RANDOMIZED AND DETERMINISTIC PARAMETERIZED ALGORITHMS AND THEIR APPLICATIONS IN BIOINFORMATICS

A Dissertation

by

SONGJIAN LU

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

December 2009

Major Subject: Computer Science

RANDOMIZED AND DETERMINISTIC PARAMETERIZED ALGORITHMS AND THEIR APPLICATIONS IN BIOINFORMATICS

A Dissertation

by

SONGJIAN LU

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

Approved by:

Co-Chairs of Committee,	Jianer Chen
	Sing-Hoi Sze
Committee Members,	Sergiy Butenko
	Donald Friesen
Head of Department,	Valerie E. Taylor

December 2009

Major Subject: Computer Science

ABSTRACT

Randomized and Deterministic Parameterized Algorithms
and Their Applications in Bioinformatics. (December 2009)
Songjian Lu, B.S., Guangxi University;
M.S., Xi'an Jiaotong University;
M.S., University of Houston
Co–Chairs of Advisory Committee: Dr. Jianer Chen
Dr. Sing-Hoi Sze

Parameterized NP-hard problems are NP-hard problems that are associated with special variables called parameters. One example of the problem is to find simple paths of length k in a graph, where the integer k is the parameter. We call this problem the P-PATH problem. The P-PATH problem is the parameterized version of the well-known NP-complete problem – the longest simple path problem.

There are two main reasons why we study parameterized NP-hard problems. First, many application problems are naturally associated with certain parameters. Hence we need to solve these parameterized NP-hard problems. Second, if parameters take only small values, we can take advantage of these parameters to design very effective algorithms.

If a parameterized NP-hard problem can be solved by an algorithm of running time in form of $f(k)n^{O(1)}$, where k is the parameter, $f(k)$ is independent of n, and n is the input size of the problem instance, we say that this parameterized NP-hard problem is fixed parameter tractable (FPT). If a problem is FPT and the parameter takes only small values, the problem can be solved efficiently (it can be solved almost in polynomial time).

In this dissertation, first, we introduce several techniques that can be used to design efficient algorithms for parameterized NP-hard problems. These techniques include branch and bound, divide and conquer, color coding and dynamic programming, iterative compression, iterative expansion and kernelization. Then we present our results about how to use these techniques to solve parameterized NP-hard problems, such as the P-PATH problem and the PD-FEEDBACK VERTEX SET problem. Especially, we designed the first algorithm of running time in form of $f(k)n^{O(1)}$ for the PD-FEEDBACK VERTEX SET problem. Thus solved an outstanding open problem, i.e. if the PD-FEEDBACK VERTEX SET problem is FPT. Finally, we will introduce how to use parameterized algorithm techniques to solve the signaling pathway problem and the motif finding problem from bioinformatics.

To my father and mother

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Jianer Chen, for all his help and guidance during my Ph.D. studies at Texas A&M University. Without his guidance and encouragement, it is impossible for me to make such a success in my research. I also enjoyed every class he taught.

Then, I would like to thank my advisor, Dr. Sing-Hoi Sze for all his help and guidance. By his help and guidance, I had a good beginning and progress in studying problems in bioinformatics.

A special thanks to Dr. Donald K. Friesen for his kindness to students. I enjoyed his CPSC 629 class which gave me a good beginning for my research.

A special thanks to Dr. Sergiy Butenko for his quick response every time I requested his help.

A special thanks to all my former teachers in China and the USA for their kindness help in my every stage of studying.

A special thanks to the National Science Foundation and the Department of Computer Science for their financial supports during my study at Texas A&M University.

A special thanks to Barry O'Sullivan, Igor Razgon, Fedor V. Fomin, Yngve Villanger for collaboration that led to excellent papers.

I would like to thank Yang Liu, Fenghui Zhang, Iyad A. Kanj, Xiuzhen Huang and Ge Xia for their helpful discussions and collaboration.

Finally, I would like to thank my father-in-law, mother-in-law, brother-in-law and sister-in-law who gave me help and encouragement, when I had hard time. I am grateful to my wife, who has been sharing happiness and hardship with me since we got married, and my son, who brings us happiness and gives me motivation to work

harder.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Traditionally, when a problem is proved to be NP-hard or NP-complete, it is believed that this problem cannot be solved efficiently, i.e. most people believe that the problem cannot be solved in polynomial time in terms of the input size of the instance of the problem. On the other hand, as in applications, such as in bioinformatics, networks and operating systems, many problems are NP-hard. There is no way to avoid them. Hence in effort to solve these practical problems, people study NP-hard problems from different directions.

One direction is exact algorithms, which use exponential time to find exact solutions with minimum or maximum sizes. One example is the MINIMUM FEEDBACK VERTEX SET problem, i.e. given a graph $G = (V, E)$, find a minimum size subset F of V such that after removing F from graph G, the remaining graph has no cycles. The MINIMUM FEEDBACK VERTEX SET problem can be solved by an algorithm of running time $O(1.9053^n)$ in a graph with n vertices [106]. Another direction is polynomial time approximation algorithms, which use polynomial time to find solutions with sizes that are close to values of exact solutions with minimum or maximum sizes. One example is using $O(n^2)$ time to find a feedback vertex set of size at most 2 times as large as the size of the minimum feedback vertex set in a graph with n vertices [7]. The third direction is parameterized algorithms. This dissertation will present new results about parameterized algorithms for parameterized NP-hard problems. The dissertation will also show how our new parameterized algorithms can be applied to problems in bioinformatics.

The journal model is Journal of Computer and System Sciences.

A. Parameterized NP-hard Problems and Motivation

First, let us define parameterized NP-hard problems. **Parameterized NP-hard problems** are NP-hard problems associated with special variables called parameters. An example of the problem is to find simple paths of length k in a graph, where the integer k is the parameter. This problem is called the P-PATH problem. The P-PATH problem is the parameterized version of the well-known NP-complete problem – the LONGEST SIMPLE PATH problem, i.e. given a graph $G = (V, E)$, find the longest simple path in G.

Following two examples can help you to further understand the concept of parameterized NP-hard problems.

> P-FEEDBACK VERTEX SET: Given an undirected graph $G = (V, E)$ and an integer k, either find a set F of k vertices in G whose removal results in an acyclic graph, or report that no such an F exists. We usually call F the FVS of the graph.

> P-NODE MULTIWAY CUT: Given an undirected graph $G = (V, E)$, a collection of pairwise disjoint terminal sets $\{T_1, \ldots, T_l\}$ (where each T_i is a subset of vertices in G), and a parameter k, either construct a separator of at most k vertices in G, or report that no such a separator exists. (Note: A separator is a vertex set that does not include any vertex from any terminal set and if we remove it from the graph, then no two vertices from different terminals are in the same connected component.)

The k in the P-FEEDBACK VERTEX SET problem and in the P-NODE MULTIWAY CUT problem is the parameter. In the P-FEEDBACK VERTEX SET problem, if instead of finding a set F of k vertices in G, we find a set F with minimum size whose removal

results in an acyclic graph, then the problem becomes the MINIMUM FEEDBACK VERTEX SET problem which is NP-hard. Similarly, the MINIMUM NODE MULTIWAY CUT problem can be obtained.

There are two main reasons why we study parameterized NP-hard problems. First, many application problems are naturally associated with certain parameters, i.e. they are parameterized NP-hard problems. Second, if the values of the parameters are small, we want to take advantage of these parameters to design very effective algorithms.

One example in computational biology is to find signaling pathways in protein interaction networks. This problem can be formulated as finding a set of short simple paths (6 to 12 proteins) with top weights and then combining these paths into a path structure. The corresponding algorithmic problem for the signaling pathway problem is the PW-PATH problem, i.e. to find a simple path of k vertices with maximum weight in a weighted graph. The formal definition of the PW-PATH problem is as following.

> PW-PATH: Given a weighted undirected graph G of n vertices and m edges and a parameter k, either construct a k-path in G whose weight is the maximum over all k-paths in G, or report that no k-path exists in G.

As the PW-PATH problem can be solved in time $O(4^k k^{2.7} m)$ by an algorithm [29], where m is the number of edges in the graph, the algorithm is very efficient for small k in the signaling pathway problem (note: the algorithm developed in [29] is for the P-PATH problem. By a minor modification, it can be used to solve the PW-PATH problem).

B. Fixed Parameter Tractable Problems

For many parameterized NP-hard problems, such as the P-FEEDBACK VERTEX SET problem and the P-NODE MULTIWAY CUT problem, their solutions are subsets of k elements in a ground set of n elements. If a subset of k elements is given, whether this subset is the solution can be verified in polynomial time. Therefore, if k is smaller than a fixed number, such as $k < 20$, then these problems are solvable in polynomial time by testing every subset of size k.

By this trivial method, the P-NODE MULTIWAY CUT can be solved in time $O(n^{k+2})$. Though this trivial algorithm for the P-NODE MULTIWAY CUT is simple and easy to be implemented, its high time complexity makes it impractical even for small k. The problem is that the time complexity increases too quickly when k increases, as in a usual given instance, n can be very large. Can the P-NODE MULTIWAY CUT problem be solved by a better algorithm? The current best algorithm for the P-NODE MULTIWAY CUT problem has a time complexity of $O(4^k kn^3)$ [26]. As usually, n is large and k is small, an algorithm with running time bounded by $O(4^k kn^3)$ is much better than an algorithm with running time bounded by $O(n^{k+2})$.

In general, if a parameterized NP-hard problem can be solved by an algorithm of running time in form of $f(k)n^{O(1)}$, where $f(k)$ is a function depending only on k and n is the input size of the problem instance, we say that the parameterized NP-hard problem is fixed-parameter tractable, simply called FPT.

The merit is obvious if a parameterized NP-hard problem is fixed-parameter tractable. Since the time complexity is in form of $f(k)n^{O(1)}$, where $f(k)$ depends only on parameter k and is independent of instance size n, if the parameter k is not very large, the problem can still be solved efficiently even when the size of the instance is very large.

There are a number of parameterized NP-hard problems from applications that can be solved by algorithms that have time complexity in form of $f(k)n^{O(1)}$. For example, the signalling pathway problem can be formulated as the PW-PATH problem which can be solved by an algorithm of running time $O(4^k k^{2.7} m)$. As for the signalling pathway problem, the parameter k takes only small values (less than 15, the input size is usually larger than $5,000$). Thus the problem can be solved very effectively.

A nature question is "Are all parameterized NP-hard problems fixed-parameter tractable?". It would be excellent if all parameterized NP-hard problems are fixed-parameter tractable, for fixed-parameter tractable problems can be solved by algorithms of running time in the form of $f(k)n^{O(1)}$, which is very effectively for small k. Unfortunately, there are strong evidences that this is not the case. For example, the following two problems are very famous problems that are believed to be not fixed-parameter tractable.

> P-INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer k, either find a subset I of size k in V such that no edge connects any two vertices in I, or report that no such an I exists. We call subset I the independent set.

> P-DOMINATING SET: Given a graph $G = (V, E)$ and an integer k, either find a subset D of size k in V such that for any $v \in V$, if $v \notin D$, then v has a neighbor in D, or report that no such a D exists. We call subset D the dominating set of the graph.

Just as many computer scientists believe that P $\neq$ NP, most people who study parameterized NP-hard problems believe that not all parameterized NP-hard problems are fixed-parameter tractable. It has been proved that the P-INDEPENDENT SET and P-DOMINATING SET problems cannot be solved by any algorithms of running time

in form of $O(n^{o(k)})$ unless all problems in SNP are solvable in subexponential time [23]. This is a very strong evidence that the P-INDEPENDENT SET and P-DOMINATING SET problems are not fixed-parameter tractable. Hence it is very possible that the best algorithms we can have for the P-INDEPENDENT SET and P-DOMINATING SET problems have the time complexities of $n^{O(k)}$ which are not much better than the trial algorithm that tests all subsets of size k and has the time complexity of $O(n^k)$.

There is a branch in complexity theory that focuses on parameterized NP-hard problems that are not fixed-parameter tractable. As this dissertation concentrates on designing algorithms for parameterized NP-hard problems that are fixed-parameter tractable, we mainly present new techniques and algorithms for parameterized NP-hard problems that are fixed-parameter tractable, while results about problems that are not fixed-parameter tractable are not the fucus of the dissertation.

C. Dissertation Outline

The dissertation is organized as follows. In Chapter II, several general techniques to design algorithms for problems that are fixed-parameter tractable will be introduced. These techniques include branch and bound, divide and conquer, color coding and dynamic programming, iterative compression, iterative expansion, and kernelization.

In Chapter III, branch and bound techniques are used to solve the P-NODE MULTIWAY CUT problem.

In Chapter IV, how to use divide and conquer techniques to solve the PW-PATH, PW-r-D MATCHING and PW-r-SET PACKING problems will be presented. How to use set partition technique to solve the PW-SET SPLITTING problem is also introduced.

In Chapter V, iterative expansion technique is combined with color coding and dynamic programming techniques to solve the P-3-D MATCHING problem.

In Chapter VI, iterative compression and branch techniques are used to solve the PD-FEEDBACK VERTEX SET, P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET problems on directed and undirected graphs. Especially, we give the first algorithm of running time in form of $f(k)n^{O(1)}$ for the PD-FEEDBACK VERTEX SET, thus solving a long-standing open problem.

In Chapter VII, the parameterized algorithms are used to solve the SIGNALLING PATHWAY and MOTIF FINDING problems in bioinformatics.

In Chapter VIII, the summary of the dissertation and the future work will be given.

CHAPTER II

GENERAL TECHNIQUES FOR PARAMETERIZED NP-HARD PROBLEMS

The most basic method to deal with NP-complete problems is branch and bound. This is also true for parameterized NP-hard problems. In addition to branch and bound techniques, there are many other techniques that are suitable to solve parameterized NP-hard problems. A partial list of these techniques includes divide and conquer, color coding and dynamic programming, iterative compression, iterative expansion, and kernelization. In this chapter, main principles of these techniques, that are suitable for parameterized NP-hard problems, in this partial list will be introduced. Hence you can acquire basic ideas of these techniques. In later chapters, these techniques will be applied to solve parameterized NP-hard problems, especially problems that are fixed parameter tractable, i.e. problems that can be solved by algorithms of running time in form of $f(k)n^{O(1)}$.

A. Branch and Bound

The branch and bound method is the most general method to deal with NP-complete problems. The basic idea of the branch and bound is very simple. Given an instance of a problem, we branch the instance into two or more simpler sub-instances that the solution of the original instance can be constructed from the solutions of all sub-instances. Then we branch each sub-instance again and repeat this process until we reach sub-instances that can be solved easily, i.e. that can be solved in constant time or polynomial time. Let us use the p-vertex cover problem as an example to explain this process.

P-VERTEX COVER: Given an graph $G = (V, E)$ and an integer k, either find a subset C, which is called the vertex cover, of size bounded by k in V such that every edge in E has at least one end in C, or report that no such a subset C exists. The instance of the P-VERTEX COVER problem is denoted as $I = \{(V, E), k\}$.

One simple way to solve this problem is that, for the given instance $I = \{(V, E), k\}$, we choose any edge $[u, v]$ in E. It is obvious that either vertex u or vertex v must be in the vertex cover C. Hence, the instance is branched into two cases. In one case, the vertex u is included into C and a new sub-instance $I_1 = \{(V - \{u\}, E - \{[u, v]\}), k - 1\}$ is obtained. In other case, we include the vertex v into C and continue to solve sub-instance $I_2 = \{(V - \{v\}, E - \{[u, v]\}), k - 1\}$. The same process is repeatedly applied to each sub-instance until $k = 0$ or no edge exists in the sub-instance. As for each branch, the parameter k will decrease by one, the parameter k will become 0 after the original instance is branched at most k times, where in this case, the instances can be solved in polynomial time.

How is the time complexity for this simple branch and bound algorithm calculated? Let $T(k)$ be the number of leaves in the branch tree for the instance $I = \{(V, E), k\}$ with parameter k. Then in each branch, $T(k) = 2T(k - 1)$, i.e. an instance with parameter k become two instances with parameter $k - 1$. It is easy to deduce that $T(k) \leq 2^k$ and the time complexity to solve the P-VERTEX COVER is bounded by $O^*(2^k)$ [1].

The most general branch for parameterized NP-hard problems has the relation $T(k) \leq \sum_{i=1}^{k} a_i T(k - i)$, where all a_i are non negative integers and not all a_i are zero.

[1] In this dissertation, $O^*(f(k))$ is used as a short for $f(k)n^{O(1)}$, where the polynomial part $n^{O(1)}$ is neglected.

Let $T(i) = t^i$ for all $0 \leq i \leq k$. Then we can obtain an equation $t^k - \sum_{i=1}^{k} a_i t^{k-i} = 0$. It can be proved that this equation has a unique positive root c and $T(k) \leq c^k$ [25]. Hence we can deduce from this equation that the time complexity is $O^*(c^k)$. By the way, a well-known algorithm that uses branch and bound for the P-VERTEX COVER has a time complexity of $O^*(1.2852^k)$ [25].

B. Divide and Conquer

The main idea of the divide and conquer method is to split an instance (x, k), where k is the parameter and x is the remaining part of the input of the instance, of parameterized NP-hard problem into two sub-instances $(x_1, k/2)$ and $(x_2, k/2)$. Then solve two sub-instances $(x_1, k/2)$ and $(x_2, k/2)$ independently. There are two important criteria when we use divide and conquer to deal with parameterized NP-hard problems. First, the time complexity to split an instance into two sub-instances must be bounded by $O^*(f(k))$, where $f(k)$ is a function depending only on k. Second, two sub-instances must be independent, i.e. the solution of the first sub-instance does not depend on the second sub-instance and the solution of the second sub-instance does not depend on the first sub-instance.

There are many parameterized NP-hard problems whose solutions are subsets of size k in a given set, such as the PW-PATH problem and the P-VERTEX COVER problem, where their solutions are subsets of k vertices in a given graph. Let the instance of the original problem have n vertices $\{v_1, v_2, \ldots, v_n\}$ and the unknown solution for the instance be $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. In $O(n)$ time, vertices in solution set S can be split into two groups of equal size, for there must be a number j between 1 and n such that half of vertices in the solution set S are in $\{v_1, v_2, \ldots, v_j\}$ and another half of vertices in the solution S are in $\{v_{j+1}, v_{j+2}, \ldots, v_n\}$. However, if the

solution set S should be split into two specific groups $S1$ and $S2$, i.e. certain elements must be in the same group, polynomial time is not enough. There is a simple random process that can split the solution set S into $S1$ and $S2$. If each vertex in the problem is randomly put into either group 1 or group 2 with equal probability 0.5, then with probability $1/2^k$, all elements in $S1$ are in group 1 and all elements in $S2$ are in group 2. Hence if this simple random process is repeated $12 \cdot 2^k$ times, the probability that S is split into $S1$ and $S2$ by at least one trial is greater than 0.99. There is an (n, k)-universal set [93] that can be used to derandomize this random process. This (n, k)-universal set will be discussed in detail in Chapter IV.

It seems that the divide and conquer method does not work for all parameterized problems. For example, for the p-vertex cover problem, after the instance is split into two sub-instances and the solution set S of size k is split into two subsets of size $k/2$ such that each sub-instance has one subset, then both sub-instances will have a vertex cover of size $k/2$. Howerver, when finding vertex cover of size $k/2$ in each sub-instance, we need to consider edges between two sub-instances, i.e. the solution of the first instance depends on the solution of the second instance if you want to combine these two solutions in two sub-instances into one solution for the original instance. Hence, the divide and conquer method is hard to be applied to the p-vertex cover problem.

For the pw-path problem, suppose $\langle u_{i_1}, \ldots, u_{i_{k/2}}, u_{i_{k/2+1}}, \ldots, u_{i_k} \rangle$ is a simple path of length k, where u_{i_j} is connected to $u_{i_{j+1}}$ for all $1 \leq j < k$. If, after the graph is partitioned into two sub-graphs G_1 and G_2, $u_{i_1}, \ldots, u_{i_{k/2}}$ are in sub-graph G_1 while $u_{i_{k/2+1}}, \ldots, u_{i_k}$ are in sub-graph G_2, then both sub-graphs will have a simple path of length $k/2$. As any simple path of length $k/2$ that ends with vertex $u_{i_{k/2}}$ in G_1 and any simple path of length $k/2$ that ends with vertex $u_{i_{k/2+1}}$ in G_2 can construct a simple path of lenght k in the orginal graph, we can find simple paths of length $k/2$

in two sub-graphs independently. Therefore we can use divide and conquer method to solve the PW-PATH problem. Chapter IV will discuss how divide and conquer method is used to solve the PW-PATH, PW-r-D MATCHING and PW-r-SET PACKING problems.

C. Color Coding and Dynamic Programming

Color coding and dynamic programming techniques were first introduced to solve the parameterized NP-hard problem by Alon et al. [3]. In their paper [3], they applied color coding and dynamic programming techniques to the P-PATH problem (unweighted version of the PW-PATH problem) and improved the time complexity for the P-PATH problem from $O^*(k^k)$ to $O^*(c^k)$ for a constance c.

