over the set of true complexes $Sep_c = \sum_{C\in\mathcal{C}}\sum_{P\in\mathcal{P}} SEP(C,P)/|\mathcal{C}|$, the average separation over the set of predicted complexes $Sep_p = \sum_{C\in\mathcal{C}}\sum_{P\in\mathcal{P}} SEP(C,P)/|\mathcal{P}|$, and the separation $Sep = \sqrt{Sep_c \times Sep_p}$, which is the geometric mean of the above two averages.

Table 5.4 shows that while MCODE has the highest separation, it produces a small number of predictions that cover very few proteins. MCL achieves the next highest separation, but it produces a disjoint partition of proteins, which is biologically inaccurate since it does not allow a protein to appear in multiple predicted complexes. COACH achieves a better separation than NDComplex, but it covers less proteins. CLIQUE and DECAFF cover the most interactions, but the number of predictions is very high and they have the worst separation. NDComplex achieves better separation than CLIQUE and DECAFF.

Table 5.3: Running Time in Seconds During Different Stages of Our Algorithm NDComplex on Each Protein Interaction Network

	Clique	Neighbor	Complex_l	Complex_h
MIPS	420	39	88	41
DIP	908	184	101	212
BioGRID	15739	13873	2950	24722

[a] Clique denotes the time to obtain all the maximal cliques; neighbor denotes the time to compute the neighborhood density of these cliques; and complex_l and complex_h denote the time to obtain predicted complexes from cliques with low neighborhood density and high neighborhood density, respectively.

Table 5.5 shows the statistics of maximal cliques and predicted complexes during different stages of our algorithm NDComplex. Within each network, the number of maximal cliques with high neighborhood density is much larger than the number of maximal cliques with low neighborhood density, which means that there are significant overlaps among the maximal cliques with high neighborhood density. For regions with low neighborhood density, our strategy can add a large number of proteins to a prediction, which is especially evident on the BioGRID network. While the density of these predictions remains high, the number of predictions decreases since some of them become identical after the addition of proteins. The situation is very different for regions with high neighborhood density, when the number of predictions decreases drastically after the intersections of cliques, and each of these predictions contains a very small number of proteins, which means that there are only a small number of highly shared parts within these cliques. The predictions from these two types of regions are distinct (compare to Table 5.4), with almost all predictions coming from regions with low neighborhood density. Since this is sufficient to ensure that the predictions have high quality, a complex can be modeled mostly as an independent unit with dense inside connections and sparse outside connections, while the small number of predictions from very dense regions are still needed to reduce false negatives.

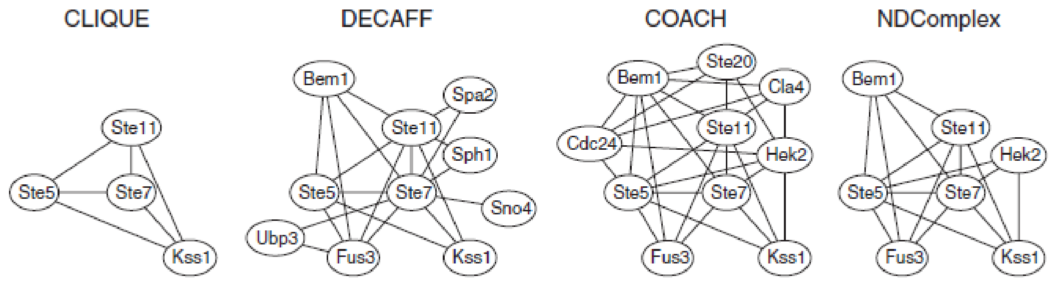


Figure 5.6: Predicted complex from each algorithm on the protein interaction network from the BioGRID database that has the highest similarity to the true complex that contains the MAP kinase cascade of the pheromone response pathway and the filamentation/invasion pathway from the MIPS database. No predicted complexes from MCL or MCODE have overlap to the true complex and are not shown.

Although the worst case time complexity of our algorithm NDComplex is high, the actual running time is not high (Table 5.3). For sparse networks such as MIPS and DIP, it takes less than an hour to complete all the steps, and the time to obtain the maximal cliques dominates. For denser networks such as BioGRID, it takes about a day, and the time to obtain predicted complexes from cliques with high neighborhood density dominates.

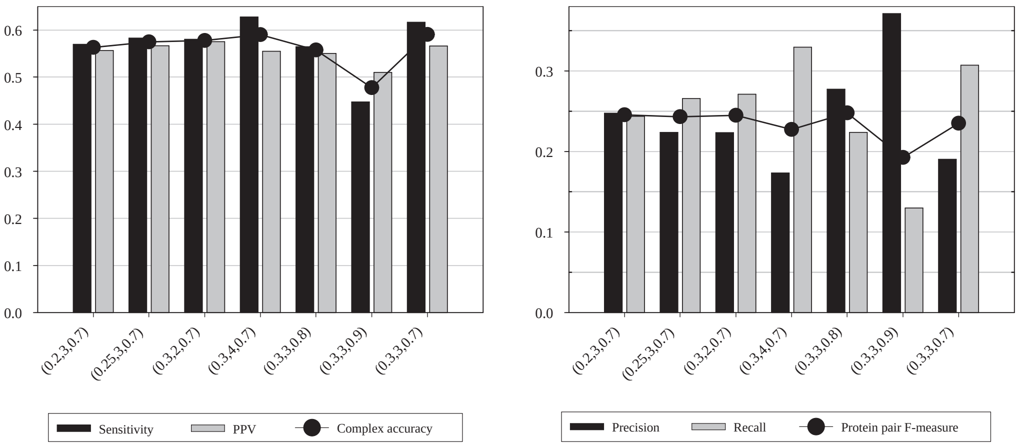


Figure 5.7: Performance of our algorithm NDComplex on the protein interaction network from the BioGRID database with respect to the complex accuracy measure and the protein pair agreement measure over different parameter settings (t, c, d), where t and c are the similarity threshold and the occurrence threshold, respectively, during the computation of neighborhood density, and d is the density threshold during the computation of predicted complexes in regions with low neighborhood density. The similarity threshold during the computation of predicted complexes in regions with high neighborhood density is fixed to $s = 0.2$. The last set of parameters $(0.3, 3, 0.7)$ is the one we chose.

To investigate the effect of different parameters on our algorithm NDComplex, we picked a few sets of parameters that are close to the one we choose and examine the performance differences. Figure 5.7 shows that NDComplex is not very sensitive to parameters.

To illustrate the differences in predictions that can be obtained from different algorithms, we examine the predicted complex from each algorithm on the BioGRID network that has the highest similarity to the true complex that contains the MAP kinase cascade of the pheromone response pathway and the filamentation/ invasion pathway [35] from the MIPS database. Figure 5.6 shows that CLIQUE finds the complex that contains the MAP kinase cascade of the filamentation/invasion pathway, while the protein Fus3 in the true complex is not included. NDComplex expands the complex to include two extra proteins Bem1 and Hek2, in addition to all the proteins in the true complex. COACH and DECAFF return the largest complexes that contain a few extra proteins, while MCL and MCODE do not return a complex that has overlap to the true complex.

One drawback of our algorithm is that it takes exponential time to identify all the maximal cliques, thus it is not likely that it will scale up to handle dense networks. One future direction is to investigate whether it is possible to develop polynomial time heuristics without a large decrease in prediction performance.

Table 5.4: Statistics of Predicted Complexes from Each Algorithm on Each Protein Interaction Network

	MIPS						DIP						BioGrid					
	CLIQUE	MCL	MCODE	DECAFF	COACH	NDComplex	CLIQUE	MCL	MCODE	DECAFF	COACH	NDComplex	CLIQUE	MCL	MCODE	DECAFF	COACH	NDComplex
Complex	2869	1740	57	5328	280	621	7129	2226	52	11434	467	1180	34383	2213	76	35048	672	2600
Protein	1387	4546	233	3550	1024	1144	2161	4945	265	4433	1530	1800	4429	5727	553	5707	2675	3217
Interaction	5648	3051	459	9364	3692	4016	11498	2971	489	18623	6392	7193	42980	6466	2344	49899	22663	27101
Separation	0.0055	0.034	0.046	0.0012	0.029	0.014	0.003	0.033	0.046	0.0006	0.025	0.011	0.0009	0.028	0.051	0.000079	0.016	0.0055

[a] Complex denotes the number of predicted complexes; protein and interaction denote the number of proteins and interactions covered by these complexes, respectively, in the network; and separation denotes the separation between the set of true complexes and the set of predicted complexes.