The basic idea for color coding and dynamic programming is that first, all vertices in the problem are colored with a number of colors, where each vertex is assigned with one color and the total number of colors used is bounded by a function $c(k)$ depending only on k. Then, the dynamic programming is applied to find the solution. In the coloring step, the goal is to color all vertices in the (unknown) solution set with different colors, i.e. no two vertices in the solution set are colored with the same color. The time to reach this goal should be bounded by $O^*(g(k))$, where $g(k)$ is a function depending only on k. In the dynamic programming step, only different combinations of color sets, not different combinations of vertex sets, need to be remembered. As the number of colors is bounded by $c(k)$, the total number of different combinations of color sets is bounded by $2^{c(k)}$. This $2^{c(k)}$ will dominate the time complexity in the dynamic programming step.

A k-coloring function is coloring n elements in the problem instance with k colors. If no two elements in a subset are colored with the same color, we say that this subset is colored properly. A k-color coding scheme for set X is a collection of k-coloring

functions such that every subset with k elements in X is colored properly by at least one k-coloring function in the scheme. A k-color coding scheme of size $O^*(6.4^k)$ [29] will be introduced in Chapter V. In the coloring step, all k-coloring functions in the k-color coding scheme are enumerated. The unknown solution set of size k will be colored properly by at least one k-coloring function.

In the coloring step, the k-color coding scheme can be replaced by a random process. Let every element in the ground set be randomly assigned color i with probability $1/k$, where $1 \leq i \leq k$. Then in each trial, the probability that a special subset of size k is colored properly is $k!/k^k$. Therefore, if we do $O(12 \cdot k^k/k!)$ trials, the probability that this special subset is colored properly by at least one trial is larger than 0.99.

For any problem that the solution is a subset of size k in a ground set of size n, the solution set can be colored properly by a k-color coding scheme or a random coloring process. However the solution seems not necessary to be found in the dynamic programming step by only remembering color sets, such as for the P-VERTEX COVER problem, even the solution subset is color properly, the solution is still hard be found in the dynamic programming step. The difficulty is that two subsets with the same color but with different vertices will cover different edges. Thus it is not enough to remember only color sets. For the P-PATH problem, if two subsets with the same color set form two sub-paths that share the same end, when a new vertex, that connects to the common end and has a new color, is added to the color sets, the length of either of this two sub-paths will increase by one. remembering either path will not effect the path extension in later step. Hence for each vertex in the graph, we only need to remember different combinations of color sets that are corresponding to sub-paths using this vertex as one end. How to use color coding and dynamic programming to solve the P-3-D MATCHING problem will be discussed in Chapter V.

D. Iterative Compression

The iterative compression technique was proposed by Reed et al. [107]. The technique is suitable for parameterized NP-hard problems whose corresponding NP-hard optimization problems are to find the solution of minimum value, such as the P-FEEDBACK VERTEX problem. The basic idea is that instead of finding a solution of size k directly, which is very hard, we solve a serial of problems that find a solution of size k under the condition that a solution of size $k+1$ is given.

Let us use the P-FEEDBACK VERTEX problem as an example to explain this technique in detail. Given an instance of the P-FEEDBACK VERTEX problem, first a set of vertices from the graph are removed so that the remaining graph has an FVS of size bounded by k. The extreme case is removing $n-k$ vertices from the graph, where n is the number of vertices in the graph. It is obvious that the remaining graph with only k vertices has an FVS of size bounded by k. Then one by one, all removed vertices will be added back. Each time, when one vertex is added back, the newly added vertex and the FVS of size k for the previous instance form an FVS of size $k+1$. If an FVS of size k is found in this new instance, we continue to add another vertex back, and so on until all vertices are added back. Thus, an FVS of size k in the original graph can be found. If in one step, an FVS of size k does not exist, then the original graph does not have an FVS of size k. It is obvious that in each step, we only need to find an FVS of size k under the condition that an FVS of size $k+1$ is given. This iterative compression technique is combined with our other techniques to solve the PD-FEEDBACK VERTEX, P-FEEDBACK VERTEX and PW-FEEDBACK VERTEX problems in Chapter VI.

E. Iterative Expansion

The *iterative expansion* is the opposite of the iterative compression. In iterative expansion, we try to solve a serial of problems that find a solution of size $k+1$ under the condition that a solution of size k is given.

The iterative expansion is a general technique that can be applied to any parameterized NP-hard problems whose corresponding NP-hard optimization problems are to find the solution of maximum value. However you only want to use this technique if you can take advantage of a solution of size k to find a solution of size $k+1$. In this case, the problem can be solved efficiently by beginning from a solution of size 1, then finding a solution of size 2, further finding a solution of size 3, and so on until finding a solution of required size.

In Chapter V, iterative expansion technique is combined with color coding and dynamic programming techniques to solve the P-3-D MATCHING problem.

F. Kernelization

In studying parameterized NP-hard problems, the kernelization is a process that uses polynomial time (in term of the input size of the problem instance) to reduce an instance (x, k), where k is the parameter and x is the remaining part of the input of the instance, of problem $\prod$ to another instance (x', k') of problem $\prod$ such that

(1) (x, k) is a "yes" instance if and only if (x', k') is a "yes" instance.

(2) $k' \leq k$.

(3) $|x'| < g(k)$, where $g(k)$ is a function depending only on k.

One advantage for kernelization is that if the parameterized NP-hard problem is kernelizable, we can use polynomial time to find the kernel and then solve the new

instance, the kernel, with a very small input size. For example, the p-vertex cover problem has a kernel which is a graph with at most $2k$ vertices. Hence instead of finding a vertex cover of size k in a huge graph directly, we first use polynomial time to find a kernel that has at most $2k$ vertices. Then we find a vertex cover of size $k' \leq k$ in the kernel. This means the hard part in computation is only on the kernel. When k is small, the kernel is small. Hence, the problem can be solved efficiently.

If a problem is kernelizable, the input size of the kernel is bounded by a function $g(k)$ depending only on k. Usually, a parameterized NP-hard problem whose input size is bounded by a function depending only on k can be solved trivially by an algorithm that the time complexity is bounded by a function $h(k)$ depending only on k. This shows directly that the problem is fixed parameter tractable if the problem is kernelizable. In fact, it can be proved that a problem is kernelizable if and only if the problem is fixed parameter tractable. Examples of kernelization can be found in papers [37, 50, 87]. In the dissertation, Chapter IV will introduce how to find the kernel for the p-set splitting problem.

CHAPTER III

BRANCH-AND-BOUND*

The branch and bound method is the most general method to deal with NP-complete problems. The basic idea of the branch and bound method is very simple. Given an instance of a problem, we branch the instance into two or more simpler sub-instances that the solution of the original instance can be constructed from solutions of all sub-instances. Then we branch each sub-instance again and repeat this process until we reach sub-instances that can be solved easily. In this chapter, we will present in detail how the branch and bound method is used to solve the p-node multiway cut problem.

A. An Introduction to the Parameterized Node Multiway Cut Problem

The minimum cut problem is a well-known problem, and has been extensively studied ([18, 69, 94]). Applications of this problem are found in distributed computing [115], VLSI [30], computer vision [16], and many other fields. The problem is defined as follows: given an undirected graph $G = (V, E)$ and a set of l vertices $\{t_1, \ldots, t_l\}$ in G (the vertices t_i are called *terminals*), find an edge set E' of minimum size in G such that after the deletion of E', no two terminals are in the same connected component. This problem is NP-hard for general graphs for any fixed integer $l \geq 3$, and is also NP-hard for planar graphs when l is not fixed [33].

A generalization of the minimum cut problem is the minimum node multiway cut problem, which, for a given graph and a given set of terminals, is to find a vertex

*Reprinted with permission from "An Improved Parameterized Algorithm for the Minimum Node Multiway Cut Problem" by Jianer Chen, Yang Liu, Songjian Lu, 2009. *Algorithmica*, 55, 1-13, Copyright 2009 by Springer.

set S of minimum size such that after the deletion of S, no two terminals are in the same connected component. The MINIMUM NODE MULTIWAY CUT problem is at least as hard as the MINIMUM CUT problem, as the latter can be reduced to the former in time $O(|V| + |E|)$, if we require that no terminal be in the separator S [32]. Therefore, the MINIMUM NODE MULTIWAY CUT problem is also NP-hard if the number l of terminals is at least 3.

When there are only two terminals s and t, the MINIMUM CUT problem and the MINIMUM NODE MULTIWAY CUT problem become the edge version and the vertex version of the MINIMUM s-t CUT problem, respectively. According to the max-flow min-cut theorem [52], the MINIMUM s-t CUT problem, for both the edge version and the vertex version, can be solved via algorithms for the MAXIMUM s-t FLOW problem. For an undirected graph G of n vertices and m edges, the MAXIMUM s-t FLOW problem can be solved in time $O(n^{7/6}m^{2/3})$ [70]. In consequence, when there are only two terminals, the MINIMUM CUT problem and the MINIMUM NODE MULTIWAY CUT problem can also be solved in time $O(n^{7/6}m^{2/3})$.

A natural extension of the MINIMUM NODE MULTIWAY CUT problem is to have a collection of terminal sets, instead of a collection of individual terminals. Formally, let $G = (V, E)$ be an undirected graph, and let $\{T_1, \ldots, T_l\}$ be a collection of *terminal sets* where each T_i is a subset of vertices in G. A *separator* S for $\{T_1, T_2, \ldots, T_l\}$ is a subset of vertices in G such that no vertex in S is in any terminal set, and after deleting S from the graph G, no connected component in the resulting graph contains vertices from more than one terminal set.

In certain real world applications, one may expect that the size of the separator be small. For example, suppose that we are given a network (i.e., a graph) $G = (V, E)$ and a collection of network node groups $\{T_1, \ldots, T_l\}$ in G, and we want to monitor the message communication among the node groups. A separator for $\{T_1, \ldots, T_l\}$ in the

network G will well serve for this purpose: any communication path between any two node groups must pass through at least one node in the separator. Therefore, if we set up a monitor process in each of the nodes in the separator, then we can monitor all communications among the node groups. Naturally, we may want to limit the cost of this monitoring system by using only a small number of "monitor nodes" in the network G.

This motivates a parameterized version of the MINIMUM NODE MULTIWAY CUT problem, which will be called the P-NODE MULTIWAY CUT, i.e. given an undirected graph $G = (V, E)$, a collection of pairwise disjoint terminal sets $\{T_1, \ldots, T_l\}$ (where each T_i is a subset of vertices in G), and a parameter k, either construct a separator of at most k vertices in G, or report that no such a separator exists. Our goal is, for the P-NODE MULTIWAY CUT problem, to develop a parameterized algorithm whose running time is of the form $f(k)n^{O(1)}$ with a function f independent of the input size n.

It can be derived from the graph minor theory of Robertson and Seymour [44] that there is a parameterized algorithm whose running time is of the form $f(k)n^{O(1)}$ for the P-NODE MULTIWAY CUT problem. However, the proof is not constructive. An explicit constructive algorithm for the problem was given by Marx [90], who developed an algorithm of running time $O(4^{k^3}n^5)$ for the P-NODE MULTIWAY CUT problem for its original version (i.e., in which each terminal set is restricted to contain a single terminal). To our knowledge, this is the only known constructive parameterized algorithm whose running time is of the form $f(k)n^{O(1)}$ for the problem.

In this chapter, we present an algorithm of running time $O(k4^k n^3)$ for the P-NODE MULTIWAY CUT problem, which significantly improves the algorithm given in [90]. In the real world of computing, this improvement makes it become possible to practically solve the problem for some reasonable values of the parameter k. For

example, for the case of $k = 10$, our algorithm has running time $O(4^{10}n^3)$, which is practically feasible using the currently available computation power. On the other hand, the algorithm in [90] in this case has running time $O(4^{1000}n^5)$, which is totally infeasible from the practical point of view. Theoretically, our result gives the first polynomial time algorithm for the MINIMUM NODE MULTIWAY CUT problem when the size of the optimal separator is of order $O(\log n)$.

We only give an algorithm of finding a separator that has no vertices in any terminal set. We call such a separator a *restricted separator* (simply call it a *separator* in case of non confusion). If a separator is allowed to include vertices from terminal sets, the separator is called an *unrestricted separator*. It can be verified easily that the instance $(G, \{T_1, \ldots, T_l\}, k)$ has an unrestricted separator of size k if and only if the instance $(G', \{\{x_1\}, \ldots, \{x_l\}\}, k)$ has a restricted separator of size k, where the graph G' is obtained from the graph G by adding l new vertices $x_1, \ldots, x_l$ and connecting x_i to each vertex in T_i for all $1 \leq i \leq l$. Therefore, our algorithm can also be used to construct unrestricted separators for undirected graphs.

Finally, we remark that the basic idea for techniques we developed for the P-NODE MULTIWAY CUT problem is branch and bound. The techniques we developed seem to be very powerful for solving various kinds of multiway cut problems. In particular, very recently the techniques have been extended to directed graphs, and led to a parameterized algorithm of running time in form of $f(k)n^{O(1)}$ for the PD-FEEDBACK VERTEX SET problem [27], thus resolving an outstanding open problem in the area of parameterized computation and complexity [44, 43].

Before our main algorithm is presented, we first introduce some terminology and lemmas that are needed in the main algorithm.

B. On Minimum V-cuts between Two Terminal Sets

We start with some terminology. All graphs in our discussion are supposed to be undirected.

Let $G = (V, E)$ be a graph and let u and v be two vertices in G. A *path between u and v* is a simple path in G whose two ends are u and v, respectively. We say that there is a *path between a vertex u and a vertex subset V'* if there is a path between the vertex u and a vertex v in the subset V'. For two vertex subsets V_1 and V_2, we say that there is a *path between V_1 and V_2* if there exist a vertex u in V_1 and a vertex v in V_2 such that there is a path between u and v. Two paths are *internally disjoint* if there is no vertex that is an internal vertex for both the paths.

Let G be a graph, and let $\{T_1, \ldots, T_l\}$ be a collection of pairwise disjoint terminal sets (each terminal set is a subset of vertices in G). A subset S of vertices in G is a *separator* for $\{T_1, \ldots, T_l\}$ if S contains no vertex in any of the sets T_1, ..., T_l, and if after deleting all vertices in S from G, there is no path between any two different subsets T_i and T_j in the resulting graph. In particular, a separator S for two terminal sets T_1 and T_2 is also called a *V-cut* between the two sets T_1 and T_2.

Let T be a subset of vertices in the graph $G = (V, E)$. By *merging T (into a single vertex)*, we mean the operation that first deletes all vertices in T then creates a new vertex w adjacent to each v of the vertices in $V - T$ where v is a neighbor of a vertex in T in the original graph G.

Finally, for a subset V' of vertices in the graph G, we will denote by $G[V']$ the subgraph of G that is induced by the vertex subset V'. Without any ambiguity, we will denote by $G - V'$ the induced subgraph $G[V - V']$, and by $G - w$, where w is a vertex in G, the induced subgraph $G[V - \{w\}]$.

Proposition B.1 [20] (Menger's Theorem–Vertex Version) *Let u and v be two dis-*

tinct and nonadjacent vertices in a graph G. Then the maximum number of internally disjoint paths between u and v in G is equal to the size of a minimum V-cut between u and v in G.

Proposition B.1 can be generalized from the case for two vertices to the case of two vertex subsets, as given in the following lemma.

Lemma B.2 *Let T_1 and T_2 be two disjoint vertex subsets in a graph G such that no vertex in T_1 is adjacent to a vertex in T_2. Then the maximum number h of internally disjoint paths between T_1 and T_2 in G is equal to the size of a minimum V-cut between T_1 and T_2 in G. Moreover, for any set π of h internally disjoint paths between T_1 and T_2 in G, every minimum V-cut between T_1 and T_2 in G contains exact one vertex in each of the paths in π.*

PROOF. Let G' be the graph obtained from the graph G by merging the two vertex subsets T_1 and T_2 into two vertices t_1 and t_2, respectively. Note that t_1 and t_2 are not adjacent in G'.

By the definition of the merge operation, it is easy to verify that a vertex subset S is a V-cut between the vertex subsets T_1 and T_2 in the graph G if and only if S is a V-cut between the vertices t_1 and t_2 in the graph G'. In particular, the size of a minimum V-cut between T_1 and T_2 in G is equal to the size of a minimum V-cut between t_1 and t_2 in G'. Moreover, it is also easy to verify that for any integer h', from a set of h' internally disjoint paths between T_1 and T_2 in G, we can construct a set of h' internally disjoint paths between t_1 and t_2 in G', and *vice versa*. Therefore, the maximum number of internally disjoint paths between T_1 and T_2 in G is equal to the maximum number of internally disjoint paths between t_1 and t_2 in G'. Now the first part of the lemma follows by applying Proposition B.1 to the graph G'.

To prove the second part of the lemma, let S be a minimum V-cut, of size h, between T_1 and T_2 in G, and let π be a set of h internally disjoint paths between T_1 and T_2. The vertex set S must contain at least one vertex from each of the paths in π: otherwise there would be a path between T_1 and T_2 in $G - S$, contradicting the assumption that S is a V-cut between T_1 and T_2. Moreover, the set S cannot contain more than one vertex in any path in π: otherwise S would not be able to contain at least one vertex for each of the paths in π (note that the paths in π are internally disjoint). □

Lemma B.2 provides an efficient algorithm that constructs the maximum number of internally disjoint paths and a minimum V-cut between two given vertex subsets in a graph.

Lemma B.3 *Let T_1 and T_2 be two disjoint vertex subsets in a graph $G = (V, E)$ such that no vertex in T_1 is adjacent to a vertex in T_2. Then in time $O((|V| + |E|)k)$, we can decide if the size h of a minimum V-cut between T_1 and T_2 is bounded by k, and in case $h \leq k$, construct h internally disjoint paths between T_1 and T_2.*

PROOF. First we merge T_1 into t_1, T_2 into t_2 and transform the undirected graph into a directed graph by replacing each edge by two reverse arcs. Then we modify the new directed graph by replacing each vertex u (except the vertices t_1 and t_2) by two vertices u_1 and u_2 with an arc from u_1 to u_2, connecting all u's incoming arcs to the vertex u_1 and connecting all u's outgoing arcs to the vertex u_2. Finally we set all edges to have capacity 1 and apply the Ford-Fulkerson algorithm k times. □

C. The Main Algorithm

Now we return back to the P-NODE MULTIWAY CUT problem. Formally, an instance $(G, \{T_1, \ldots, T_l\}, k)$ of the P-NODE MULTIWAY CUT problem consists of an undirected graph G, a collection $\{T_1, \ldots, T_l\}$ of pairwise disjoint *terminal sets* (each terminal set is a vertex subset in G), and a parameter k. The objective is to either construct a separator of at most k vertices for $\{T_1, \ldots, T_l\}$, or conclude that no such a separator exists.

Before we formally present our algorithm, we give a less formal but intuitive explanation on the basic idea of the algorithm. Let the size of a minimum V-cut between T_1 and $\bigcup_{j \neq 1} T_j$ be m.

Pick a vertex u that is not in any terminal set and has a neighbor in T_1. If u also has a neighbor in another terminal set T_i, $i \neq 1$, then we can directly include u in the separator (this is necessary because the separator must separate T_1 and T_i), and recursively find a separator of size $k-1$ in the remaining graph. On the other hand, if u has no neighbor in other terminal sets, then we compute the size m' of a minimum V-cut between the sets $T_1' = T_1 \cup \{u\}$ and $\bigcup_{i \neq 1} T_i$. It can be proved that we must have $m \leq m'$. Note that by Lemma B.3, the values m and m' can be computed in polynomial time.

In the case $m = m'$, we will show that the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ has a separator of size bounded by k if and only if the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$ has a separator of size bounded by k. Then we recursively work on the new instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$. Thus, in the case of $m = m'$, we can reduce the number of vertices that are not in the separator by 1.

On the other hand, suppose $m < m'$. Then we branch on the vertex u in two cases, one includes u in the separator and the other excludes u from the separator.

In the case of including the vertex u in the separator, we recursively work on the instance $(G - \{u\}, \{T_1, T_2, \ldots, T_l\}, k-1)$, in which the parameter value is decreased by 1; and in the case of excluding the vertex u from the separator, we recursively work on the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$, in which the size of the minimum V-cut between T_1' and $\bigcup_{i \neq 1} T_i$ is increased by at least 1.

Therefore, for the given instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$, we can either (1) apply a polynomial time process that either decreases the parameter value by 1 or reduces the number of vertices not in the separator by 1, or (2) branch into two cases, of which one decreases the parameter value by 1 and the other increases the value m by at least 1 (see the definition of m given in the second paragraph in this section). Note that all these generated new instances will be "simpler" than the original given instance: (i) reducing the number of vertices not in the separator will narrow down our search space for the separator; (ii) an instance of parameter value bounded by 1 can be solved in polynomial time; and (iii) an instance in which the value m is larger than the parameter value k obviously has no separator of size bounded by k.

To present our formal discussions, we fix an instance $(G, \{T_1, \ldots, T_l\}, k)$ of the p-node multiway cut problem, where $G = (V, E)$ is a graph, and $\{T_1, \ldots, T_l\}$ is a collection of terminal sets in G. Let the size of a minimum V-cut between T_1 and $\bigcup_{j \neq 1} T_j$ be m. Moreover, fix a vertex u that is not in any of the terminal sets but has a neighbor in the terminal set T_1. Let $T_1' = T_1 \cup \{u\}$.

Lemma C.1 *Let m be the size of a minimum V-cut between the two sets T_1 and $\bigcup_{j \neq 1} T_j$, and let m' be the size of a minimum V-cut between the two sets T_1' and $\bigcup_{j \neq 1} T_j$. Then $m' \geq m$.*

Proof. The lemma follows from the observation that every V-cut between the sets

T_1' and $\bigcup_{j\neq 1} T_j$ is also a V-cut between the sets T_1 and $\bigcup_{j\neq 1} T_j$. □

The following theorem is the most crucial observation for our algorithm.

Theorem C.2 *If the minimum V-cuts between the sets T_1 and $\bigcup_{j\neq 1} T_j$ and the minimum V-cuts between the sets T_1' and $\bigcup_{j\neq 1} T_j$ have the same size, then the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ has a separator of size bounded by k if and only if the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$ has a separator of size bounded by k.*

PROOF. If the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$ has a separator S of size bounded by k, then it is obvious that S is also a separator for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$. In consequence, the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ also has a separator of size bounded by k.

Now we consider the other direction. Suppose that the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ has a separator S_k of size bounded by k.

To simplify the discussion, denote by T_{other} the set $\bigcup_{j\neq 1} T_j$. Let S_m be a minimum V-cut between T_1' and T_{other} (note that S_m does not contain u). Then S_m is also a V-cut between T_1 and T_{other}. In fact, by the assumption of the theorem, S_m is also a minimum V-cut between T_1 and T_{other}. Let $C(T_1)$ be the set of vertices x such that either $x \in T_1$ or there is a path between x and T_1 in the subgraph $G - S_m$. In particular, since u is not in S_m and u is adjacent to T_1, we have $u \in C(T_1)$. Moreover, let $C(T_{other}) = V - C(T_1) - S_m$.

By Lemma B.2, there exist $|S_m|$ internally disjoint paths between T_1 and T_{other}, each contains exactly one vertex in the set S_m. Therefore, each of these $|S_m|$ paths is cut into two subpaths by a vertex in S_m, such that one subpath is in the induced subgraph $G[C(T_1)]$ and the another subpath is in the induced subgraph $G[C(T_{other})]$. From this, we derive that there are $|S_m|$ internally disjoint paths between T_1 and S_m

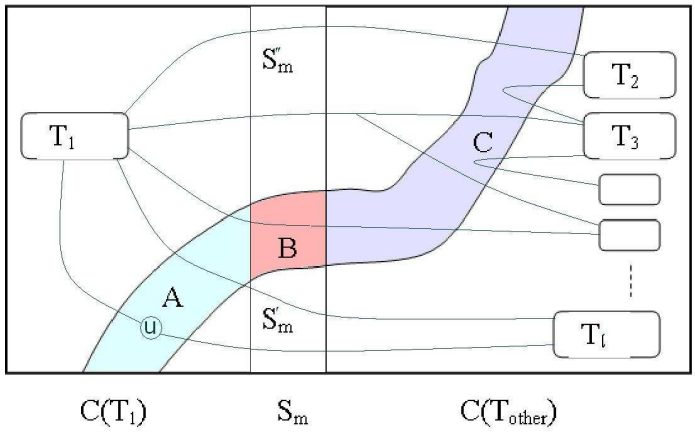


Fig. 1. Decomposition of separators

in the induced subgraph $G[C(T_1] \cup S_m)$, each contains a distinct vertex in the set S_m.

Define $A = S_k \cap C(T_1)$, $B = S_k \cap S_m$, and $C = S_k \cap C(T_{other})$. Finally, let S'_m be the set of vertices x in S_m such that there is a path between x and T_{other} in the induced subgraph $G[C(T_{other}) \cup S_m - S_k]$ (see Figure 1 for an intuitive illustration of these sets).

We first prove that $|A| \geq |S'_m|$.

From the fact that there are $|S_m|$ internally disjoint paths between T_1 and S_m in the induced subgraph $G[C(T_1) \cup S_m]$ in which each path contains a distinct vertex in the set S_m, we derive that there are $|S'_m|$ internally disjoint paths between T_1 and S'_m in the induced subgraph $G[C(T_1) \cup S'_m]$. If $|A| < |S'_m|$, then there must be a path P_1 between T_1 and a vertex v' in S'_m in the subgraph $G[C(T_1) \cup S'_m - A] = G[C(T_1) \cup S'_m - S_k]$. Moreover, by the definition of the set S'_m, there is also a path P_2 between v' and T_{other} in the induced subgraph $G[C(T_{other}) \cup S_m - S_k]$. The concatenation of the paths P_1 and P_2 would give a path between T_1 and T_{other} in the induced subgraph $G[V - S_k]$, which contradicts the assumption that S_k is a separator of the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$. Therefore, we must have $|A| \geq |S'_m|$.

Define a set $S'_k = S'_m \cup B \cup C$. We now prove that the set S'_k is a separator of the instance $(G, \{T'_1, T_2, \ldots, T_l\}, k)$. Suppose that the set S'_k is not a separator of the instance $(G, \{T'_1, T_2, \ldots, T_l\}, k)$, then there are two vertices v_1 and v_2 that are in two different terminal sets in $\{T'_1, T_2, \ldots, T_l\}$ and there exists a path P between v_1 and v_2 in the induced subgraph $G[V - S'_k]$. We discuss this in two possible cases.

Case 1: There is a vertex w in the path P such that $w \in C(T_1)$. Because (1) at least one of the vertices v_1 and v_2 is in the set T_{other}, (2) there is a path between T_1 and w in the induced subgraph $G[C(T_1)]$, and (3) S_m is a V-cut between T_1 and T_{other}, we conclude that there must be a vertex $s \in S_m$ that is also on the path P. Without loss of generality, we can suppose that the vertex v_1 is in the set T_{other}, and that the subpath P' of P that begins from v_1 and ends at s has no vertices from $C(T_1)$ – for this we only have to pick the first vertex s in S_m when we traverse on the path P from v_1 to v_2. Then the path P' is in the induced subgraph $G[C(T_{other}) \cup S_m - S'_k]$, which is a subgraph of the induced subgraph $G[C(T_{other}) \cup S_m - S_k]$. Now by the definition of the set S'_m, the vertex s is in the set S'_m, thus in the set S'_k. But this is impossible because we assumed that the path P is in the induced subgraph $G[V - S'_k]$.

Case 2: All vertices of the path P come from the induced subgraph $G[V - S'_k - C(T_1)]$. Then neither of the vertices v_1 and v_2 can be from the set T_1. Moreover, since $G[V - S'_k - C(T_1)]$ is a subgraph of the induced subgraph $G[V - S_k]$, the path P, which is between two different terminal sets in $\{T_2, \ldots, T_l\}$, would contain no vertex in S_k. But this again contradicts the assumption that S_k is a separator of the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$.

Combining the discussions in Case 1 and Case 2, we conclude that the set S'_k is a separator for the instance $(G, \{T'_1, T_2, \ldots, T_l\}, k)$.

Since $|A| \geq |S'_m|$, $S_k = A \cup B \cup C$, and $S'_k = S'_m \cup B \cup C$, and A does not intersect $B \cup C$, we conclude that $|S_k| \geq |S'_k|$. In particular, if the instance

$(G, \{T_1, T_2, \ldots, T_l\}, k)$ has the separator S_k of size bounded by k, then the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$ has the separator S_k' of size also bounded by k.

This completes the proof of the theorem. □

The proof of Theorem C.2 becomes complicated partially because the vertex u may be included in a separator for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$. If we restrict that the vertex u is not in the separators for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$, then a result similar to Theorem C.2 can be obtained much more easily, even without the need of the condition that the minimum V-cuts between T_1 and $\bigcup_{j \neq 1} T_j$ and the minimum V-cuts between T_1' and $\bigcup_{j \neq 1} T_j$ have the same size. This is given in the following lemma. This result will also be needed in our algorithm.

Lemma C.3 *Let S be a vertex subset in the graph G such that S does not include the vertex u. Then S is a separator for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ if and only if S is a separator for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$.*

PROOF. If S is a separator for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$, then as explained in Theorem C.2, S is also a separator for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$.

For the other direction, suppose that S is a separator for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$. We show that S is also a separator for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$. Suppose that S is not a separator for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$. Then there is a path P in $G - S$ between two different terminal sets in $\{T_1', T_2, \ldots, T_l\}$. Let one of these two terminal sets in $\{T_1', T_2, \ldots, T_l\}$ be T_i, where $i \neq 1$. The path P must contain the vertex u (recall that S does not contain u) – otherwise the path P in $G - S$ would be between two different terminal sets in $\{T_1, T_2, \ldots, T_l\}$, contradicting the assumption that S is a separator for $(G, \{T_1, T_2, \ldots, T_l\}, k)$. However, this would imply that the path from T_1 to u (recall that u has a neighbor in T_1) then following

the path P to the terminal set T_i would give a path in $G - S$ between T_1 and T_i, again contradicting the assumption that S is a separator for $(G, \{T_1, T_2, \ldots, T_l\}, k)$. This contradiction shows that the set S must be also a separator for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$. □

Now, we are ready to present our algorithm. For an instance $(G, \{T_1, \ldots, T_l\}, k)$ of the P-NODE MULTIWAY CUT problem, a vertex in the graph G that does not belong to any terminal sets will be called a "non-terminal".

The algorithm is given in Figure 2.

Algorithm NMC$(G, \{T_1, T_2, \ldots, T_l\}, k)$
input: an instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ of the P-NODE MULTIWAY CUT problem $(l \geq 2)$
output: a separator of size bounded by k for $(G, \{T_1, T_2, \ldots, T_l\}, k)$, or report "No" (i.e., no such a separator)

1. **if** an edge has its two ends in two different terminal sets **then return** "No";
2. **if** a non-terminal w has two neighbors in two different terminal sets **then return** $\{w\} \cup$ **NMC**$(G - w, \{T_1, \ldots, T_l\}, k-1)$; ‡
3. find the size m_1 of a minimum V-cut between T_1 and $\bigcup_{j=2}^{l} T_j$;
4. **if** $m_1 > k$ **then return** "No";
5. **if** ($m_1 = 0$ and $l = 2$) **then return** $\emptyset$;
5.1. **if** ($m_1 = 0$ and $l > 2$) **then return NMC**$(G, \{T_2, \ldots, T_l\}, k)$;
6. **else** pick a non-terminal u that has a neighbor in T_1; let $T_1' = T_1 \cup \{u\}$;
6.1. **if** the size of a minimum V-cut between T_1' and $\bigcup_{j=2}^{l} T_j$ is equal to m_1 **then return NMC**$(G, \{T_1', T_2, \ldots, T_l\}, k)$;
6.2. **else** $S = \{u\} \cup$ **NMC**$(G - u, \{T_1, T_2, \ldots, T_l\}, k-1)$;
if S is not "No" **then return** S;
6.3. **else return NMC**$(G, \{T_1', T_2, \ldots, T_l\}, k)$.

‡To simplify the expression, we suppose that "No" union any vertex set gives a "No".

Fig. 2. An algorithm for the P-NODE MULTIWAY CUT problem

Theorem C.4 *The algorithm* **NMC**$(G, \{T_1, T_2, \ldots, T_l\}, k)$ *solves the* P-NODE MULTIWAY CUT *problem in time* $O(n^3 k 4^k)$.

PROOF. We first prove the correctness of the algorithm. Let $(G, \{T_1, T_2, \ldots, T_l\}, k)$ be an input to the algorithm, which is an instance of the P-NODE MULTIWAY CUT problem, where $G = (V, E)$ is a graph, $\{T_1, T_2, \ldots, T_l\}$ is a collection of terminal sets, and k is the upper bound of the size of the separator we are looking for.

If there is an edge whose two ends are in two different terminal sets, then we have no way to separate these two terminal sets since all vertices in a separator are supposed to be non-terminals. Step 1 handles this case correctly.

If a non-terminal w has two neighbors that are in two different terminal sets, then w must be in the separator because otherwise the two terminal sets will not be separated. Thus, we can simply include the vertex w in the separator, and recursively find a separator of size bounded by $k - 1$ for the same collection of terminal sets $\{T_1, T_2, \ldots, T_l\}$ in the remaining graph $G - w$. This case is correctly handled by step 2.

Step 3 computes the size m_1 of a minimum V-cut between the sets T_1 and $\bigcup_{j=2}^{l} T_j$. By Lemma B.3, the value m_1 can be computed in time $O((|V| + |E|)k)$.

If $m_1 > k$, then the size of a minimum V-cut between T_1 and $\bigcup_{j=2}^{l} T_j$ is larger than k, which means that even separating the set T_1 from the other sets $\bigcup_{j=2}^{l} T_j$ requires more than k vertices. Thus, no separator of size bounded by k can exist for the terminal sets T_1, T_2, ..., T_l. This is handled by step 4.

In step 5 we handle the case $m_1 = 0$ and $l = 2$, which we do not need to remove any vertex to separate T_1 and T_2, i.e. the problem is solved. So we return an empty set $\emptyset$ as a separator of size 0 (note that because of step 4, here we must have $k \geq 0$). In step 5.1, $m_1 = 0$ and $l > 2$, which means that T_1 is already separated from $T_2, \ldots, T_l$. Hence we only need to find a separator to separate $T_2, \ldots, T_l$. Therefore, step 5.1 handles this case correctly.

When the algorithm reaches step 6, the following conditions hold true: (1) no edge has its two ends in two different terminal sets (because of step 1); (2) no non-terminal has two neighbors in two different terminal sets (because of step 2); (3) $0 < m_1 \leq k$ (because of steps 4-5). In particular, by condition (3), there must be a non-terminal u that has a neighbor in T_1.

Let $T_1' = T_1 \cup \{u\}$ and let m' be the size of a minimum V-cut between the sets T_1' and $\bigcup_{j=2}^{l} T_j$. If $m' = m_1$, then by Theorem C.2, the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$ has a separator of size bounded by k if and only if the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$ has a separator of size bounded by k. In particular, as shown in the proof of Theorem C.2, a separator of size bounded by k for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$ is actually also a separator for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$. Therefore, in this case, we can recursively work on the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$, as given in step 6.1. On the other hand, if $m' \neq m_1$, which means $m' > m_1$, then we simply branch on the vertex u in two cases: (1) including u in the separator and recursively working on the remaining graph for a separator of size bounded by $k - 1$, as given by step 6.2; and (2) excluding u from the separator thus looking for a separator that does not include u and is of size bounded by k for the instance $(G, \{T_1, T_2, \ldots, T_l\}, k)$. By Lemma C.3, the second case is equivalent to finding a separator of size bounded by k for the instance $(G, \{T_1', T_2, \ldots, T_l\}, k)$. This case is thus handled by step 6.3.

This completes the proof of the correctness of the algorithm. Now we analyze the complexity of the algorithm.

The recursive execution of the algorithm can be described as a search tree $\mathcal{T}$. We first count the number of leaves in the search tree $\mathcal{T}$. Note that only steps 6.2-6.3 of the algorithm correspond to branches in the search tree $\mathcal{T}$. Let $D(k, m_1)$ be the total number of leaves in the search tree $\mathcal{T}$ for the algorithm $\mathbf{NMC}(G, \{T_1, T_2, \ldots, T_l\}, k)$, where m_1 is the size of a minimum V-cut between the sets T_1 and $\bigcup_{j=2}^{l} T_j$. Then

steps 6.2-6.3 induce the following recurrence relation:

$$D(k, m_1) \leq D(k-1, m'') + D(k, m''') \tag{3.1}$$

where m'' is the size of a minimum V-cut between T_1 and $\bigcup_{j=2}^{l} T_j$ *in the graph* $G - u$ as given in step 6.2, and m''' is the size of a minimum V-cut between T_1' and $\bigcup_{j=2}^{l} T_j$ *in the graph* G as given in step 6.3. Note that $m_1 - 1 \leq m'' \leq m_1$ because removing the vertex u from G cannot increase the size of a minimum V-cut between two sets, and can decrease the size of a minimum V-cut between the two sets by at most 1. Moreover, by Lemma C.1 and because of step 6.1, the size m''' of a minimum V-cut between T_1' and $\bigcup_{j=2}^{l} T_j$ in step 6.3 is at least $m_1 + 1$. Summarizing these, we have

$$m_1 - 1 \leq m'' \leq m_1 \qquad \text{and} \qquad m''' \geq m_1 + 1 \tag{3.2}$$

Define a measure function M such that $M(D(k, m_1)) = 2k - m_1$. Then $M(D(k-1, m'')) = 2(k-1) - m''$, and $M(D(k, m''')) = 2k - m'''$. By Inequalities (3.2), $2(k-1) - m'' < 2k - m_1$, and $2k - m''' < 2k - m_1$. Hence $M(D(k-1, m'')) < M(D(k, m_1))$ and $M(D(k, m''')) < M(D(k, m_1))$, i.e. in every branching step, the measures of two search tree corresponding to the two new branched instances will reduce by at least 1.

We also point out that certain non-branching steps (i.e., steps 2, 5.1, and 6.1) may also change the values of k and m_1, thus changing the value $M(D(k, m_1)) = 2k - m_1$. However, none of these steps increases the value $M(D(k, m_1)) = 2k - m_1$: (1) step 2 decreases the value k by 1 and the value m_1 by at most 1, which as a total will decrease the value $M(D(k, m_1)) = 2k - m_1$ by at least 1; (2) step 5.1 keeps the value k unchanged and, since we have $m_1 = 0$ before the execution of this step, the new value m_1 is at least as large as the old value m_1. As a consequence, the value $M(D(k, m_1)) = 2k - m_1$ is not increased; (3) finally, step 6.1 does not change the

values k and m_1, thus neither changes the value $M(D(k, m_1)) = 2k - m_1$. In summary, the value $M(D(k, m_1)) = 2k - m_1$ after a branching step to the next branching step can never be increased.

Our initial instance starts with $M(D(k, m_1)) = 2k - m_1 \leq 2k$. In the case $M(D(k, m_1)) = 2k - m_1 = 0$, because we also have the conditions $k \geq m_1 \geq 0$, we must have $m_1 = 0$ and $k = 0$, in this case the algorithm can solve the instance without further branching. Therefore, the original instance is branched at most $2k$ times. Combining Inequalities (3.1), we have

$$D(k, m_1) \leq 2^{2k},$$

and the search tree $\mathcal{T}$ has at most 2^{2k} leaves.

Finally, it is easy to verify that along each root-leaf path in the search tree $\mathcal{T}$, the running time of the algorithm is bounded by $O(kn^3)$, where n is the number of vertices in the graph. In conclusion, the running time of the algorithm $\mathbf{NMC}(G, \{T_1, T_2, \ldots, T_l\}, k)$ is bounded by $O(k4^k n^3)$.

This completes the proof of the theorem. □

D. Chapter Conclusion

In this chapter, we developed new and powerful techniques that lead to a more efficient branch and bound algorithm of running time $O(k4^k n^3)$ for the p-node multiway cut problem. The algorithm significantly improves previous algorithms for the problem. More recently, our techniques have been extended to directed graphs that lead to a parameterized algorithm whose running time is of the form $f(k)n^{O(1)}$ for the pd-feedback vertex set problem [27], thus resolving an outstanding open problem in parameterized computation and complexity.

One interesting place in this chapter is the extended idea for the branch and bound method in designing our algorithm for the p-node multiway cut problem. Usually, when we use the branch and bound method to design parameterized algorithms, the original instance, that is related to a parameter k, is branched into two or more sub-instances that each is related to a parameter $k' < k$. We continue the process of branching until we reach sub-instances that their parameters are 0, where in this case, all sub-instances can be solved in polynomial time. Basically, every time when we branch, we have only one direction to make the instance simpler, i.e. we reduce the value of the only parameter. Correspondingly, we have only one condition to stop branching, i.e. we stop branching when the parameter is 0. However, when solving the p-node multiway cut problem, we had more than one direction (i.e. values of more than one parameter are changed) to simplify the instance and more than one condition to stop branching. In each branch, we obtained two sub-instances such that for one sub-instance, k was reduced by 1 and m_1 was reduced by at most 1 (m_1 might keep the same value), where for another sub-instance, k kept the same value and m_1 was increased by 1. The condition to stop branching was either $k = 0$ or $k < m_1$.

This new idea about branch and bound helped us to develop improved algorithm for the p-node multiway cut problem and will further help us to design effective parameterized algorithms for parameterized NP-hard problems.

CHAPTER IV

DIVIDE-AND-CONQUER*

The main idea of the divide and conquer method is to split an instance (x, k), where k is the parameter and x is the remaining part of the input of the instance, of parameterized NP-hard problems into two sub-instances $(x_1, k/2)$ and $(x_2, k/2)$. Then two sub-instances $(x_1, k/2)$ and $(x_2, k/2)$ are solved independently. In this chapter, we will use divide and conquer method to design improved parameterized algorithms for the PW-PATH, PW-r-D MATCHING and PW-r-SET PACKING problems. We also use the subset partition technique to solve the PW-SET SPLITTING problem.