Table 5.5: Statistics of Maximal Cliques and Predicted Complexes During Different Stages of Our Algorithm

	MIPS				DIP				BioGrid			
	clique_l	clique_h	complex_l	complex_h	clique_l	clique_h	complex_l	complex_h	clique_l	clique_h	complex_l	complex_h
Number	735	2134	595	26	1333	5796	1159	21	2715	31668	2554	46
Avg_pro	3.2	4.7	8.1	2.5	3.2	3.7	6.6	2.5	3.3	7.7	19.0	2.8
Max_pro	14	12	27	4	9	10	22	4	10	33	80	7

[a] Clique_l and complex_l denote cliques with low neighborhood density and predicted complexes from these cliques, respectively; and clique_h and complex_h denote cliques with high neighborhood density and predicted complexes from these cliques, respectively. In each case, number denotes the total number of cliques or predicted complexes, and avg_pro and max_pro denote the average and maximum number of proteins within a clique or a predicted complex, respectively.

6. CONCLUSIONS AND FUTURE RESEARCH

In this thesis, we developed algorithms for three cut and partition problems in graphs and trees, and studied their applications. The three problems are the maximum agreement forest problem, the multicut on trees problem and the protein complex prediction problem. We summarize our results and describe future work to each of these three problems in the following paragraphs.

For the maximum agreement forest problem, we presented a parameterized algorithm and an approximation algorithm on unrooted general trees. Our parameterized algorithm is the first fixed-parameter tractable algorithm, which resolves an open problem posed in the literature [36, 71]. Our approximation algorithm is also the first constant-ratio approximation algorithm on unrooted genearl trees, which matches the best known approximation ratio to the problem in binary trees [70, 71]. The problem is motivated in the study of evolution trees in bioinformatics. Our algorithms could therefore be applied to computing the similarity between phylogenetic trees and reveal the evolutionary relationship among species, efficiently. In addition, our algorithms to the maximum agreement forest problem are based on the concept of a bottommost sibling set in one tree and the structure of the corresponding leaf set in the other tree. Our techniques should be useful for the study on parameterized and approximation algorithms for other problems related to phylogenetic similarity, such as those related to the SPR distance, the rooted SPR distance, and the hybridization distance. We are currently working on these extensions and aimed at new and more efficient algorithms for these problems.

For multicut on trees problem, we presented a kernelization algorithm that computes a kernel of size at most $O(k^3)$, improving the upper bound of $O(k^6)$ on the kernel size given in [13]. We also presented a parameterized algorithm that solves the multicut on trees problem in time $O^*(\rho^k)$, where $\rho = (\sqrt{5}+1)/2 \approx 1.618$ is the positive root of the polynomial $x^2 - x - 1$; this improves the $O^*(2^k)$-time algorithm given in [34]. The presented algorithm itself is a simple search-tree algorithm that distinguishes very few cases, and exploits the connection between multicut on trees problem and vertex cover problem. The multicut on trees problem has applications in networking. Our algorithms to multicut on trees problem can facilitate the performance of relative problems in networking. With respect to the parameterized algorithms, improving the $O(k^3)$ upper bound further is an interesting open problem. On the other hand, with respect to the kernelization algorithms, the polynomial kernels to multiway cut problem and multicut problem are still unknwon. It is also interesting for finding polynomial kernels to multiway cut problem or some other relative problems.

For protein complex prediction problem, first, we formulated the problem as a graph clustering problems. Then, we observed that most complexes either reside in very dense regions or they reside in regions with low neighborhood density. Base on these observations, we develop an algorithm to predict protein complexes by refining the maximal cliques in protein-protein interaction networks. We tested our algorithm on three most popular protein-protein interaction networks in yeast (MIPS [56], DIP [73] and BioGRID [66]) and compared our results with the curated protein complexes. We shown that our predictions are more accurate than any other algorithms. In addition, we implemented our algorithms as a software application which can be found at following hyperlink, http://faculty.cs.tamu.edu/shsze/ndcomplex/. Currently, most research papers on protein complexes prediction are limited to yeast. It is worthy to extend our algorithms to predicting the protein complexes in other species.

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] B. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Ann. Comb.*, 5:2001, 2000.