A. Parameterized Path, Matching and Packing Problems

1. Introduction

The PATH, MATCHING, and PACKING problems are well-known NP-hard problems. As their new applications in bioinformatics, the study of new parameterized algorithms for the parameterized PATH, MATCHING, and PACKING problems has recently drawn considerable attention [3, 22, 50, 66, 73, 75, 91, 92, 101, 110].

Let G be an undirected graph. A *path* ρ in G is a sequence of vertices $\langle v_1, \ldots, v_k \rangle$ in G such that for all $1 \leq i \leq k-1$, $[v_i, v_{i+1}]$ is an edge in G, where k is called the *length* of ρ. The path ρ is *simple* if no vertex is repeated in the sequence. A *k-path* in

---

*Reprinted with permission from "Randomized Divide-and-Conquer: Improved Path, Matching, and Packing Algorithms" by Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, Fenghui Zhang, 2009. *SIAM Journal on Computing*, 38, 2526-2547, Copyright 2009 by Society for Industrial and Applied Mathematics (SIAM).

*Reprinted with permission from "Improved Parameterized Set Splitting Algorithm: A Probabilistic Approach" by Jianer Chen, Songjian Lu, 2009. *Algorithmica*, 54, 472-489, Copyright 2009 by Springer.

G is a simple path of length k in G. If the graph G is weighted (i.e., if each vertex in G is assigned a *weight* that is a real number), then the *weight* of the path ρ is equal to the sum of weights of the vertices in ρ.

> PW-PATH: Given a weighted undirected graph G of n vertices and m edges and a parameter k, either construct a k-path in G whose weight is the maximum over all k-paths in G, or report that no k-path exists in G.

There is an *unweighted version* for the PW-PATH problem, i.e. the P-PATH problem, in which the input graph is unweighted. The unweighted version can be trivially reduced to the general version by regarding an unweighted graph as a weighted graph in which all vertices are assigned weight 1. There is also a version of the problem on directed graphs, where we are looking for a directed k-path of the maximum weight in a weighted and directed graph. We will be focused on the problem on undirected graphs, and in certain places, extend our discussion to directed graphs.

A set M of points in the r-dimensional Euclidean space $\mathbb{R}^r$ is a *matching* if no two points in M agree in any coordinate. A *k-matching* is a matching consisting of exactly k points in $\mathbb{R}^r$. If each point in $\mathbb{R}^r$ is assigned a weight, then the *weight* of a matching M is equal to the sum of weights of the points in M.

> PW-r-D MATCHING: Given a set S of n points in the r-dimensional Euclidean space $\mathbb{R}^r$ and a parameter k, where each point in S is assigned a weight, either construct a k-matching in S whose weight is the maximum over all k-matchings in S, or report that no k-matching exists in S.

We also have an *unweighted version* for the PW-r-D MATCHING problem, which assumes that all points in the input set S are assigned weight 1. We usually call it the P-r-D MATCHING problem.

An *r-set* is a set containing exactly r elements. A collection $\mathcal{P}$ of r-sets is a *packing* if no two r-sets in $\mathcal{P}$ intersect. A *k-packing* is a packing consisting of exactly k r-sets. If each r-set is assigned a weight, then the *weight* of a packing $\mathcal{P}$ is the sum of weights of the r-sets in $\mathcal{P}$.

> PW-r-SET PACKING: Given a collection $\mathcal{C}$ of n r-sets and a parameter k, where each r-set in $\mathcal{C}$ is assigned a weight, either construct a k-packing in $\mathcal{C}$ whose weight is the maximum over all k-packings in $\mathcal{C}$, or report that no k-packing exists in $\mathcal{C}$.

The *unweighted version* of the PW-r-SET PACKING problem assumes that all r-sets in the input collection $\mathcal{C}$ are assigned weight 1. we usually call it the P-r-SET PACKING problem.

It is easy to see that the PW-r-D MATCHING problem can be trivially reduced to the PW-r-SET PACKING problem. Therefore, any algorithm solving the latter can be directly used to solve the former.

Most previous algorithms for the PW-PATH, PW-r-D MATCHING and PW-r-SET PACKING problems were presented for the unweighted versions of the problems. Many of these algorithms, with minor modifications, also work for the general versions of the problems.

The PW-PATH problem is closely related to a number of well-known NP-hard problems, such as the LONGEST PATH, HAMILTONIAN PATH, and TRAVELING SALESMAN problems. Earlier algorithms [12, 92] for the P-PATH problem have running time bounded by $2^k k! n^{O(1)}$. Papadimitriou and Yannakakis [97] studied a restricted version of the problem, and conjectured that it is solvable in polynomial time to determine if a graph contains a $(\log n)$-path. This conjecture was confirmed by Alon, Yuster, and Zwick [3], who presented for the P-PATH problem randomized and deterministic

algorithms of running time $2^{O(k)}n^{O(1)}$. Recently, the PW-PATH problem has found applications in bioinformatics for detecting signaling pathways in protein interaction networks [110] and for biological subnetwork matchings [73].

The P-r-D MATCHING and P-r-SET PACKING problems were first studied by Downey and Fellows [44], where they developed deterministic algorithms of running time $(rk)!(rk)^{3rk}n^{O(1)}$. Based on the *greedy localization* techniques [22], Chen et al. improved complexities of these problems to $(r-1)^k((r-1)k/e)^{k(r-2)}n^{O(1)}$ [22, 66] . Koutis [75] developed randomized algorithms of time $10.88^{rk}n^{O(1)}$ and space $O(2^{rk}+rn)$, and deterministic algorithms of time $2^{O(rk)}n^{O(1)}$ and space $O(2^{rk}+rn)$ for the problems. The deterministic upper bound was further improved to $2^{5rk-4k}\binom{6(r-1)k+k}{rk}n^{O(1)}$ (still with exponential space) by Fellows *et al.* [50]. The problems of packing a small subgraph in a given graph, such as the P-TRIANGLE PACKING problem, can be transformed into the PW-r-SET PACKING problem directly. Algorithms for this kind of graph packing problems have also been studied [50, 91, 101].

Currently, the best randomized and deterministic algorithms for the P-PATH, P-r-D MATCHING, and P-r-SET PACKING problems [3, 50, 75] are based on the *color-coding* technique developed by Alon, Yuster, and Zwick [3]. The technique is based on constructing an (n,k)-family of perfect hash functions and a dynamic programming process on k-colored instances. For example, the randomized algorithm for the P-PATH problem given in [3] based on this technique runs in time $(2e)^k n^{O(1)} = 5.44^k n^{O(1)}$ and space $2^k n^{O(1)}$. Because of the lower bound $\Omega(e^k)$ on the size of (n,k)-families of perfect hash functions [95], and of the nature of dynamic programming, it seems difficult to further improve the time complexity and to avoid exponential space complexity for algorithms based on this technique.

In this section, we develop new exponential-time algorithmic techniques that lead to improved randomized and deterministic algorithms for the PW-PATH, PW-r-D

MATCHING, and PW-r-SET PACKING problems. Our main idea is using a divide-and-conquer method, as follows. Suppose that we are looking for a subset S_h of h elements in a universal set U. Moreover, we assume that the subset S_h is "hierarchical" in the sense that such a subset is constituted by smaller subsets of similar properties. We first randomly partition the universal set U into two disjoint parts U_1 and U_2. A simple probabilistic analysis shows that with an effective probability, the desired subset S_h is evenly split by this random partition into two smaller subsets of similar properties. This enables us to recursively look in each of the parts U_1 and U_2 for a smaller subset of $h/2$ elements in S_h and of similar properties, and to finally combine the smaller subsets to obtain the desired subset S_h in the original universal set U.

Take the PW-3-SET PACKING problem as an example. For a given collection $\mathcal{C}$ of 3-sets and a parameter k, the universal set U is the set of all elements that appear in the 3-sets in $\mathcal{C}$, and the desired subset S_{3k} consists of the $3k$ elements from k 3-sets that make a k-packing $\mathcal{P}_k$ of the maximum weight in $\mathcal{C}$. It is not difficult to see that with a probability of at least $1/2^{3k}$, a random partition of U into two parts U_1 and U_2 includes all $3k/2$ elements from $k/2$ 3-sets in $\mathcal{P}_k$ in the part U_1 and all $3k/2$ elements from the other $k/2$ 3-sets in $\mathcal{P}_k$ in the part U_2.[3] Moreover, the two parts U_1 and U_2 induce two subcollections $\mathcal{C}_1$ and $\mathcal{C}_2$ of $\mathcal{C}$, respectively, after deleting all 3-sets that contain both elements in U_1 and U_2. Therefore, a k-packing of the maximum weight in $\mathcal{C}$ can be constructed by combining a $(k/2)$-packing of the maximum weight in $\mathcal{C}_1$ and a $(k/2)$-packing of the maximum weight in $\mathcal{C}_2$, which can be recursively constructed.

This simple method leads directly to randomized algorithms with improved running time and with polynomial space. For the PW-PATH problem, this new method

[3]We will show in subsection 4 that with a more careful analysis, we can derive a better probability.

gives a randomized algorithm of time $O(4^k k^{2.7} m)$ and space $O(nk \log k + m)$, improving the previously best randomized algorithm for the problem of time $O(5.44^k km)$ and space $O(2^k kn + m)$ [3]. For the PW-r-D MATCHING and PW-r-SET PACKING problems, the method gives randomized algorithms of time $4^{(r-1)k} k^{\log k/3} n^{O(1)}$ and space $(rk \log k + rn)$, improving the previously best randomized algorithms for the problems of time $10.88^{rk} n^{O(1)}$ and space $O(2^{rk} + rn)$ [75]. Furthermore, these randomized algorithms can be derandomized, which leads to deterministic algorithms that significantly improve previous best deterministic algorithms for the corresponding problems.

The major techniques and results reported in this section were independently discovered by our research group at Texas A&M University and by the research group at RWTH Aachen University. Preliminary results from the two groups were reported independently at SODA'2007 and WG'2006, respectively [29, 74].

2. On a class of recurrence relations

The analysis of many of our randomized and deterministic algorithms presented in this section requires solving recurrence relations of a special form, which seems neither standard nor trivial. This subsection is devoted to giving a thorough and formal study of this class of recurrence relations.

Theorem A.1 *Let $T(k, n)$ be a function satisfying the following conditions: there are functions $t(n)$, $f(k)$ and $h(n)$, and a real number $a \geq 1$ such that*

(A) *$T(k, n) \leq O(a^{2k} t(n))$ for all $k \leq h(n)$; and*

(B) *for $k > h(n)$, the function $T(k, n)$ satisfies $T(k, n) \leq f(k) a^k [t(n) + T(k_1, n) + T(k_2, n)]$,*

where $k_1 = \lceil k/2 \rceil$ and $k_2 = \lfloor k/2 \rfloor$.

Then $T(k,n) = O(a^{2k}g(k)t(n))$*, where* $g(k)$ *is any function satisfying:*

(C) $g(k) \geq 1$*, for all* $k \geq 1$*; and*

(D) *there exist an integer* $k_0 \geq 1$ *and a positive real number* $d_0 < 1$ *such that for all* $k \geq k_0$*,*

$$H(a,k,f(k),g(k)) \stackrel{\text{def}}{=\!=} \frac{a^{2k_1}f(k)g(k_1) + a^{2k_2}f(k)g(k_2)}{a^k g(k)} \leq d_0.$$

PROOF. Pick a constant c_0 so that $d_0(1 + 1/(2c_0)) \leq 1$. We claim

$$T(k,n) \leq c_0 a^{2k} g(k) t(n), \tag{4.1}$$

which will prove the theorem.

We prove (4.1) by induction on k. For the values of k that are bounded by $h(n)$, (4.1) holds true by Condition (A) if the constant c_0 is sufficiently large. Applying induction on Condition (B) when $k > h(n)$, we obtain

$$\begin{aligned} T(k,n) &\leq f(k)a^k[t(n) + T(k_1,n) + T(k_2,n)] \\ &\leq f(k)a^k[t(n) + c_0 a^{2k_1} g(k_1)t(n) + c_0 a^{2k_2} g(k_2)t(n)] \\ &= c_0 a^{2k} g(k) t(n) \left[\frac{f(k)}{c_0 a^k g(k)} + \frac{a^{2k_1}f(k)g(k_1) + a^{2k_2}f(k)g(k_2)}{a^k g(k)}\right]. \end{aligned} \tag{4.2}$$

By Condition (D), the second term in the bracket in (4.2) is $H(a,k,f(k),g(k))$, which is bounded by d_0. Combining Condition (D) with the fact that the values k_1, k_2, $g(k_1)$, $g(k_2)$, a, and k are all larger than or equal to 1, we also get the following bound for the first term in the bracket in (4.2):

$$\frac{f(k)}{c_0 a^k g(k)} \leq \frac{d_0}{2c_0}.$$

By the way we selected the value of c_0, we have $d_0 + d_0/(2c_0) = d_0(1 + 1/(2c_0)) \leq 1$.

Therefore,

$$T(k,n) \leq c_0 a^{2k} g(k) t(n),$$

and the induction goes through. □

Applying Theorem A.1, we obtain a sequence of corollaries that give the bounds that are specifically needed in the analysis of our algorithms presented in this section. Corollary A.2 below will be used in the analysis of our algorithm for the PW-PATH problem.

Corollary A.2 *Suppose that a function $T(k,n)$ satisfies $T(1,n) = O(t(n))$ and the following recurrence relation for $k \geq 2$:*

$$T(k,n) \leq c_0 a^k [t(n) + T(k_1,n) + T(k_2,n)],$$

where $k_1 = \lceil k/2 \rceil$, $k_2 = \lfloor k/2 \rfloor$, and c_0 and $a \geq 1$ are constants. Then $T(k,n) = O(a^{2k} k^\alpha t(n))$, where α is any constant satisfying $\alpha > \log_2(c_0(a^2+1)/a)$.

PROOF. Using the notations in Theorem A.1, here we have $f(k) = c_0$ and $h(n) \equiv 1$. We verify that the function $g(k) = k^\alpha$ satisfies Conditions (C) and (D) in Theorem A.1. Condition (C) is trivially satisfied. To verify Condition (D), we have

$$H(a,k,c_0,g(k)) = \frac{c_0 a^{2k_1} k_1^\alpha + c_0 a^{2k_2} k_2^\alpha}{a^k k^\alpha}.$$

If k is even, we have

$$H(a,k,c_0,g(k)) = \frac{c_0 a^k (k/2)^\alpha + c_0 a^k (k/2)^\alpha}{a^k k^\alpha} = \frac{c_0}{2^{\alpha-1}}.$$

Since $\alpha > \log_2(c_0(a^2+1)/a)$ and it is easy to see that $(a^2+1)/a \geq 2$, we get $c_0/2^{\alpha-1} < 1$.

On the other hand, suppose that k is odd, then

$$\begin{aligned} H(a,k,c_0,g(k)) &= \frac{c_0 a^{k+1}((k+1)/2)^\alpha + c_0 a^{k-1}((k-1)/2)^\alpha}{a^k k^\alpha} \\ &= \frac{c_0}{2^\alpha a}\left[a^2\left(1+\frac{1}{k}\right)^\alpha + \left(1-\frac{1}{k}\right)^\alpha\right]. \end{aligned} \tag{4.3}$$

Since when $k \to \infty$, $a^2(1+1/k)^\alpha + (1-1/k)^\alpha \to a^2+1$, the above value approaches to $c_0(a^2+1)/(2^\alpha a)$. From $\alpha > \log_2(c_0(a^2+1)/a)$, we have $c_0(a^2+1)/(2^\alpha a) < 1$. Therefore, there must be a constant k_0 such that $d = c_0[a^2(1+1/k_0)^\alpha + (1-1/k_0)^\alpha]/(2^\alpha a) < 1$, and for all odd numbers $k \geq k_0$, we have $H(a,k,c_0,g(k)) \leq d$.

Now if we let $d_0 = \max\{c_0/2^{\alpha-1}, d\}$, then $d_0 < 1$ and for all $k \geq k_0$, we have

$$H(a,k,c_0,g(k)) \leq d_0.$$

Thus, the function $g(k) = k^\alpha$ satisfies Condition (D) in Theorem A.1. The corollary follows. □

Corollary A.3 below will be used in the analysis of our algorithms for the PW-r-D MATCHING and PW-r-SET PACKING problems.

Corollary A.3 *Suppose that a function $T(k,n)$ satisfies $T(1,n) = O(t(n))$ and the following recurrence relation for $k \geq 2$:*

$$T(k,n) \leq c_0 k^b a^k [t(n) + T(k_1,n) + T(k_2,n)],$$

where $k_1 = \lceil k/2 \rceil$, $k_2 = \lfloor k/2 \rfloor$, and c_0, $b > 0$, and $a \geq 1$ are all constants. Then $T(k,n) = O(a^{2k} k^{\alpha \log k} t(n))$, where α is any constant satisfying $\alpha > b/2$.

PROOF. Using the notations in Theorem A.1, here we have $f(k) = c_0 k^b$ and $h(n) \equiv 1$. We verify that the function $g(k) = k^{\alpha \log k}$ satisfies Conditions (C) and (D) in Theorem A.1. Condition (C) is trivially satisfied. To verify Condition (D), we

have

$$\begin{aligned} H(a,k,c_0k^b,g(k)) &= \frac{c_0k^ba^{2k_1}k_1^{\alpha\log k_1}+c_0k^ba^{2k_2}k_2^{\alpha\log k_2}}{a^kk^{\alpha\log k}} \\ &\le \frac{c_0k^ba^{k+1}((k+1)/2)^{\alpha\log((k+1)/2)}+c_0k^ba^{k+1}((k+1)/2)^{\alpha\log((k+1)/2)}}{a^kk^{\alpha\log k}} \\ &= \frac{2^{\alpha+1}c_0k^ba}{(k+1)^{2\alpha}}\cdot\frac{(k+1)^{\alpha\log(k+1)}}{k^{\alpha\log k}} \\ &\le \frac{2^{\alpha+1}c_0a}{(k+1)^{2\alpha-b}}\cdot\frac{(k+1)^{\alpha\log(k+1)}}{k^{\alpha\log k}}. \end{aligned}$$

To see the limit of this value when k approaches ∞, note that if we let $r>1$ be a constant such that $2\alpha-b-2\alpha\log r>0$ (note $\alpha>b/2$), then the above expression can be rewritten as

$$\frac{2^{\alpha+1}c_0a}{(k+1)^{2\alpha-b}}\cdot\frac{(k+1)^{\alpha\log(k+1)}}{k^{\alpha\log k}}=\frac{2^{\alpha+1}c_0a}{r^{\alpha\log r}(k+1)^{2\alpha-b-2\alpha\log r}}\cdot\frac{((k+1)/r)^{\alpha\log((k+1)/r)}}{k^{\alpha\log k}}.$$

Now since $r>1$, $(k+1)/r<k$ when k is sufficiently large. Therefore, the value approaches 0 when $k\to\infty$. In particular, this implies that there is an integer k_0 and a constant $d_0<1$ such that when $k\ge k_0$, $H(a,k,c_0k^b,g(k))\le d_0$. This completes the proof of the corollary. □

3. A randomized algorithm for the PW-PATH

Now we are ready to present our randomized algorithms. The first problem we are dealing with is the PW-PATH problem that looks for a k-path of the maximum weight in a weighted graph.

Fix a weighted graph $G=(V,E)$. For any $V'\subseteq V$, denote by $G[V']$ the subgraph of G induced by V'. The *concatenation* of two paths $\rho_1=\langle v_1,\ldots,v_l\rangle$ and $\rho_2=\langle w_1,\ldots,w_h\rangle$ in G, where $[v_l,w_1]$ is an edge in G, is the path $\langle v_1,\ldots,v_l,w_1\ldots,w_h\rangle$. We denote by $\rho_\emptyset$ the special 0-*path* (i.e., the empty path containing no vertex), and define that the concatenation of $\rho_\emptyset$ and any path ρ gives the path ρ. An h-path ρ is

also called a (v, h)-*path* if v is an end vertex of ρ.

Let P_l be a set of l-paths in G, and let $V' \subseteq V$ such that no vertex in V' is on any path in P_l. A (v, h)-*path* ρ *is in* $P_l \odot V'$ if $v \in V'$ and ρ is a concatenation of an l-path in P_l and an $(h-l)$-path in $G[V']$. In particular, for $P_0 = \{\rho_\emptyset\}$, any $(v, 1)$-path in $P_0 \odot V'$ consists of the single vertex v in V'.

On a set P_l of l-paths in G and $V' \subseteq V$, where P_l contains at most one (v, l)-path for each vertex v, and no vertex in V' is on any path in P_l, our algorithm **find-paths**(P_l, V', h) returns a set P_{l+h} of $(l + h)$-paths in $P_l \odot V'$. In particular, the algorithm **find-paths**$(\{\rho_\emptyset\}, V, k)$ returns a set of k-paths in the graph G. The algorithm is given in Figure 3.

Theorem A.4 *Let P_l and V' be defined as above. For any vertex v in V', if there are $(v, l + h)$-paths in $P_l \odot V'$, then with probability larger than $1 - 1/e > 0.632$ (here e is the base of the natural logarithm), the set P_{l+h} returned by the algorithm* **find-paths**(P_l, V', h) *contains a $(v, l + h)$-path in $P_l \odot V'$ whose weight is the maximum over all $(v, l + h)$-paths in $P_l \odot V'$. The algorithm* **find-paths**(P_l, V', h) *runs in time $O(4^h h^{2.7} m)$ and in space $O(n(l + h) \log h + m)$.*

Proof. First note that by steps 2.4–2.6 and steps 3.6–3.8, if the set P_{l+h} contains a $(v, l+h)$-path ρ, then ρ must be a valid $(v, l+h)$-path in $P_l \odot V'$. Therefore, if there is no $(v, l + h)$-path in $P_l \odot V'$, then the set P_{l+h} cannot contain a $(v, l + h)$-path.

Thus we assume that in the graph G there are $(v, l + h)$-paths in $P_l \odot V'$. Let

$$\rho_{l+h} = \langle u_1, \ldots, u_l, w_1, \ldots, w_h \rangle$$

be a $(v, l + h)$-path in $P_l \odot V'$ whose weight is the maximum over all $(v, l + h)$-paths in $P_l \odot V'$, where $\langle u_1, \ldots, u_l \rangle$ is an l-path in P_l, $\langle w_1, \ldots, w_h \rangle$ is an h-path in $G[V']$,

find-paths(P_l, V', h)
input: a set P_l of l-paths; $V' \subseteq V$ and no vertex in V' is on a path in P_l; an integer $h \geq 1$;
output: a set P_{l+h} of $(l+h)$-paths in $P_l \odot V'$;

1. $P_{l+h} = \emptyset$;
2. **if** $h = 1$ **then**
2.1. **if** $P_l = \{\rho_\emptyset\}$ **then** P_{l+1} contains a $(u, 1)$-path for each vertex $u \in V'$;
return P_{l+1};
2.2. **else for** each (w, l)-path ρ_l in P_l and each $u \in V'$, where $[w, u]$ is an edge in G, **do**
2.3. concatenate ρ_l and u to make a $(u, l+1)$-path ρ_{l+1} in $P_l \odot V'$;
2.4. **if** P_{l+1} contains no $(u, l+1)$-path **then** add ρ_{l+1} to P_{l+1};
2.5. **else if** the $(u, l+1)$-path ρ'_{l+1} in P_{l+1} has a weight smaller than that of ρ_{l+1} **then**
2.6. replace ρ'_{l+1} in P_{l+1} by ρ_{l+1};
2.7. **return** P_{l+1};
3. **loop** $2.51 \cdot 2^h$ times **do**
3.1. randomly partition the vertices in V' into two parts V_L and V_R;
3.2. $P^L_{l+\lceil h/2 \rceil} =$ **find-paths**$(P_l, V_L, \lceil h/2 \rceil)$;
3.3. **if** $P^L_{l+\lceil h/2 \rceil} \neq \emptyset$ **then**
3.4. $P^R_{l+h} =$ **find-paths**$(P^L_{l+\lceil h/2 \rceil}, V_R, \lfloor h/2 \rfloor)$;
3.5. **for** each $(u, l+h)$-path ρ_{l+h} in P^R_{l+h} **do**
3.6. **if** P_{l+h} contains no $(u, l+h)$-path in $P_l \odot V'$ **then** add ρ_{l+h} to P_{l+h};
3.7. **else if** the $(u, l+h)$-path ρ'_{l+h} in P_{l+h} has a weight smaller than that of ρ_{l+h} **then**
3.8. replace ρ'_{l+h} in P_{l+h} by ρ_{l+h};
4. **return** P_{l+h}.

Fig. 3. A randomized algorithm for the PW-PATH problem

and $w_h = v$. We prove the theorem by induction on $h \geq 1$.

Consider the case $h = 1$. If $P_l = \{\rho_\emptyset\}$ (in this case $l = 0$), then the set P_{l+1} returned by step 2.1 contains the (unique) $(v, 1)$-path in $P_l \odot V'$, which is obviously of the maximum weight. On the other hand, if $l > 0$, then when the (u_l, l)-path $\langle u_1, \ldots, u_l \rangle$ in P_l and the vertex $w_h = w_1 = v$ are examined in step 2.2, the path ρ_{l+1} is constructed in step 2.3, and steps 2.4–2.6 ensure that a $(v, l+1)$-path of the maximum weight is included in the set P_{l+1}. This proves the case $h = 1$.

Now suppose that $h > 1$. We rewrite the path ρ_{l+h} as

$$\rho_{l+h} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1}, \ldots, w_h \rangle,$$

where $h_1 = \lceil h/2 \rceil$. With probability $1/2^h$, step 3.1 of the algorithm puts vertices $w_1, \ldots, w_{h_1}$ into V_L, and vertices $w_{h_1+1}, \ldots, w_h$ into V_R. If this is the case, then the path

$$\rho_{l+h_1} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1} \rangle$$

is a $(w_{h_1}, l+h_1)$-path in $P_l \odot V_L$. By the induction hypothesis, with probability larger than $1 - 1/e$, the set $P^L_{l+h_1}$ obtained in step 3.2 contains a $(w_{h_1}, l + h_1)$-path $\overline{\rho}_{l+h_1}$ in $P_l \odot V_L$ whose weight is at least as large as that of ρ_{l+h_1}. Now the concatenation of the path $\overline{\rho}_{l+h_1}$ and the path $\langle w_{h_1+1}, \ldots, w_h \rangle$ is a $(w_h, (l + h_1) + (h - h_1))$-path (i.e., a $(v, l + h)$-path) in $P^L_{l+h_1} \odot V_R$. Thus, by our induction hypothesis again, with probability larger than $1-1/e$, the set P^R_{l+h} obtained in step 3.4 contains a $(v, l+h)$-path $\overline{\rho}_{l+h}$ in $P^L_{l+h_1} \odot V_R$ whose weight is at least as large as the sum of the weight of the path $\overline{\rho}_{l+h_1}$ and the weight of the path $\langle w_{h_1+1}, \ldots, w_h \rangle$. Since the weight of $\overline{\rho}_{l+h_1}$ is not smaller than that of ρ_{l+h_1}, we conclude that the weight of the path $\overline{\rho}_{l+h}$ is not smaller than that of ρ_{l+h}. Finally, since the path $\overline{\rho}_{l+h}$ is a concatenation of a $(w, l+h_1)$-path ρ'_{l+h_1} in $P_l \odot V_L$ (for some vertex $w \in V_L$) and a path in $G[V_R]$, where the path ρ'_{l+h_1} is a concatenation of a path in P_l and a path in $G[V_L]$, we derive that $\overline{\rho}_{l+h}$ is actually a $(v, l+h)$-path in $P_l \odot V'$. Since the weight of $\overline{\rho}_{l+h}$ is not smaller than the weight of ρ_{l+h}, and by our assumption, the path ρ_{l+h} has the maximum weight over all $(v, l+h)$-paths in $P_l \odot V'$, we conclude that $\overline{\rho}_{l+h}$ must also be a $(v, l+h)$-path of the maximum weight in $P_l \odot V'$.

In conclusion, with probability $1/2^h$, step 3.1 includes the vertices w_1, …, w_{h_1} in V_L and the vertices w_{h_1+1}, …, w_h in V_R. If this is the partition, then with probability

larger than $1 - 1/e$, the set $P^L_{l+h_1}$ in step 3.2 contains the $(w_{h_1}, l + h_1)$-path $\overline{\rho}_{l+h_1}$. In case the set $P^L_{l+h_1}$ contains the path $\overline{\rho}_{l+h_1}$, with probability larger than $1 - 1/e$, the set P^R_{l+h} in step 3.4 contains a $(v, l + h)$-path of the maximum weight. Therefore, by steps 3.5–3.8, in each loop of step 3, the probability q that the set P_{l+h} contains a $(v, l + h)$-path of the maximum weight is larger than

$$\frac{(1 - 1/e)^2}{2^h} > \frac{1}{2.51 \cdot 2^h}.$$

Since step 3 of the algorithm loops $2.51 \cdot 2^h$ times, the overall probability that the algorithm returns the set P_{l+h} that contains a $(v, l + h)$-path of the maximum weight is

$$1 - (1 - q)^{2.51 \cdot 2^h} > 1 - \left(1 - \frac{1}{2.51 \cdot 2^h}\right)^{2.51 \cdot 2^h} > 1 - \frac{1}{e}.$$

This proves the first part of the theorem.

To analyze the time complexity, let $T(h, m)$ be the running time of the algorithm **find-paths**(P_l, V', h), where m is the number of edges in the original graph G. Clearly we have $T(1, m) = O(m)$. From the algorithm, we have the following recurrence relation when $h > 1$:

$$T(h, m) = 2.51 \cdot 2^h [cm + T(\lceil h/2 \rceil, m) + T(\lfloor h/2 \rfloor, m)],$$

where $c > 0$ is a constant. Using the notations in Corollary A.2, here we have $c_0 = 2.51$, $a = 2$, and $t(n) = cm$. Now $\log_2(c_0(a^2 + 1)/a) \geq 2.64$. Thus, by Corollary A.2, the running time $T(h, m)$ of the algorithm **find-paths**(P_l, V', h) is $O(4^h h^{2.7} m)$.

In terms of the space complexity, each recursive call to the algorithm **find-paths**(P_l, V', h) uses $O(n(l+h))$ space (mainly for the sets $P^L_{l+h_1}$, P^R_{l+h}, and P_{l+h}, noting that for each vertex w in the graph G, each of these sets contains at most one $(w, \times *)$-path). Since the recursive depth of the algorithm **find-paths**(P_l, V', h) is $O(\log h)$,

we conclude that the space complexity of the algorithm **find-paths**(P_l, V', h) is $O(n(l+h)\log h + m)$. □

To obtain a randomized algorithm solving the PW-PATH problem with a required error bound, we simply run **find-paths**$(\{\rho_\emptyset\}, V, k)$ sufficiently many times, each taking $O(4^k k^{2.7} m)$ time and $O(nk \log k + m)$ space. For example, to achieve an error bound of 0.0001, we can run the algorithm t times, where t satisfies $(1/e)^t \leq 0.0001$ (e.g., $t = 10$). Note that the set P_k returned by the algorithm **find-paths**$(\{\rho_\emptyset\}, V, k)$ contains at most one (v, k)-path for each vertex v in the graph G. Therefore, by picking the (v, k)-path of the maximum weight in P_k, this process returns a k-path of the maximum weight in the graph G with an arbitrarily small error bound.

Corollary A.5 *There is a randomized algorithm of running time $O(4^k k^{2.7} m)$ and space $O(nk \log k + m)$ that solves the* PW-PATH *problem with an arbitrarily small error bound.*

Remark. The algorithm **find-paths** can be used directly to solve the PW-PATH problem on *directed graphs*, as long as we interpret the edge $[w, u]$ in step 2.2 of the algorithm as a directed edge from w to u. The proof of Theorem A.4 can be applied to directed graphs with no change.

We compare our algorithm in Corollary A.5 with previously known algorithms for the P-PATH problem. To our knowledge, there are two kinds of randomized algorithms for the problem. The first kind is based on random permutations of vertices followed by searching in a directed acyclic graph [3, 73]. The algorithm runs in time $O(mk!)$ and space $O(m)$. The second kind, proposed by Alon, Yuster, and Zwick [3], is based on random coloring of vertices in a graph followed by dynamic programming to search for a simple path of length k in the colored graph. The algorithm runs in

time $O((2e)^k km) = O(5.44^k km)$ and space $O(2^k kn + m)$ (the space is mainly for the dynamic programming phase). Compared to these algorithms, our algorithm has a significantly improved running time and uses polynomial space. In fact, if we are only interested in knowing whether the graph has a path of length k, a slight modification of our algorithm can further reduce the space complexity to $O(n \log k + m)$.

4. Randomized algorithms for PW-MATCHING and PW-PACKING

The randomized divide-and-conquer method described in the previous subsection can also be used to develop improved algorithms for PW-MATCHING and PW-PACKING problems.

Recall that any algorithm solving the PW-r-SET PACKING problem can be used directly to solve the PW-r-D MATCHING problem. Therefore, our discussion in this subsection will be focused on the PW-r-SET PACKING problem, which looks for a k-packing of the maximum weight in a collection of r-sets. We have the following result.

Theorem A.6 *There is a randomized algorithm that solves the* PW-r-SET PACKING *problem in time* $O(4^{(r-1)k} k^{\log k/3} rn)$ *and space* $O(rk \log k + rn)$*, where* n *is the number of* r*-sets in the given instance of the* PW-r-SET PACKING *problem.*

PROOF. Consider the algorithm in Figure 4. Let $k_1 = \lceil k/2 \rceil$. We first prove, by induction on the parameter k, that if the collection $\mathcal{C}$ contains k-packings, then with probability larger than $1 - 1/e$, the algorithm **set-packing**$(\mathcal{C}, k)$ returns a k-packing in $\mathcal{C}$ whose weight is the maximum over all k-packings in $\mathcal{C}$.

This is obviously true when $k = 1$. Now suppose that $k > 1$. Again first note that if the collection $\mathcal{C}$ has no k-packings, then the algorithm **set-packing**$(\mathcal{C}, k)$

set-packing$(\mathcal{C}, k)$
input: a collection $\mathcal{C}$ of r-sets, in which each r-set is assigned a weight, and an integer $k \geq 1$;
output: a k-packing in $\mathcal{C}$ that has the maximum weight over all k-packings in $\mathcal{C}$, or $\emptyset$ if $\mathcal{C}$ has no k-packing.

1. **if** $k = 1$ **then**
   if $\mathcal{C} \neq \emptyset$ **then return** an r-set of the maximum weight in $\mathcal{C}$
   else return $\emptyset$;
2. $\mathcal{P} = \emptyset$;
3. **loop** $2.51 \cdot 2^{rk} / \binom{k}{\lceil k/2 \rceil}$ times **do**
3.1. randomly partition the set elements into two parts S_L and S_R;
3.2. let $\mathcal{C}_L$ be the sub-collection of r-sets in $\mathcal{C}$ in which all elements are in S_L;
3.3. let $\mathcal{C}_R$ be the sub-collection of r-sets in $\mathcal{C}$ in which all elements are in S_R;
3.4. $\mathcal{P}_L =$ **set-packing**$(\mathcal{C}_L, \lceil k/2 \rceil))$; $\mathcal{P}_R =$ **set-packing**$(\mathcal{C}_R, \lfloor k/2 \rfloor))$;
3.5. **if** $\mathcal{P}_L \neq \emptyset$, $\mathcal{P}_R \neq \emptyset$, and the sum of the weights of $\mathcal{P}_L$ and $\mathcal{P}_R$ is larger than the weight of $\mathcal{P}$ **then** $\mathcal{P} = \mathcal{P}_L \cup \mathcal{P}_R$;
4. **return** $\mathcal{P}$.

Fig. 4. A randomized divide-and-conquer algorithm for the PW-r-SET PACKING problem

must return the empty set $\emptyset$. Now suppose that $\mathcal{C}$ contains k-packings, and let $\mathcal{P}_k$ be a k-packing in $\mathcal{C}$ whose weight is the maximum over all k-packings in $\mathcal{C}$. With a probability $\binom{k}{k_1}/2^{rk}$, step 3.1 of the algorithm partitions the rk elements in the k r-sets in $\mathcal{P}_k$ such that the rk_1 elements in (any) k_1 r-sets in $\mathcal{P}_k$ are in S_L while the $r(k-k_1)$ elements in the other $k-k_1$ r-sets in $\mathcal{P}_k$ are in S_R. If this is the case, let $\mathcal{P}_{k_1}^L$ be the set of k_1 r-sets in $\mathcal{P}_k$ whose elements are all in S_L and let $\mathcal{P}_{k-k_1}^R$ be the set of $k-k_1$ r-sets in $\mathcal{P}_k$ whose elements are all in S_R. Note that $\mathcal{P}_{k_1}^L$ is a k_1-packing in $\mathcal{C}_L$ and that $\mathcal{P}_{k-k_1}^R$ is a $(k-k_1)$-packing in $\mathcal{C}_R$. Thus, by the induction hypothesis, with a probability larger than $(1-1/e)^2$, step 3.4 of the algorithm generates a k_1-packing $\mathcal{P}_L$ in $\mathcal{C}_L$ and a $(k-k_1)$-packing $\mathcal{P}_R$ in $\mathcal{C}_R$, such that the weight of $\mathcal{P}_L$ is not smaller than that of $\mathcal{P}_{k_1}^L$ and that the weight of $\mathcal{P}_R$ is not smaller than that of $\mathcal{P}_{k-k_1}^R$. Note that the union of $\mathcal{P}_L$ and $\mathcal{P}_R$ must be a k-packing in $\mathcal{C}$ because no set element appears in both $\mathcal{C}_L$ and $\mathcal{C}_R$. Since the union of $\mathcal{P}_{k_1}^L$ and $\mathcal{P}_{k-k_1}^R$ is the k-packing $\mathcal{P}_k$ that has the

maximum weight over all k-packings in $\mathcal{C}$, we conclude that the union of $\mathcal{P}_L$ and $\mathcal{P}_R$ must be a k-packing of the maximum weight in $\mathcal{C}$. In summary, if the collection $\mathcal{C}$ contains k-packings, then with a probability q larger than

$$\frac{(1-1/e)^2\binom{k}{k_1}}{2^{rk}} > \frac{\binom{k}{k_1}}{2.51 \cdot 2^{rk}},$$

an execution of steps 3.1–3.5 of the algorithm will make $\mathcal{P}$ a k-packing of the maximum weight in $\mathcal{C}$. Now since the loop in step 3 is executed $2.51 \cdot 2^{rk}/\binom{k}{k_1}$ times, with probability at least

$$1-(1-q)^{2.51\cdot 2^{rk}/\binom{k}{k_1}} > 1-\left(1-\frac{\binom{k}{k_1}}{2.51\cdot 2^{rk}}\right)^{2.51\cdot 2^{rk}/\binom{k}{k_1}} > 1-\frac{1}{e},$$

the algorithm **set-packing**$(\mathcal{C}, k)$ returns a k-packing of the maximum weight in $\mathcal{C}$.

To analyze the complexity of the algorithm, let $T(k, n)$ be the running time of the algorithm **set-packing**$(\mathcal{C}, k)$, where n is the total number of r-sets in the collection $\mathcal{C}$. Clearly we have $T(1, n) = O(rn)$. Moreover, for $k > 1$,

$$\begin{aligned} T(k,n) &= \left(\frac{2.51\cdot 2^{rk}}{\binom{k}{k_1}}\right)\cdot[crn + T(k_1,n) + T(k-k_1,n)] \\ &\leq 14\sqrt{\pi}\sqrt{k}(2^{r-1})^k[crn + T(\lceil k/2\rceil,n) + T(\lfloor k/2\rfloor,n)], \end{aligned}$$

where c is a constant, and based on Stirling's formula [57], we have used the inequality $\binom{k}{k_1} \geq 2^k/(2e\sqrt{\pi k})$. Using the notations in Corollary A.3, here we have $c_0 = 14\sqrt{\pi}$, $k^b = \sqrt{k} = k^{1/2}$, $a = 2^{r-1}$, and $t(n) = crn$. By Corollary A.3, the running time of the algorithm **set-packing**$(\mathcal{C}, k)$ is bounded by

$$T(k,n) = O((2^{r-1})^{2k}k^{\log k/3}rn) = O(4^{(r-1)k}k^{\log k/3}rn).$$

Moreover, since the recursion depth of the algorithm is bounded by $O(\log k)$, it is easy to see that the space complexity of the algorithm is $O(rk\log k + rn)$.

The theorem now follows by an argument similar to that given for Corollary A.5. □

Corollary A.7 *The* PW-r-D MATCHING *problem can be solved (by a randomized algorithm) in time* $O(4^{(r-1)k}k^{\log k/3}rn)$ *and space* $O(rk \log k + rn)$*, where* n *is the number of points in the given instance of the* PW-r-D MATCHING *problem.*

The previously best randomized algorithms for the P-r-D MATCHING and P-r-SET PACKING problems are due to Koutis and have running time $10.88^{rk}n^{O(1)}$ and space complexity $O(2^{rk} + rn)$. Therefore, Theorem A.6 and Corollary A.7 not only give significantly improved running time but also bring the space complexity from exponential down to polynomial.

The techniques used in Theorem A.6 can be used to solve graph packing problems in a very general form. Let H be a fixed graph. A *k-H-packing* of a graph G is a collection of k vertex disjoint subgraphs $\mathcal{P}_k = \{H_1, \ldots, H_k\}$ of G such that each H_i is isomorphic to the graph H. Suppose that there is also a *weight function* f_W from the subgraphs of G to real numbers (that is, for each subgraph G' of G, $f_W(G')$ is a real number that is the *weight* of the subgraph G'). Then we define the *weight* of a k-H-packing $\mathcal{P}_k$ to be the sum of the weights of the subgraphs in $\mathcal{P}_k$. Now we can define a graph packing problem as follows.

> PW-H-GRAPH PACKING: Given a graph G and a parameter k, where there is a weight function f_W from the subgraphs of G to real numbers, either construct a k-H-packing of G that has the maximum weight over all k-H-packings of G, or report that no k-H-packing exists in G.