[3] P. Aloy, B. Böttcher, H. Ceulemans, C. Leutwein, C. Mellwig, S. Fischer, A.-C. Gavin, P. Bork, G. Superti-Furga, L. Serrano, and R. Russell. Structure-based assembly of protein complexes in yeast. *Science*, 303(5666):2026–2029, 2004.

[4] M. Altaf-Ul-Amin, Y. Shinbo, K. Mihara, K. Kurokawa, and S. Kanaya. Development and implementation of an algorithm for detection of protein complexes in large interaction networks. *BMC Bioinformatics*, 7(1):207, 2006.

[5] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM J. Comput.*, 26(6):1656–1669, 1997.

[6] G. Bader and C. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, 2003.

[7] E. Bapteste, M. O'Malley, R. Beiko, M. Ereshefsky, J. Gogarten, L. Franklin-Hall, F. Lapointe, J. Dupre, T. Dagan, Y. Boucher, and W. Martin. Prokaryotic evolution and the tree of life are two different things. *Biol. Direct*, 4:34, 2009.

[8] C. Bentz, M.-C. Costa, L. Létocart, and F. Roupin. Erratum to "minimal multicut and maximal integer multiflow: A survey" [european journal of operational research 162 (1) (2005) 55-69]. *European Journal of Operational Research*, 177(2):1312, 2007.

[9] M. Bonet, K. John, R. Mahindru, and N. Amenta. Approximating subtree distances between phylogenies. *J. Comput. Biol.*, 13(8):1419–1434, 2006.

[10] M. Bordewich, C. McCartin, and C. Semple. A 3-approximation algorithm for the subtree distance between phylogenies. *J. Discrete Alg.*, 6(3):458–471, 2008.

[11] M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Ann. Comb.*, 8(4):409–423, 2005.

[12] N. Bousquet, J. Daligault, and S. Thomassé. Multicut is fpt. In *STOC*, pages 459–468, 2011.

[13] 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.

[14] S. Brohée and J. Van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7(1):488, 2006.

[15] J. Buss and J. Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22:560–572, 1993.

[16] G. Calinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *J. Comput. Syst. Sci.*, 60(3):564–574, 2000.

[17] F. Chataigner. Approximating the maximum agreement forest on k trees. *Inf. Process. Lett.*, 93(5):239–244, 2005.

[18] J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

[19] H. N. Chua, W.-K. Sung, and L. Wong. Using indirect protein interactions for the prediction of gene ontology functions. *BMC Bioinformatics*, 8(S-4), 2007.

[20] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[21] M. Costa, L. Letocart, and F. Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55–69, 2005.

[22] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. Wojtaszczyk. On multiway cut parameterized above lower bounds. *CoRR*, abs/1107.1585, 2011.

[23] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[24] B. DasGupta, X. He, T. Jiang, M. Li, and J. Tromp. On the linear-cost subtree-transfer distance between phylogenetic trees. *Algorithmica*, 25(2-3):176–195, 1999.

[25] R. Downey and M. Fellows. *Parameterized Complexity (Monographs in Computer Science)*. Springer-verlag, Berlin, Germany, 1998.

[26] J. Edmonds and R. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[27] A. Enright, S. Van Dongen, and C. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.*, 30(7):1575–1584, 2002.

[28] M. Farach and M. Thorup. Fast comparison of evolutionary trees. In *SODA*, pages 481–488, 1994.

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

[30] C. Friedel, J. Krumsiek, and R. Zimmer. Bootstrapping the interactome: unsupervised identification of protein complexes in yeast. In *Research in Computational Molecular Biology*, pages 3–16. Springer, 2008.

[31] N. Garg, V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput.*, 25(2):235–251, 1996.

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

[33] A.-C. Gavin, M. Bösche, R. Krause, P. Grandi, M. Marzioch, A. Bauer, J. Schultz, J. Rick, A.-M. Michon, C.-M. Cruciat, M. Remor, C. Hofert, M. Schelder, M. Brajenovic, H. Ruffner, A. Merino, K. Klein, M. Hudak, D. Dickson, T. Rudi, V. Gnau, A. Bauch, S. Bastuck, B. Huhse, C. Leutwein, M.A. Heurtier, R.R. Copley, A. Edelmann, E. Querfurth, V. Rybin, G. Drewes, M. Raida, T. Bouwmeester, P. Bork, B. Seraphin, B. Kuster, G. Neubauer, and G. Superti-Furga. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–147, 2002.