Suppose that the graph H contains r vertices. Then the PW-H-GRAPH PACKING problem can be reduced to the PW-r-SET PACKING problem, as follows. On the input

graph G, let $\mathcal{C}_G$ be the collection of all subsets V_r of r vertices in G such that the induced subgraph $G[V_r]$ contains the graph H (or more formally, H is isomorphic to a subgraph of $G[V_r]$). For each subset V_r in $\mathcal{C}_G$, define the weight of V_r to be the weight of H', where H' is the subgraph in $G[V_r]$ that is isomorphic to H, and the weight of H' is the maximum over all subgraphs of $G[V_r]$ that are isomorphic to H. It is easy to verify that there is a one-to-one mapping between the k-packings of the maximum weight in the collection $\mathcal{C}_G$ and the k-H-packings of the maximum weight of the graph G. Thus, the PW-H-GRAPH PACKING problem on the graph G can be solved directly by applying the algorithm given in Theorem A.6 to the collection $\mathcal{C}_G$. Furthermore, we can avoid the explicit construction of the collection $\mathcal{C}_G$ and further reduce the complexity of the algorithm. The detailed algorithm is given in Figure 5.

H-**graph packing**(G, k)
input: a graph G, an integer $k \geq 1$, and a weight function f_W
from subgraphs of G to real numbers;
output: a k-H-packing of G that has the maximum weight over all
k-H-packings of G,or $\emptyset$ if there is no k-H-packing of G.

1. **if** $k = 1$ **then**
   if H is isomorphic to a subgraph of G **then**
   let H' be the subgraph of G that is isomorphic to H and has the maximum weight over all subgraphs of G that are isomorphic to H, **return** H';
   else return $\emptyset$;
2. $\mathcal{P}_k = \emptyset$;
3. **loop** $2.51 \cdot 2^{rk} / \binom{k}{\lceil k/2 \rceil}$ times **do**
3.1. randomly partition the vertices of G into two parts V_L and V_R;
3.2. $\mathcal{P}_L = H$-**graph packing**$(G[V_L], \lceil k/2 \rceil)$;
$\mathcal{P}_R = H$-**graph packing**$(G[V_R], \lfloor k/2 \rfloor)$;
3.3. **if** $\mathcal{P}_L \neq \emptyset$, $\mathcal{P}_R \neq \emptyset$, and the sum of the weights of $\mathcal{P}_L$ and $\mathcal{P}_R$ is larger than the weight of $\mathcal{P}_k$ **then**
$\mathcal{P}_k = \mathcal{P}_L \cup \mathcal{P}_R$;
4. **return** $\mathcal{P}_k$.

Fig. 5. A randomized divide-and-conquer algorithm for the PW-H-GRAPH PACKING problem

Theorem A.8 *Let H be a fixed graph of r vertices. Then the algorithm* **H-graph packing** *solves the* PW-H-GRAPH PACKING *problem in time $O(4^{(r-1)k}k^{\log k/3}n^r r^2)$ and space $O(rk\log k + n^2)$ on a graph of n vertices.*

PROOF. The correctness proof and the complexity analysis of the algorithm ***H*-graph packing** are completely similar to that of Theorem A.6, except for the case $k = 1$ in step 1. To find the subgraph H' in G that is isomorphic to H and has the maximum weight, we enumerate all subsets of r vertices in G. For each subset V_r, we consider all possible vertex mappings from H to $G[V_r]$. Note that each isomorphic mapping from H to a subgraph of G is uniquely determined by a subset V_r of r vertices in G and a vertex mapping from H to $G[V_r]$. Therefore, this process enumerates all possible isomorphic mappings from H to subgraphs of G. There are $\binom{n}{r}$ subsets of r vertices in G, and for each subset V_r, there are $r!$ vertex mappings from H to $G[V_r]$. For such a vertex mapping from H to $G[V_r]$, it takes time $O(r^2)$ to check if the mapping induces an isomorphic mapping from H to a subgraph of $G[V_r]$, assuming the graph G being given by an adjacency matrix. In summary, step 1 takes time $O(\binom{n}{r}r^2 r!) = O(n^r r^2)$. That is, we have

$$T(1, n) = O(n^r r^2).$$

The rest of the derivation of the running time of the algorithm follows directly from Corollary A.3.

For space complexity, since the recursion depth is bounded by $O(\log k)$, and all copies of the graph H are disjoint, the space complexity of the algorithm is $O(rk\log k + n^2)$, where we assume that the graph G is given by an adjacency matrix that takes space $O(n^2)$. □

We have finished the introduction of our new randomized algorithms for PW-PATH, PW-MATCHING and PW-PACKING problems. In next section, we will introduce how to use the subset partition technique to solve the PW-SPLITTING problem.

B. The Parameterized Set Splitting Problem

1. Introduction

Let X be a set. A *partition* of X is a pair of subsets (X_1, X_2) of X such that $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \emptyset$. We say that a subset S of X is *split* by the partition (X_1, X_2) of X if $S \cap X_1 \neq \emptyset$ and $S \cap X_2 \neq \emptyset$. The SET SPLITTING problem is defined as follows: given a collection $\mathcal{F}$ of subsets of a ground set X, construct a partition of X that maximizes the number of split subsets in $\mathcal{F}$.

A more generalized version of the SET SPLITTING problem is the *weighted* SET SPLITTING problem, in which each subset in the collection $\mathcal{F}$ is associated with a weight that is a real number, and the objective is to construct a partition of the ground set that maximizes the sum of the weights of the split subsets.

The SET SPLITTING problem is an important NP-hard problem [56]. A number of well-known NP-complete problems are related to the SET SPLITTING problem, including the HITTING SET problem that is to find a small subset of the ground X that intersects all subsets in a given collection $\mathcal{F}$, and the SET PACKING problem that is to find a large sub-collection $\mathcal{F}'$ of a given collection $\mathcal{F}$ of subsets such that the subsets in $\mathcal{F}'$ are all pairwise disjoint.

In terms of approximability, the SET SPLITTING problem is APX-complete [6]. Andersson and Engebretsen [4] gave a polynomial time approximation algorithm for the problem that has an approximation ratio bounded by 0.724. Zhang and Ling [121] presented an improved polynomial time approximation algorithm of approximation

ratio 0.7499 for the problem. Better polynomial time approximation algorithms can be achieved if we further restrict the number of elements in each subset in the input [68, 121, 122, 123].

On certain applications, such as the analysis of micro-array data, people have studied the parameterized version of the SET SPLITTING problem, by associating each instance of the problem with a parameter k, which is in general a small positive integer [37]. The P-SET SPLITTING problem is defined as follows: given a triple $(X, \mathcal{F}, k)$, where X is a ground set, $\mathcal{F}$ is a collection of subsets of the ground set X, and k is the parameter that is a non-negative integer, decide if there is a partition of the ground set X that splits at lease k subsets in $\mathcal{F}$.

In this section, we are mainly concerned with parameterized algorithms for the P-SET SPLITTING problem, i.e. the algorithms run in time $f(k)n^{O(1)}$, with $f(k)$ being a function that only depends on the parameter k.

The P-SPLITTING problem has been studied in the literature. Dehne, Fellows, and Rosamond [37] were the first to study the problem and provided a parameterized algorithm of running time $O^*(72^k)$ for the problem In the same paper, the authors also proved that the P-SET SPLITTING problem has a kernel of fewer than $2k$ subsets: that is, there is a polynomial time algorithm that on a given instance $(X, \mathcal{F}, k)$ of P-SET SPLITTING, produces another instance $(X', \mathcal{F}', k')$ for the problem such that $|X'| \leq |X|$, $|\mathcal{F}'| < 2k$, $k' \leq k$, and that the set X has a partition that splits k subsets in the collection $\mathcal{F}$ if and only if the set X' has a partition that splits k' subsets in the collection $\mathcal{F}'$. Later, Dehne, Fellows, Rosamond, and Shaw [38] developed an improved algorithm of running time $O^*(8^k)$ for the problem. The improved algorithm was obtained by combining the recently developed techniques of *greedy localization* and *modeled crown reduction* in the study of parameterized algorithms. The current best algorithm for the P-SET SPLITTING problem is developed by Lokshtanov and

Sloper [87], where they used Chen and Kanj's result for P-MAX-SAT problem [24] and reached a time complexity of $O^*(2.65^k)$.

A natural generalization of the P-SET SPLITTING problem is the PW-SET SPLITTING problem defined as follows: given a triple $(X, \mathcal{F}, k)$, where X is a ground set, $\mathcal{F}$ is a collection of subsets of the ground set X, in which each subset is assigned a weight (that is a real number), and k is the parameter that is a non-negative integer, either construct a partition of X that maximizes the weighted sum of k split subsets in $\mathcal{F}$, or report that no partition of X can split k subsets in $\mathcal{F}$. Note that there is an essential difference between P-SET SPLITTING and PW-SET SPLITTING. P-SET SPLITTING is a decision problem that only requires a yes/no answer, while PW-SET SPLITTING is an optimization problem that, in case a partition of the ground set X splitting k subsets in $\mathcal{F}$ exists, requires to construct such a partition that maximizes the weighted sum of the split subsets.

No parameterized algorithms of running time of the form $f(k)n^{O(1)}$ have been known for the PW-SET SPLITTING problem. In fact, none of the techniques developed previously for the P-SET SPLITTING problem, such as those in [37, 38, 87], seems to be extendable to the weighted case.

In this section, we develop new techniques in dealing with the P-SET SPLITTING and the PW-SET SPLITTING problems. First, we develop a new and effective technique based on a probabilistic method that allows us to develop a *deterministic* kernelization algorithm for the P-SET SPLITTING problem. The new kernelization algorithm is simpler and more efficient compared with the previous kernelization algorithm given in [37]. We then propose a randomized algorithm for the PW-SET SPLITTING problem (thus, also for the P-SET SPLITTING problem) that is based on a new subset partition technique and has its running time bounded by $O^*(2^k)$. The running time of our randomized algorithm is significantly better than that of the previous best deterministic

algorithm of running time $O^*(2.65^k)$ given in [87], which only works for the (simpler) P-SET SPLITTING problem. We will show in section D that, using the structure of (n, k)-universal sets developed by Naor, Schulman, and Srinivasan [93], we can derandomize our randomized algorithm, which leads to a parameterized algorithm of running time $O^*(4^{k+o(k)})$ for the PW-SET SPLITTING problem and gives the first proof that the problem is fixed parameter tractable.

2. A new kernelization algorithm for the P-SET SPLITTING

In this subsection, we focus on the P-SET SPLITTING problem. By a *kernelization algorithm* for P-SET SPLITTING, we mean a polynomial time algorithm that, on an instance $(X, \mathcal{F}, k)$ of P-SET SPLITTING, produces another instance $(X', \mathcal{F}', k')$ for the problem such that $k' \leq k$ and the size of the instance $(X', \mathcal{F}', k')$ only depends on the parameter k. The instance $(X', \mathcal{F}', k')$ will be called a *kernel* for the instance $(X, \mathcal{F}, k)$. Dehne, Fellows, and Rosamond [37] developed a kernelization algorithm by which the kernel $(X', \mathcal{F}', k')$ satisfies the conditions $|\mathcal{F}'| < 2k$ and that each subset in $\mathcal{F}'$ has at most $2k$ elements. Lokshtanov and Sloper [87] used the crown decomposition method to obtain a kernel such that both $|\mathcal{F}'|$ and $|X'|$ are less than $2k$. We introduce a new method to find the kernel for the P-SET SPLITTING problem. What is interesting in our method is that we use a probabilistic method to derive a deterministic kernelization algorithm. In particular, our method is simpler, has lower time complexity, and can also obtain a better kernel in term of the number of subsets in $\mathcal{F}'$ if there are subsets in $\mathcal{F}'$ whose size is larger than 2.

Lemma B.1 *Given an instance $(X, \mathcal{F}, k)$ of the* P-SET SPLITTING *problem, let m_1 be the number of subsets in $\mathcal{F}$ that have only one element. If $|\mathcal{F}| - m_1 \geq 2k$, then a partition of X exists that splits at least k subsets in $\mathcal{F}$.*

PROOF. For each subset $S \in \mathcal{F}$, if S has at least two elements, we pick any two elements from S. Let V be the set of all these elements picked from the subsets in $\mathcal{F}$ that have more than one element. Note that for two subsets S_1 and S_2 in $\mathcal{F}$ that have more than one element, the two elements in S_1 and the two elements in S_2 may not be disjoint.

Suppose $|V| = t$. We randomly partition V into two subsets V_l and V_r, such that $|V_l| = \lfloor t/2 \rfloor$, $|V_r| = t - |V_l|$, i.e. we randomly pick $\lfloor t/2 \rfloor$ elements of V and put them in V_l and let the remaining $t - \lfloor t/2 \rfloor$ elements of V be in V_r. Thus, for any subset S in $\mathcal{F}$:

$$Pr(\text{S is split}) \begin{cases} \geq \frac{2\binom{t-2}{\lfloor t/2 \rfloor - 1}}{\binom{t}{\lfloor t/2 \rfloor}} = \frac{2\lfloor t/2 \rfloor (t - \lfloor t/2 \rfloor)}{t(t-1)} > \frac{1}{2}, & \text{if } S \text{ has more than one element} \\ = 0, & \text{otherwise.} \end{cases}$$

If we let:

$$X_S = \begin{cases} 1, & \text{if } S \text{ is split,} \\ 0, & \text{otherwise,} \end{cases}$$

then the expectation of the number of split subsets in $\mathcal{F}$ satisfies

$$E\left(\sum_{S \in \mathcal{F}} X_S\right) \geq \frac{1}{2}(|\mathcal{F}| - m_1),$$

Therefore, if $|\mathcal{F}| - m_1 \geq 2k$, then the expectation of the number of split subsets in $\mathcal{F}$ is larger than or equal to k. That is, there must exist a partition of the ground set X such that the number of split subsets in $\mathcal{F}$ is at least k. This completes the proof of the lemma. □

The result of Lemma B.1 was first observed by Lokshtanov and Sloper, who presented a proof in [87]. Our proof above is very different from that given in [87] and takes a probabilistic approach. Furthermore this new approach can lead to a better

result for the kernelization when many subsets in $\mathcal{F}$ have more than 2 elements, as described in Lemma B.4.

The following lemma shows that we can directly include subsets of at least k elements in our split subsets while we are solving the P-SET SPLITTING problem.

Lemma B.2 *Let $(X, \mathcal{F}, k)$ be an instance of the* P-SET SPLITTING *problem, and let S be a subset in $\mathcal{F}$ that contains at least k elements. Then there is a partition of X that splits k subsets in $\mathcal{F}$ if and only if there is a partition of X that splits $k-1$ subsets in $\mathcal{F} - \{S\}$.*

PROOF. Suppose that there is a partition (X_l, X_r) of the ground set X that splits k subsets in $\mathcal{F}$. Then it is obvious that (X_l, X_r) splits (at least) $k-1$ subsets in $\mathcal{F} - \{S\}$.

On the other hand, suppose that there is a partition (X_l, X_r) of the ground set X that splits $k-1$ subsets S_1, ..., S_{k-1} in $\mathcal{F} - \{S\}$. Let $l_i, r_i \in S_i$, $l_i \in X_l$, and $r_i \in X_r$, for all $1 \leq i \leq k-1$. Since S has at least k elements, there are at least two different elements l and r in S such that $l \notin \{r_1, \ldots, r_{k-1}\}$ and $r \notin \{l_1, \ldots, l_{k-1}\}$. Therefore, if we modify the partition (X_l, X_r) to enforce l in X_l and r in X_r (note that this modification still keeps l_i in X_l and r_i in X_r for all $1 \leq i \leq k-1$), then the new partition of X splits the subset S, as well as the $k-1$ subsets S_1, ..., S_{k-1} in $\mathcal{F} - \{S\}$. In consequence, the new partition of the ground set X splits (at least) k subsets in the collection $\mathcal{F}$. □

Now we are ready to state our first kernelization result. For a given instance $(X, \mathcal{F}, k)$ of the P-SET SPLITTING problem, consider the following reduction rules.

Rule R1. If a subset S in $\mathcal{F}$ has only one element, remove S from $\mathcal{F}$.

Rule R2. If a subset S in $\mathcal{F}$ has at lease k elements, remove S from $\mathcal{F}$ and decrease k by 1.

The correctness of Rule **R1** is obvious: a subset of a single element can never be split by any partition of the ground set X. The correctness of Rule **R2** follows from Lemma B.2.

Theorem B.3 *Given an instance $(X, \mathcal{F}, k)$ of the* P-SET SPLITTING *problem, we can construct a kernel $(X_1, \mathcal{F}_1, k_1)$ such that $|\mathcal{F}_1| < 2k_1$, $k_1 \leq k$, $|X_1| < 2k_1^2$, and that each subset in $\mathcal{F}_1$ has at most $k_1 - 1$ elements. The running time of this process is bounded by $O(N)$, where N is the input size in terms of $(X, \mathcal{F}, k)$.*

PROOF. For the given instance $(X, \mathcal{F}, k)$, we first apply Rule **R1** to remove all subsets that contain a single element. Then, we apply bucket-sort to sort in linear time the remaining subsets in $\mathcal{F}$ in non-increasing order in terms of their sizes. Finally, we apply Rule **R2** on the subsets in order in the sorted list, and stop at a subset on which Rule **R2** is not applicable or when the parameter value k reaches 0. This process obviously takes time $O(N)$.

Suppose that the instance produced by the above process is $(X_1, \mathcal{F}_1, k_1)$. If $k_1 = 0$ or $|\mathcal{F}_1| \geq 2k_1$, then by Lemma B.1 (note that $\mathcal{F}_1$ contains no subsets of one element), $(X_1, \mathcal{F}_1, k_1)$ (as well as the original instance $(X, \mathcal{F}, k)$) is a "Yes" instance. In this case, our algorithm returns a trivial "Yes" instance $(\{a, b\}, \{\{a, b\}\}, 1)$. Otherwise, the correctness of the reduction rules **R1** and **R2** ensure that $(X_1, \mathcal{F}_1, k_1)$ is a "Yes" instance if and only if $(X, \mathcal{F}, k)$ is a "Yes" instance for the P-SET SPLITTING problem. So our algorithm simply returns $(X_1, \mathcal{F}_1, k_1)$.

To see that the instance $(X_1, \mathcal{F}_1, k_1)$ satisfies the conditions in the lemma, first note that if a non-trivial instance is returned by the process, then we must have

$|\mathcal{F}_1| < 2k_1$. Moreover, since the subsets in $\mathcal{F}_1$ are sorted in non-increasing order in terms of their sizes, and Rule **R2** is not applicable to the first subset in the list, no subset in $\mathcal{F}_1$ contains more than $k_1 - 1$ elements. In consequence, we also have $|X_1| < 2k_1^2$. This completes the proof of the lemma. □

Theorem B.3 improves the time complexity of the kernelization algorithm given in [37], which takes time $O(N + n^4)$, as well as that given in [87], which takes time $O(N + n^2)$, where n is the maximum of $|\mathcal{F}|$ and $|X|$.

Intuitively, when we randomly partition X into $X_l \cup X_r$ such that each element in X has a probability of $1/2$ to be assigned to X_l and a probability of $1/2$ to be assigned to X_r, larger subsets (i.e., subsets with more elements) will have a better chance to be split. The following lemma confirms this intuition. Thus, if the collection $\mathcal{F}$ contains many large subsets, then we can obtain a better kernel, or a kernel with fewer subsets.

Lemma B.4 *Let $(X, \mathcal{F}, k)$ be an instance of the* P-SET SPLITTING *problem. Suppose that the number of subsets of i elements in $\mathcal{F}$ is m_i for $1 \leq i \leq k-1$, and that the number of subsets that have at least k elements is m'_k. If $\sum_{i=2}^{k-1} \frac{2^i-2}{2^i} m_i + m'_k \geq k$, then a partition of X exists that splits at least k subsets in $\mathcal{F}$.*

PROOF. Let $S_1, \cdots, S_{m'_k}$ be the subsets in $\mathcal{F}$ that have at least k elements and let $\mathcal{F}_{<k} = \mathcal{F} - \{S_1, \cdots, S_{m'_k}\}$.

We use a randomized process to partition X into (X_l, X_r) and let each element in X go to X_l with a probability of $1/2$ and go to X_r with a probability of $1/2$, then for any subset $S \in \mathcal{F}_{<k}$ that has i elements:

$$Pr(S \text{ is split}) = \frac{2^i - 2}{2^i}.$$

If we let:

$$X_S = \begin{cases} 1, \text{ if } S \text{ is split,} \\ 0, \text{ otherwise.} \end{cases}$$

then the expectation of the number of split subsets in $\mathcal{F}_{<k}$ satisfies

$$E\left(\sum_{S\in\mathcal{F}_{<k}} X_S\right) = \sum_{i=1}^{k-1}\sum_{|S|=i} E(S \text{ is split}) = \sum_{i=1}^{k-1} \frac{2^i-2}{2^i} m_i.$$

So there exists a partition of X such that the number of subsets in $\mathcal{F}_{<k}$ that are split is at least $\sum_{i=1}^{k-1} \frac{2^i-2}{2^i} m_i$. Hence if $\sum_{i=1}^{k-1} \frac{2^i-2}{2^i} m_i \geq k - m'_k$, there must exist a partition of X such that $k-m'_k$ subsets in $\mathcal{F}_{<k}$ are split. By repeatedly using Lemma B.2, there is a partition of X that splits $k - m'_k + 1$ subsets in $\mathcal{F}_{<k} \cup \{S_1\}$; there is a partition of X that splits $k - m'_k + 2$ subsets in $\mathcal{F}_{<k} \cup \{S_1, S_2\}$; and so on. In conclusion, there is a partition of X that splits k subsets in $\mathcal{F}_{<k} \cup \{S_1, \cdots, S_{m'_k}\} = \mathcal{F}$. □

Using the procedure that is similar to Theorem B.3, but counting the number of subsets in $\mathcal{F}$ of different size and using the result of Lemma B.4, we have the following theorem that is stronger than Theorem B.3.

Theorem B.5 *Given an instance $(X, \mathcal{F}, k)$ of the* P-SET SPLITTING *problem, we can find a kernel $(X_1, \mathcal{F}_1, k_1)$ in time $O(N)$ such that $|\mathcal{F}_1| < 2k_1 - \sum_{i=3}^{k_1-1} \frac{2^{i-1}-2}{2^{i-1}} m_i$, that $k_1 \leq k$, that each subset in $\mathcal{F}_1$ has at most $k_1 - 1$ elements, and that $|X_1| < 2k_1^2$, where N is the input size in terms of $(X, \mathcal{F}, k)$, and m_i is the number of subsets of i elements in $\mathcal{F}_1$, $1 \leq i \leq k_1 - 1$.*

PROOF. We use the same procedure as the one presented in Theorem B.3 to find the kernel. Let the resulting instance be $(X_1, \mathcal{F}_1, k_1)$. We also calculate the values m_i from $\mathcal{F}_1$, for $1 \leq i \leq k_1 - 1$. If $\sum_{i=2}^{k_1-1} \frac{2^i-2}{2^i} m_i \geq k_1$ (note that no subset in $\mathcal{F}_1$ contains more than $k_1 - 1$ elements), then by Lemma B.4, the instance $(X_1, \mathcal{F}_1, k_1)$ is a "Yes"

instance so we return a trivial "Yes" instance. Otherwise, we have $\sum_{i=2}^{k_1-1} \frac{2^i-2}{2^i} m_i < k_1$, which gives $|\mathcal{F}_1| = \sum_{i=2}^{k_1-1} m_i < 2k_1 - \sum_{i=3}^{k_1-1} \frac{2^{i-1}-2}{2^{i-1}} m_i$. In this case, we obtain a kernel $(X_1, \mathcal{F}_1, k_1)$ such that $|\mathcal{F}_1| < 2k_1 - \sum_{i=3}^{k_1-1} \frac{2^{i-1}-2}{2^{i-1}} m_i$, that $k_1 \leq k$, that each subset in $\mathcal{F}_1$ has at most $k_1 - 1$ elements, and that $|X_1| < 2k_1^2$. □

3. A randomized algorithm for the PW-SET SPLITTING

For the P-SET SPLITTING problem, Lokshtanov and Sloper [87] have currently the best parameterized algorithm, whose time complexity is bounded by $O^*(2.65^k)$. Unfortunately, their method does not seem to be extendable to the weighted case, neither do the methods presented in [37, 38] for the unweighted case. In fact, no previous work is known that gives a parameterized algorithm of running time of the form $f(k)n^{O(1)}$ for the PW-SET SPLITTING problem.

In this section, we present a randomized algorithm to solve the PW-SET SPLITTING problem. Our basic idea is that if a given instance $(X, \mathcal{F}, k)$ of the PW-SET SPLITTING problem has a partition of the ground set X that splits k subsets in the collection $\mathcal{F}$, then there exists a subset X' of at most $2k$ elements in X such that a proper partition of the elements in X' can split at least k subsets in $\mathcal{F}$. If we use a randomized process to partition X into (X_l, X_r) and let each element in X go to X_l with a probability of $1/2$ and go to X_r with a probability of $1/2$, then the probability that the elements in X' are partitioned properly is at least $2/2^{2k}$. Thus, if we try $O(4^k)$ times of the randomized partitioning of the ground set X, we have a good chance to find the proper partition of X' if it exists. In fact, a more thorough analysis reveals that only $O(2^k)$ trials are needed in this randomized algorithm.

Theorem B.6 *The* PW-SET SPLITTING *problem can be solved by a randomized algorithm of running time $O(2^k N)$, where N is the input size in terms of $(X, \mathcal{F}, k)$.*

Algorithm-1 SetSplitting$(X, \mathcal{F}, k)$
input: A ground set X, a collection $\mathcal{F}$ of subsets of X, and an integer k
output: A partition (X_l, X_r) of X and k subsets in $\mathcal{F}$ that are split by (X_l, X_r), or report "no partition of X splits k subsets in $\mathcal{F}$".

1. $\mathcal{Q}_0 = \emptyset$;
2. **for** $i = 1$ **to** $10 \cdot 2^k$ **do**
2.1. randomly partition X into X_l and X_r such that each element in X has a probability $1/2$ in X_l and a probability $1/2$ in X_r;
2.2. let $\mathcal{Q}$ be the collection of subsets in $\mathcal{F}$ that are split by (X_l, X_r);
2.3. **if** $\mathcal{Q}$ contains at least k subsets **then**
delete all but the k subsets of maximum weight in $\mathcal{Q}$;
2.4. **if** the weighted sum of subsets in $\mathcal{Q}$ is larger than that in $\mathcal{Q}_0$ **then** $\mathcal{Q}_0 = \mathcal{Q}$;
3. **return** $\mathcal{Q}_0$.

Fig. 6. Randomized algorithm for the PW-SET SPLITTING problem

PROOF. Let $(X, \mathcal{F}, k)$ be an instance of the PW-SET SPLITTING problem. Suppose that there is a partition of the ground set X that splits at least k subsets in the collection $\mathcal{F}$. Let (X_l, X_r) be a partition of the ground set X and let S_1, ..., S_k be k subsets in the collection $\mathcal{F}$ that are split by the partition (X_l, X_r), such that the weighted sum of S_1, ..., S_k is the maximum over all collections of k subsets in $\mathcal{F}$ that can be split by a partition of X. More specifically, let (l_1, r_1), ..., (l_k, r_k) be k pairs of elements in the ground set X such that $l_i, r_i \in S_i$, $l_i \in X_l$, and $r_i \in X_r$ for all $1 \leq i \leq k$. Note that it is possible that $l_i = l_j$ or $r_i = r_j$ for some $i \neq j$. In consequence, each of the sets $\{l_1, \ldots, l_k\}$ and $\{r_1, \ldots, r_k\}$ may contain fewer than k elements.

We construct a graph $G = (V, E)$, where $V = \{l_1, l_2, \cdots, l_k\} \cup \{r_1, r_2, \cdots, r_k\}$ and $E = \{[l_i, r_i] \mid 1 \leq i \leq k\}$. It is obvious that G is a bipartite graph with the left vertex set $L = \{l_1, l_2, \cdots, l_k\}$ and the right vertex set $R = \{r_1, r_2, \cdots, r_k\}$. Suppose that the graph G has t connected components C_1, $\cdots$, C_t, where $C_i = (V_i, E_i)$, with $n_i = |V_i|$

and $m_i = |E_i|$, for $1 \leq i \leq t$. Then $n_i \leq m_i + 1$ for $1 \leq i \leq t$ and $\sum_{i=1}^{t} m_i = k$. If we use a randomized process to partition X into (X_l, X_r) and let each element in X go to X_l with a probability of $1/2$ and go to X_r with a probability of $1/2$, then for each connected component C_i of the graph G, the probability that the vertex set V_i of C_i is properly partitioned, i.e., either $L \cap V_i \subseteq X_l$ and $R \cap V_i \subseteq X_R$, or $R \cap V_i \subseteq X_l$ and $L \cap V_i \subseteq X_R$, is $2/2^{n_i}$. Therefore, the total probability that the vertex set V_i for every connected component C_i is properly partitioned, i.e., that the pair (l_i, r_i) intersects with both X_l and X_r for all $1 \leq i \leq k$, is not less than

$$\frac{2}{2^{n_1}} \cdot \frac{2}{2^{n_2}} \cdot \cdots \cdot \frac{2}{2^{n_t}} \geq \frac{2}{2^{m_1+1}} \cdot \frac{2}{2^{m_2+1}} \cdot \cdots \cdot \frac{2}{2^{m_t+1}} = \frac{2^t}{2^{\sum_{i=1}^{t} m_i + t}} = \frac{1}{2^k}.$$

The algorithm in Figure 6 implements the above idea. By the above discussion, each random partition (X_l, X_r) constructed in step 2.1 has a probability of at least $1/2^k$ to split the k subsets S_1, ..., S_k (recall that S_1, ..., S_k are the k subsets in $\mathcal{F}$ whose weighted sum is the maximum over all collections of k subsets in $\mathcal{F}$ that are split by a partition of X). Since step 2 loops $10 \cdot 2^k$ times, with a probability of at least

$$1 - \left(1 - \frac{1}{2^k}\right)^{10 \cdot 2^k} \geq 99.99\%,$$

one partition (X_l, X_r) constructed by step 2.1 splits the k subsets S_1, ..., S_k. For this partition (X_l, X_r), steps 2.2-2.4 produces a collection $\mathcal{Q}$ of k subsets in $\mathcal{F}$ whose weighted sum is the maximum over all collections of k subsets in $\mathcal{F}$ that can be split by a partition of the ground set X.

Since each execution of steps 2.1-2.4 obviously takes time $O(N)$, we conclude that the running time of the algorithm **SetSplitting** is bounded by $O(2^k N)$.

For a general error bound $\epsilon > 0$, we can simply run the algorithm **SetSplitting** c times, where the constant c satisfies the condition $(1 - 0.9999)^c \leq \epsilon$, which will

produce, in time $O(2^k N)$ and with a probability at least $1 - \epsilon$, a collection $\mathcal{Q}$ of k subsets in $\mathcal{F}$ whose weighted sum is the maximum over all collections of k subsets in $\mathcal{F}$ that can be split by a partition of the ground set X. □

Obviously, the randomized algorithm **SetSplitting** of running time $O(2^k N)$ can be directly used to solve the simpler P-SET SPLITTING problem, and its running time is significantly better than that of the previous best deterministic algorithm [87] for the problem. Moreover, the algorithm **SetSplitting** is much simpler than the one presented in [87]. The algorithm in [87] needs to call the algorithm for the parameterized P-MAX-SAT problem developed in [24], which is quite involved.

By combining the kernelization algorithm, the time complexity for the P-SET SPLITTING problem can be further improved.

Theorem B.7 *The* P-SET SPLITTING *problem can be solved by a randomized algorithm of running time $O(2^k k^2 + N)$, where N is the input size in terms of $(X, \mathcal{F}, k)$.*

The introduction of our new randomized algorithm for the PW-SET SPLITTING problem is finished. In next section, we will introduce how to construct the (n, k)-universal set developed by Noar, Schulman, and Srinivasan. In section D, we will use this (n, k)-universal set to derandomized our randomized algorithms for the PW-PATH, PW-MATCHING, PW-PACKING and PW-SET SPLITTING problems.

C. (n, k)-universal Sets

Naor, Schulman, and Srinivasan developed a deterministic construction of (n, k)-universal sets [93]. The construction was described via the construction of a more general structure, i.e., (n, k, l)-splitters. Moreover, the construction was presented in an extended abstract [93] in which many details were omitted. For the completeness

of our discussion, we will reproduce in this section a slightly different and simpler construction and its analysis specifically for (n, k)-universal sets. The presentation here is more precise with all needed details provided. These (n, k)-universal sets will be used to derandomized our algorithms in sections A and B for the PW-PATH, PW-r-D MATCHING, PW-r-SET PACKING, and PW-SET SPLITTING problems.

We start with some terminologies and definitions in probability theory.

Let $(\Omega, \Pr)$ be a probability space, where Ω is a finite set and Pr is the probability measure. The *size* of $(\Omega, \Pr)$ is the number of elements in Ω. The probability space $(\Omega, \Pr)$ is *uniform* if $\Pr(a) = 1/|\Omega|$ for all $a \in \Omega$ (in this case, we will simply write the probability space as Ω).

A $\{0, 1\}$-*random variable* ξ over the probability space $(\Omega, \Pr)$ is a function from Ω to $\{0, 1\}$. A group of h $\{0, 1\}$-random variables ξ_1, ξ_2, ..., ξ_h are *mutually independent* if for any combination of h binary bits b_1, ..., b_h in $\{0, 1\}$, the following holds:

$$\Pr(\xi_1 = b_1, \xi_2 = b_2, \ldots, \xi_h = b_h) = \Pr(\xi_1 = b_1)\Pr(\xi_2 = b_2) \cdots \Pr(\xi_h = b_h).$$

A group of n $\{0, 1\}$-random variables ξ_1, ξ_2, ..., ξ_n are *k-wise independent* if every group of k different $\{0, 1\}$-random variables among ξ_1, ξ_2, ..., ξ_n are mutually independent.

Assume that n and k are integers such that $n \geq k$. Denote by Z_n the set $\{0, 1, \ldots, n-1\}$. A *splitting function* over Z_n is a $\{0, 1\}$ (i.e., Boolean) function over Z_n. A subset S of Z_n is a *k-subset* if S consists of exactly k elements. Let (S_0, S_1) be a partition of the k-subset S, i.e., $S_0 \cup S_1 = S$ and $S_0 \cap S_1 = \emptyset$. We say that a splitting function f over Z_n *implements* the partition (S_0, S_1) if $f(x) = 0$ for all $x \in S_0$ and $f(y) = 1$ for all $y \in S_1$.

Definition [93] A set Ψ of splitting functions over Z_n is an (n, k)-*universal set* if for

every k-subset S of Z_n and any partition (S_0, S_1) of S, there is a splitting function f in Ψ that implements (S_0, S_1). The *size* of an (n, k)-universal set Ψ is the number of splitting functions in Ψ.

Before we present the (n, k)-universal set, a proposition need to be introduced.

A function f on Z_n is *injective from a subset S of Z_n* if for any two different elements x and y in S, $f(x) \neq f(y)$.

By Bertrand's postulate, proved by Chebyshev in 1850 (see [113], Section 5.2), there is a prime number q such that $n \leq q < 2n$. Moreover, the smallest prime number q_0 between n and $2n$ can be constructed in time $O(n)$, as follows. By the Prime Number Theorem (see [113], Theorem 5.19), there is a constant d_0 such that for any integer $n \geq 2$, there is a prime number between n and $n + d_0 \log n$. Therefore, by checking each of the integers between n and $n + d_0 \log n$, and testing its primality using the trivial primality testing algorithm of running time $O(\sqrt{n})$, we can always find the smallest prime number q_0 between n and $2n$ in time $O(\sqrt{n} \log n) = O(n)$.

Proposition C.1 [54] *Let n and k be integers, $n \geq k$, and let q_0 be the smallest prime number such that $n \leq q_0 < 2n$. For any k-subset S in Z_n, there is an integer z, $0 \leq z < q_0$, such that the function $\psi_{z,k^2,n}$ over Z_n, defined as $\psi_{z,k^2,n}(x) = (zx \bmod q_0) \bmod k^2$, is injective from S.*

The following lemma is crucial to our construction, and was first proved in [1].

Lemma C.2 [1] *Let $n = 2^d - 1$ for an integer d and let k be an odd number, $k \leq n$. There is an algorithm of running time $O(n(n+1)^{(k-1)/2})$ that constructs a uniform probability space Ω of size $2(n+1)^{(k-1)/2}$ and a group of n k-wise independent $\{0, 1\}$-random variables $\xi_1, \ldots, \xi_n$ over Ω such that $Pr(\xi_i = 0) = Pr(\xi_i = 1) = 1/2$ for all $1 \leq i \leq n$.*

We start with a simple observation.

Lemma C.3 *For every integer $n \geq 1$, there is an $(n, 1)$-universal set of size 2.*

PROOF. The splitting functions $f_0 \equiv 0$ and $f_1 \equiv 1$ over Z_n obviously form an $(n, 1)$-universal set for Z_n. □

Now we present a construction of a general (n, k)-universal set of small size that, however, is not sufficiently efficient. Compared to the work presented in [93], the size of our structure is more precise (and slightly improved), which will be important for the later construction. Moreover, the time complexity of our construction is lower than that described in [93].

Lemma C.4 *Let k be an odd number, $n \geq k$ and $n \geq 2$. There is an (n, k)-universal set of size bounded by $2^k k \log n$, which can be constructed in time $O(\binom{n}{k} 2^k k (2n)^{(k-1)/2})$.*

PROOF. The lemma is true for $k = 1$ by Lemma C.3. Thus, we assume $k \geq 3$.

Let $n_1 = 2^d - 1$, where d is the smallest integer such that $n \leq n_1$ (note that $n \leq n_1 \leq 2n-1$). By Lemma C.2, we can construct, in time $O(n_1(n_1+1)^{(k-1)/2})$, a uniform probability space Ω of size $2(n_1+1)^{(k-1)/2}$ and a group of n_1 k-wise independent $\{0, 1\}$-random variables $\xi_1, \ldots, \xi_{n_1}$ over Ω such that $\Pr(\xi_i = 0) = \Pr(\xi_i = 1) = 1/2$ for all $1 \leq i \leq n_1$. By picking the first n of these n_1 random variables, we get a group of n k-wise independent $\{0, 1\}$-random variables ξ_1, ..., ξ_n over the uniform probability space Ω such that $\Pr(\xi_i = 0) = \Pr(\xi_i = 1) = 1/2$ for all $1 \leq i \leq n$. All these can be constructed in time $O(n(2n)^{(k-1)/2})$.

Note that the uniform probability space Ω and the random variables ξ_1, ..., ξ_n constructed above actually make a collection $\mathcal{P}$ of $D = 2(n_1 + 1)^{(k-1)/2}$ splitting functions over Z_n. In fact, for each element a in Ω, the values of the random variables

$\xi_1, \ldots, \xi_n$ make a binary string $\xi_1(a) \cdots \xi_n(a)$ of length n, which can be interpreted as a splitting function over Z_n.

Construct a bipartite graph $G = (V_1 \cup V_2, E)$ with the vertex bipartition (V_1, V_2), where V_1 consists of D vertices, corresponding to the D splitting functions in the collection $\mathcal{P}$, and the vertex set V_2 consists of $D' = \binom{n}{k}2^k$ vertices such that for each k-subset S of Z_n and each partition (S_1, S_2) of S, there is a corresponding vertex in V_2. An edge $[v, w]$ is created in G if the splitting function corresponding to the vertex $v \in V_1$ implements the partition (S_1, S_2) of a k-subset S of Z_n that correspond to the vertex $w \in V_2$.

Claim. Each vertex in V_2 has a degree $D/2^k$.

Proof of the Claim. Let $S = \{h_1, \ldots, h_k\}$ be any k-subset of Z_n and let (S_1, S_2) be a partition of S. Define k binary bits b_{h_i}, $1 \leq i \leq k$ such that $b_{h_i} = 0$ if $h_i \in S_1$ and $b_{h_i} = 1$ if $h_i \in S_2$. Consider the k mutually independent random variables ξ_{h_1}, $\ldots$, ξ_{h_k} (they are mutually independent because the random variables $\xi_1, \ldots, \xi_n$ are k-wise independent), we have

$$\Pr(\xi_{h_1} = b_{h_1}, \ldots, \xi_{h_k} = b_{h_k}) = \Pr(\xi_{h_1} = b_{h_1}) \cdots \Pr(\xi_{h_k} = b_{h_k}) = 1/2^k.$$

Thus, there are $D/2^k$ elements a in Ω such that $\xi_{h_i}(a) = b_{h_i}$ for $1 \leq i \leq k$. By the interpretation above, there are $D/2^k$ splitting functions in the collection $\mathcal{P}$ that implement the partition (S_1, S_2) of the k-subset S. By our construction of the graph G, the vertex w in V_2 corresponding to the partition (S_1, S_2) of the k-subset S has degree $D/2^k$. The claim now is proved because S is an arbitrary k-subset in Z_n and (S_1, S_2) is an arbitrary partition of S. End of the Claim

Since there are D' vertices in V_2, the above claim shows that the bipartite graph

G contains exactly $(D'D)/2^k$ edges. Now since there are D vertices in V_1, there is a vertex v_1 in V_1 in the graph G whose degree is at least $D'/2^k$. In other words, there is a splitting function in the collection $\mathcal{P}$ that implements at least $D'/2^k$ partitions of k-subsets of Z_n (these partitions can be partitions for different k-subsets of Z_n).

We perform the following operations on the graph G: mark the vertex v_1 in V_1 and remove all vertices in V_2 that are adjacent to v_1 (or, equivalently, we mark a splitting function f in $\mathcal{P}$ and remove all partitions of k-subsets of Z_n that are implemented by f). Let D'_1 be the number of vertices in V_2 in the remaining bipartite graph G'. $D'_1 \leq (1 - 1/2^k)D'$.