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

[35] M. Gustin, J. Albertyn, M. Alexander, and K. Davenport. Map kinase pathways in the yeastsaccharomyces cerevisiae. *Microbiology and Molecular Biology Reviews*, 62(4):1264–1300, 1998.

[36] M. Hallett and C. McCartin. A faster FPT algorithm for the maximum agreement forest problem. *Theory Comput. Syst.*, 41(3):539–550, 2007.

[37] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Appl. Mathematics*, 71(1-3):153–169, 1996.

[38] E. Hirsh and R. Sharan. Identification of conserved protein complexes based on a model of protein network evolution. *Bioinformatics*, 23(2):170–176, 2007.

[39] Y. Ho, A. Gruhler, A. Heilbut, G. Bader, L. Moore, S.-L. Adams, A. Millar, P. Taylor, K. Bennett, K. Boutilier, L. Yang, C. Wolting, I. Donaldson, S. Schandorff, J. Shew-

narane, M. Vo, J. Taggart, M. Goudreault, B. Muskat, C. Alfarano, D. Dewar, Z. Lin, K. Michalickova, A.R. Willems, H. Sassi, P.A. Nielsen, K.J. Rasmussen, J.R. Andersen, L.E. Johansen, L.H., L.H. Hansen, H. Jespersen, A. Podtelejnikov, E. Nielsen, J. Crawford, V. Poulsen, B.D. Sorensen, J. Matthiesen, R.C. Hendrickson RC, F. Gleeson, T. Pawson, M.F. Moran, D. Durocher, M. Mann, C.W. Hogue, D. Figeys, and M. Tyers. Systematic identification of protein complexes in saccharomyces cerevisiae by mass spectrometry. *Nature*, 415(6868):180–183, 2002.

[40] W.-K. Hon and T. Lam. Approximating the nearest neighbor interchange distance for evolutionary trees with non-uniform degrees. In *COCOON*, pages 61–70, 1999.

[41] L. Issel-Tarver, K. Christie, K. Dolinski, R. Andrada, R. Balakrishnan, C. Ball, G. Binkley, S. Dong, S. Dwight, D. Fisk, M. Harris, M. Schroeder, A. Sethuraman, K. Tse, S. Weng, D. Botstein, and J.M. Cherry JM. Saccharomyces genome database. *Methods in Enzymology*, 350:329, 2002.

[42] S. H. Jung, B. r. Hyun, W.-H. Jang, H.-Y. Hur, and D.-S. Han. Protein complex prediction based on simultaneous protein interaction network. *Bioinformatics*, 26(3):385–391, 2010.

[43] D. Karger, P. Klein, C. Stein, M. Thorup, and N. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. *Math. Oper. Res.*, 29(3):436–461, 2004.

[44] A. D. King, N. Przulj, and I. Jurisica. Protein complex prediction via cost-based clustering. *Bioinformatics*, 20(17):3013–3020, 2004.

[45] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *FOCS*, pages 726–737, 1990.

[46] R. Kliman, P. Andolfatto, J. Coyne, F. Depaulis, M. Kreitman, A. Berry, J. McCarter, J. Wakeley, and J. Hey. The population genetics of the origin and divergence of the *Drosophila simulans* complex species. *Genetics*, 156:1913–1931, 2000.

[47] N. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. Tikuisis, T. Punna, J.M. Peregrin-Alvarez, M. Shales, X. Zhang, M. Davey, M.D. Robinson, A. Paccanaro, J.E. Bray, A. Sheung, B. Beattie, D.P. Richards, V. Canadien, A. Lalev, F. Mena, P. Wong, A. Starostine, M.M. Canete, J. Vlasblom, S. Wu, C. Orsi, S.R. Collins, S. Chandran, R. Haw, J.J. Rilstone, K. Gandi, N.J. Thompson, G. Musso, P. St Onge, S. Ghanny, M.H. Lam, G. Butland, A.M. Altaf-Ul, S. Kanaya, A. Shilatifard, E. O'Shea, J.S. Weissman, C.J. Ingles, T.R. Hughes, J. Parkinson, M. Gerstein, S.J. Wodak, A. Emili, and J.F. Greenblatt. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084):637–643, 2006.

[48] S. Leibler A. Murray L. Hartwell, J. Hopfield. From molecular to modular cell biology. *Nature*, 402:C47–C52, 1999.

[49] H. Leung, Q. Xiang, SM Yiu, and F. Chin. Predicting protein complexes from ppi data: a core-attachment approach. *Journal of Computational Biology*, 16(2):133–144, 2009.

[50] X.-L. Li, C.-S. Foo, and S.-K. Ng. Discovering protein complexes in dense reliable neighborhoods of protein interaction networks. In *Comput Syst Bioinformatics Conf*, volume 6, pages 157–168, 2007.

[51] G. Lin, C. Zhang, and D. Xu. Polytomy identification in microbial phylogenetic reconstruction. *BMC Sys. Biol.*, 5(Suppl 3):S2, 2011.

[52] S. Linz and C. Semple. Hybridization in nonbinary trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6:30–45, 2009.

[53] G. Liu, L. Wong, and H. N. Chua. Complex discovery from weighted ppi networks. *Bioinformatics*, 25(15):1891–1897, 2009.

[54] D. Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

[55] D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *STOC*, pages 469–478, 2011.

[56] H. W. Mewes, D. Frishman, K. F. X. Mayer, M. Munsterkotter, O. Noubibou, T. Rattei, M. Oesterheld, and V. Stumpflen. Mips: Analysis and annotation of proteins from whole genomes. *Nucleic Acids Res.*, 32:41–44, 2004.

[57] C. Moschopoulos, G. Pavlopoulos, R. Schneider, S. Likothanassis, and S. Kossida. Giba: a clustering tool for detecting protein complexes. *BMC Bioinformatics*, 10(S-6), 2009.

[58] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31. Oxford University Press, New York, U.S.A., 2006.

[59] Y. Qi, F. Balem, C. Faloutsos, J. Klein-Seetharaman, and Z. Bar-Joseph. Protein complex identification by supervised graph local clustering. In *ISMB*, pages 250–268, 2008.

[60] I. Razgon. Large isolating cuts shrink the multiway cut. *CoRR*, abs/1104.5361, 2011.

[61] E. Rodrigues, M. Sagot, and Y. Wakabayashi. Some approximation results for the maximum agreement forest problem. In *RANDOM-APPROX*, pages 159–169, 2001.

[62] E. Rodrigues, M. Sagot, and Y. Wakabayashi. The maximum agreement forest problem: Approximation algorithms and computational experiments. *Theor. Comput. Sci.*, 374(1-3):91–110, 2007.

[63] R. Sharan, T. Ideker, B. Kelley, R. Shamir, and R. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *Journal of Computational Biology*, 12(6):835–846, 2005.

[64] L. Shi, Y. Lei, and A. Zhang. Protein complex detection with semi-supervised learning in protein interaction networks. *Proteome Science*, 9:1–9, 2011.

[65] V. Spirin and L. Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences*, 100(21):12123–12128, 2003.

[66] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. Biogrid: a general repository for interaction datasets. *Nucleic Acids Res.*, 34(Database issue):D535–D539, 2006.

[67] S. Tu, R. Chen, and L. Xu. A binary matrix factorization algorithm for protein complex prediction. *Proteome Science*, 9(Suppl 1):S18, 2011.

[68] D. West. *Introduction to Graph Theory.* Prentice Hall Inc., Upper Saddle River, NJ, 1996.

[69] D. Wheeler, C. Chappey, A. Lash, D. Leipe, T. Madden, G. Schuler, T. Tatusova, and B. Rapp. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, 28:10–14, 2000.

[70] C. Whidden, R. Beiko, and N. Zeh. Fixed-parameter and approximation algorithms for maximum agreement forests. *CoRR*, abs/1108.2664, 2011.

[71] C. Whidden and N. Zeh. A unifying view on approximation and FPT of agreement forests. In *WABI*, pages 390–402, 2009.

[72] M. Wu, X. Li, C.-K. Kwoh, and S.-K. Ng. A core-attachment based method to detect protein complexes in ppi networks. *BMC Bioinformatics*, 10(1):169, 2009.

[73] I. Xenarios, E. Fernandez, L. Salwinski, X. Duan, M. Thompson, E. Marcotte, and D. Eisenberg. Dip: The database of interacting proteins: 2001 update. *Nucleic Acids Res.*, 29:239–241, 2001.

[74] M. Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory Comput. Syst.*, 46(4):723–736, 2010.