Note that each vertex in V_2 in the remaining graph G' still has degree $D/2^k$. Therefore, by repeating the above process, in the remaining graph G', we can find a vertex v_2 in V_1 that is adjacent to at least $D'_1/2^k$ vertices in V_2. Now we mark v_2, and remove the vertices in V_2 that are adjacent to v_2. Now there are at most $D'_2 \leq (1 - 1/2^k)D'_1 \leq (1 - 1/2^k)^2 D'$ vertices in V_2 in the remaining graph.

Repeat the above process until all vertices in V_2 are removed. The number t' of times the above process is repeated is not larger than the smallest integer t such that $(1 - 1/2^k)^t D' < 1$. For $k = 3$, we can directly verify that $t' \leq 2^k k \log n$; and for $k \geq 5$, since $D' = \binom{n}{k} 2^k \leq n^k$ and $(1 - 1/2^k)^{2^k} < 1/e$, we can also formally prove that $t' \leq 2^k k \log n$.

Each execution of the above process marks a vertex in V_1, therefore, there are at most $2^k k \log n$ vertices in V_1 that are marked in the above process. By our construction, all vertices in the set V_2 are adjacent to at least one marked vertex in V_1. Accordingly, there are $D'' \leq 2^k k \log n$ splitting functions in the collection $\mathcal{P}$ such that every partition of any k-subset in Z_n is implemented by at least one of these D'' splitting functions. That is, these D'' splitting functions make an (n, k)-universal set $\mathcal{P}'$ of size bounded by $2^k k \log n$.

Now we analyze the time complexity for the entire construction of the (n, k)-universal set $\mathcal{P}'$. As given earlier, the construction of the uniform probability space Ω and the $\{0, 1\}$-random variables $\xi_1, \ldots, \xi_n$ takes time $O(n(2n)^{(k-1)/2})$. The construction of the bipartite graph G takes time $O(k|V_1||V_2|) = O(kDD') = O(\binom{n}{k}k2^k (2n)^{(k-1)/2})$. To perform the above iteration process of marking vertices in V_1, we can represent the bipartite graph G as a $D \times D'$ matrix, and keep an array for the degrees of the vertices in V_1. It is not difficult to verify that on such data structures, the entire vertex marking process takes time $O(t'(D + D') + DD') = O(\binom{n}{k}2^k(2n)^{(k-1)/2})$. Thus, the total construction of the (n, k)-universal set $\mathcal{P}'$ takes time $O(\binom{n}{k}2^k k(2n)^{(k-1)/2})$.

□

The size of the (n, k)-universal set constructed in Lemma C.4 is quite small. Unfortunately, the time complexity for constructing such an (n, k)-universal set given in the lemma is unacceptably high. Therefore, we need to use additional techniques to reduce the construction time.

Fix n and k, where $n \geq k$. At the moment, we assume $k \geq 2$. Define

$$\begin{aligned}
&k_1 = \text{ the largest odd number bounded by } k/(4\log k),\\
&t = \lceil k/k_1 \rceil, \qquad \text{(it is not hard to verify that } t \leq 4\log k + 2),\\
&k_2 = k - k_1(t-1), \qquad \text{(note that } k_2 \leq k_1),\\
&n_1 = k^2, \text{ and}\\
&p = \text{ a prime number such that } n \leq p < 2n,
\end{aligned} \tag{4.4}$$

where the existence of the prime number p above is guaranteed by Bertrand's postulate [113].

Consider the set $Z_{k^2} = \{0, 1, \ldots, k^2 - 1\}$. Pick any $t - 1$ elements $i_2, i_3, \ldots, i_t$ in Z_{k^2}, such that $i_2 < i_3 < \cdots < i_t$. These $t - 1$ elements naturally divide the set Z_{k^2}

into t sets consisting of consecutive elements (where X_1 may be an empty set):

$$X_1 = \{0, \ldots, i_2 - 1\}, \quad X_2 = \{i_2, \ldots, i_3 - 1\}, \quad \ldots, \quad X_t = \{i_t, \ldots, k^2 - 1\}.$$

Such a division $(X_1, X_2, \ldots, X_t)$ of the set Z_{k^2} based on $t-1$ selected elements in Z_{k^2} will be called a *t-grouping* of the set Z_{k^2}.

According to Lemma C.4, we can construct (note that k_1 is an odd number, and that $n_1 \geq 4$) an (n_1, k_1)-universal set $\mathcal{P}_1$ of size $D_1 \leq k_1 2^{k_1} \log n_1$ in time $O(\binom{n_1}{k_1} 2^{k_1} k_1 (2n_1)^{(k_1-1)/2})$. Moreover, we define $k_2' = k_2$ if k_2 is odd, and $k_2' = k_2 + 1$ if k_2 is even, and construct an (n_1, k_2')-universal set $\mathcal{P}_2$ of size $D_2 \leq k_2' 2^{k_2'} \log n_1 \leq k_1 2^{k_2+1} \log n_1$ in time $O(\binom{n_1}{k_1} 2^{k_1} k_1 (2n_1)^{(k_1-1)/2})$ (we can replace k_2' by k_1 because by definition $k_2' \leq k_1$). Using the definitions of k_1, k_2, and n_1, it is not hard to verify that

$$D_1 \leq k 2^{k_1 - 1} \qquad \text{and} \qquad D_2 \leq k 2^{k_2}. \tag{4.5}$$

Lemma C.5 *Let $\mathcal{P}$ be an (n, k)-universal set. Then for any n', $k \leq n' \leq n$, $\mathcal{P}$ is also an (n', k)-universal set; and for any $k' \leq k$, $\mathcal{P}$ is also an (n, k')-universal set.*

Proof. Each splitting function in $\mathcal{P}$ can be regarded as a splitting function over $Z_{n'}$. Since every k-subset of $Z_{n'}$ is also a k-subset of Z_n, we conclude that any partition of any k-subset in $Z_{n'}$ is implemented by a splitting function in $\mathcal{P}$, i.e., $\mathcal{P}$ is also an (n', k)-universal set.

Every partition (S_1', S_2') of any k'-subset S' of Z_n can be extended to a partition (S_1, S_2) of a k-subset of Z_n by adding $k - k'$ elements in $Z_n - S'$ to S_1'. Now the splitting function in $\mathcal{P}$ that implements (S_1, S_2) also implements the partition (S_1', S_2') of S'. Thus, $\mathcal{P}$ is also an (n, k')-universal set. □

Now we are ready to construct our (n, k)-universal set $\mathcal{P}$. Each splitting function

over Z_n in $\mathcal{P}$ is defined based on an integer z between 0 and $p-1$, a t-grouping $(X_1, \ldots, X_t)$ of the set Z_{k^2}, $t-1$ splitting functions $f_1, \ldots, f_{t-1}$ in the (n_1, k_1)-universal set $\mathcal{P}_1$, and a splitting function f_t in the (n_1, k_2')-universal set $\mathcal{P}_2$. The splitting function over Z_n is defined in Figure 7.

Splitting $f_{z,(X_1,\ldots,X_t),(f_1,\ldots,f_{t-1},f_t)}(a)$
where $a \in Z_n$ is the input of the function, $0 \leq z < p$, $(X_1, \ldots, X_t)$ is a t-grouping of Z_{k^2}, $f_1, \ldots, f_{t-1}$ are splitting functions in $\mathcal{P}_1$, and f_t is a splitting function in $\mathcal{P}_2$.

1. $x = (az \bmod p) \bmod k^2$;
2. suppose that x is the j-th smallest element in X_i;
3. **return** $f_i(j)$.

Fig. 7. A splitting function over Z_n

First note that the function $f_{z,(X_1,\ldots,X_t),(f_1,\ldots,f_{t-1},f_t)}(a)$ is a well-defined splitting function. In fact, by step 1, x is an element in Z_{k^2}. Since $(X_1, \ldots, X_t)$ is a t-grouping of Z_{k^2}, x must belong to a unique X_i and have a unique rank j in X_i. Thus, step 3 will return a Boolean value $f_i(j)$.

Definition Let $\mathcal{P}$ be the collection of all possible splitting functions over Z_n defined in Figure 7, over all integers z, $0 \leq z < p$, all t-groupings $(X_1, \ldots, X_t)$ of Z_{k^2}, all possible lists $(f_1, \ldots, f_{t-1})$ of splitting functions in the (n_1, k_1)-universal set $\mathcal{P}_1$ (where the same function may appear more than once in the list), and all splitting functions f_t in the (n_1, k_2')-universal set $\mathcal{P}_2$.

Now we are ready for our main result.

Theorem C.6 [93] *For all integers $k \geq 1$ and $n \geq k$, there is an (n, k)-universal set of size bounded by $n2^{k+12\log^2 k+2}$, which can be constructed in time $O(n2^{k+12\log^2 k})$.*

PROOF. The theorem holds true for $k = 1$ by Lemma C.3. Thus, we assume $k \geq 2$. We prove that the collection $\mathcal{P}$ given in the definition before this theorem is an (n, k)-universal set that satisfies the conditions given in the theorem.

We first consider the size of the collection $\mathcal{P}$. There are $p < 2n$ possible integers z. As described earlier, each t-grouping of Z_{k^2} can be given by $t-1$ different elements in Z_{k^2}. Therefore, the total number of different t-groupings of Z_{k^2} is bounded by $\binom{k^2}{t-1} \leq k^{2(t-1)}$. The number of possible lists $(f_1, \ldots, f_{t-1})$ of splitting functions in $\mathcal{P}_1$ is $|\mathcal{P}_1|^{t-1} = D_1^{t-1}$, and finally, the number of splitting functions in $\mathcal{P}_2$ is $|\mathcal{P}_2| = D_2$. Putting all these together, and recall the definitions and inequalities in (4.4) and (4.5), we conclude that the size of $\mathcal{P}$ is bounded by

$$\begin{aligned}
& 2nk^{2(t-1)} D_1^{t-1} D_2 \\
\leq\ & 2nk^{2(t-1)} (k2^{k_1-1})^{t-1} (k2^{k_2}) \\
=\ & 2nk^{3t-2} 2^{(k_1-1)(t-1)+k_2} \\
\leq\ & 2nk^{12\log k+4} 2^{(k_1-1)(t-1)+(k-k_1(t-1))} \\
=\ & n2^{12\log^2 k+4\log k+1} 2^{k-(t-1)} \\
\leq\ & n2^{k+12\log^2 k+2},
\end{aligned}$$

where the last inequality has used the facts $t = \lceil k/k_1 \rceil \geq k/k_1 \geq k/(k/(4\log k)) = 4\log k$.

To construct the collection $\mathcal{P}$, we first construct the collections $\mathcal{P}_1$ and $\mathcal{P}_2$. As discussed earlier, these two collections can be constructed in $O(\binom{n_1}{k_1} 2^{k_1} k_1 (2n_1)^{(k_1-1)/2}) = O(2^k)$ time. Once the collections $\mathcal{P}_1$ and $\mathcal{P}_2$ are available, the integers z, the t-groupings $(X_1, \ldots, X_t)$ of Z_{k^2}, and the lists $(f_1, \ldots, f_{t-1})$ of splitting functions in $\mathcal{P}_1$ and the splitting functions f_t in $\mathcal{P}_2$ can be systematically enumerated, in con-

stant time per combination, which gives a representation of the corresponding splitting function in $\mathcal{P}$. In conclusion, the collection $\mathcal{P}$ can be constructed in time $O(|\mathcal{P}|) = O(n2^{k+12\log^2 k})$.

What remains is to show that $\mathcal{P}$ is an (n,k)-universal set. For this, let S be a given k-subset of Z_n and let (S_1, S_2) be a partition of S. By Proposition C.1, there is an integer z_0, $0 \leq z_0 < p$, such that the function $\psi_{z_0,k^2,n}$ over Z_n is injective from S. Let S', S'_1, and S'_2 be the subsets of Z_{k^2} that are the images of S, S_1, and S_2 under $\psi_{z_0,k^2,n}$, respectively. By the definitions, we have $|S'| = |S|$, $|S'_1| = |S_1|$, $|S'_2| = |S_2|$, and (S'_1, S'_2) is a partition of the k-subset S' in Z_{k^2}.

It is easy to see that there is a t-grouping $(X^0_1, \ldots, X^0_t)$ of the set Z_{k^2} such that each of the first $t-1$ subsets X^0_1, ..., X^0_{t-1} contains exactly k_1 elements in S', and the last subset X^0_t contains k_2 elements in S'. Let $T_i = X^0_i \cap S'$ for $1 \leq i \leq t$. Then T_i is a k_1-subset of X^0_i for $1 \leq i \leq t-1$, and T_t is a k_2-subset of X^0_t. Moreover, the partition (S'_1, S'_2) of S' induces a partition $(T_{i,1}, T_{i,2})$ for each T_i, $1 \leq i \leq t$, where $T_{i,1} = T_i \cap S'_1$ and $T_{i,2} = T_i \cap S'_2$.

Since $\mathcal{P}_1$ is an (n_1, k_1)-universal set, which by Lemma C.5 is also a $(|X^0_i|, k_1)$-universal set, for each i, $1 \leq i \leq t-1$, there is a splitting function f^0_i in $\mathcal{P}_1$ that implements the partition $(T_{i,1}, T_{i,2})$ of the k_1-subset T_i of X^0_i (note that the subset X^0_i can be regarded as the set $Z_{|X^0_i|}$), for $1 \leq i \leq t-1$. That is, $f^0_i(x) = 0$ if $x \in T_{i,1}$ and $f^0_i(y) = 1$ if $y \in T_{i,2}$. Similarly, there is a splitting function f^0_t in $\mathcal{P}_2$ that implements the partition $(T_{t,1}, T_{t,2})$ of the k_2-subset T_t.

Now consider the splitting function $f_{z_0,(X^0_1,\ldots,X^0_t),(f^0_1,\ldots,f^0_t)}$. On an element a in the subset S_1, step 1 of the algorithm **Splitting** produces an element $x = \psi_{z_0,k^2,n}(a)$ in the set S'_1. Suppose that x is in the set X^0_i, then x is in the set $T_{i,1}$. By the way we selected the splitting function f^0_i, we have $f^0_i(x) = 0$. In summary, on an element a in the subset S_1, we have $f_{z_0,(X^0_1,\ldots,X^0_t),(f^0_1,\ldots,f^0_t)}(a) = 0$. Using the same reasoning,

we can show $f_{z_0,(X_1^0,\ldots,X_t^0),(f_1^0,\ldots,f_t^0)}(a) = 1$ for every element a in S_2. Therefore, the function $f_{z_0,(X_1^0,\ldots,X_t^0),(f_1^0,\ldots,f_t^0)}$ in the collection $\mathcal{P}$ implements the partition (S_1, S_2) of the k-subset S of Z_n.

Since S is an arbitrary k-subset of Z_n and (S_1, S_2) is an arbitrary partition of S, we conclude that the collection $\mathcal{P}$ is an (n, k)-universal set. □

we have introduced in detail how to construct the (n, k)-universal set developed by Noar, Schulman, and Srinivasan. Next, we will use this (n, k)-universal set to derandomized our randomized algorithms for PW-PATH, PW-r-D MATCHING, PW-r-SET PACKING, and PW-SET SPLITTING problems.

D. Derandomization

In this section, we discuss how the randomized algorithms presented in the previous sections can be derandomized. Our derandomization process is based on the construction of (n, k)-universal sets, which we discussed in the last section, proposed by Naor, Schulman, and Srinivasan [93].

For a function $\psi_{z,k^2,n}$ from Z_n to $\{0, 1, \ldots, k^2 - 1\}$, as defined in Proposition C.1, we say that the function $\psi_{z,k^2,n}$ *partitions* the set Z_n into k^2 pairwise disjoint subsets $\{W_0, W_1, \ldots, W_{k^2-1}\}$ if for all $0 \leq i \leq k^2 - 1$, $W_i = \{a \in Z_n \mid \psi_{z,k^2,n}(a) = i\}$.

Now let us introduce our derandomized algorithms. First consider the PW-PATH problem. Without loss of generality, we assume that the vertices in the input graph G are labeled by the integers $\{0, 1, \ldots, n-1\}$. The algorithm for the PW-PATH problem is given in Figure 8.

Theorem D.1 *For a graph G of n vertices and m edges, the algorithm* **D-paths**(G, k) *solves the* PW-PATH *problem in time* $4^{k+O(\log^3 k)}nm$.

D-paths(G, k)
input: a weighted graph G with vertex set $V = \{0, 1, \ldots, n-1\}$,
and an integer $k \geq 1$;
output: a k-path of the maximum weight in G if G contains k-paths;

1. **for** $h = 1$ to k **do** construct a (k^2, h)-universal set Ψ_h;
2. let q_0 be the smallest prime number such that $n \leq q_0 < 2n$;
3. $\rho_0 = \emptyset$; {suppose that $\emptyset$ is a virtual k-path of infinitely small weight.}
4. **for** each z, $0 \leq z < q_0$ **do**
4.1. $P_k =$ **path-ext**$(\{\rho_\emptyset\}, V, z, k)$;
4.2. **if** P_k contains a k-path whose weight is larger than that of ρ_0 **then**
replace ρ_0 by the k-path of the maximum weight in P_k;
5. **return** ρ_0.

path-ext(P_l, V', z, h)
input: a set P_l of l-paths in G; a subset V' of vertices in G (V' contains no vertex in P_l); an integer z that lets the function $\psi_{z,k^2,n}$ give an initial partition of V'; and an integer $h \geq 1$;
output: a set P_{l+h} of $(l+h)$-paths in $P_l \odot V'$;

1. $P_{l+h} = \emptyset$;
2. **if** $h = 1$ **then**
2.1. **if** $P_l = \{\rho_\emptyset\}$ **then** P_{l+1} contains a $(u, 1)$-path for each vertex $u \in V'$;
return P_{l+1};
2.2. **else for** each (w, l)-path ρ_l in P_l and each $u \in V'$, where $[w, u]$ is n edge in G, **do**
2.3. oncatenate ρ_l and u to make a $(u, l+1)$-path ρ_{l+1} in $P_l \odot V'$;
2.4. **if** P_{l+1} contains no $(u, l+1)$-path **then** add ρ_{l+1} to P_{l+1};
2.5. **else if** the $(u, l+1)$-path ρ'_{l+1} in P_{l+1} has a weight smaller than that of ρ_{l+1} **then**
2.6. replace ρ'_{l+1} in P_{l+1} by ρ_{l+1};
2.7. **return** P_{l+1};
3. **for** each splitting function f in the (k^2, h)-universal set Ψ_h **do**
3.1. $V_L = \{v, \mid v \in V'$ and $f(\psi_{z,k^2,n}(v)) = 0\}$;
3.2. $V_R = \{v, \mid v \in V'$ and $f(\psi_{z,k^2,n}(v)) = 1\}$;
3.3. $P^L_{l+\lceil h/2 \rceil} =$ **path-ext**$(P_l, V_L, z, \lceil h/2 \rceil)$;
3.4. **if** $P^L_{l+\lceil h/2 \rceil} \neq \emptyset$ **then**
3.5. $P^R_{l+h} =$ **path-ext**$(P^L_{l+\lceil h/2 \rceil}, V_R, z, \lfloor h/2 \rfloor)$;
3.6. **for** each $(u, l+h)$-path ρ_{l+h} in P^R_{l+h} **do**
3.7. **if** P_{l+h} contains no $(u, l+h)$-path in $P_l \odot V'$ **then** add ρ_{l+h} to P_{l+h};
3.8. **else if** the $(u, l+h)$-path ρ'_{l+h} in P_{l+h} has a weight smaller than that of ρ_{l+h} **then**
3.9. replace ρ'_{l+h} in P_{l+h} by ρ_{l+h};
4. **return** P_{l+h}.

Fig. 8. A deterministic algorithm for the PW-PATH problem

PROOF. First consider the correctness of the algorithm. Note that the path ρ_0 returned by the algorithm **D-paths** remains $\emptyset$ unless step 4.2 of the algorithm replaces ρ_0 by a real k-path in G. In particular, the algorithm works correctly if the graph G contains no k-paths.

Now let ρ_k be a k-path of the maximum weight in the graph G. By Proposition C.1, there is an integer z_0, $0 \leq z_0 < q_0$, such that the function $\psi_{z_0,k^2,n}$ is injective from the k-subset consisting of the k vertices in the path ρ_k. Consider the execution of step 4.1 in algorithm **D-paths** on this particular integer z_0, which calls the subroutine **path-ext**$(\{\rho_\emptyset\}, V, z_0, k)$.

The subroutine **path-ext** has a similar structure as that of the algorithm **find-paths** in Figure 3, and our discussion will concentrate on the differences. For the integer z_0 and for any subset V' of V, we say that a path ρ is (z_0, V')-*separated* if for any two vertices u_1 and u_2 in ρ, where $u_1, u_2 \in V'$, we have $\psi_{z_0,k^2,n}(u_1) \neq \psi_{z_0,k^2,n}(u_2)$. For a general input (P_l, V', z_0, h) to the subroutine **path-ext**, we prove the following claim:

> For any vertex $v \in V'$, there is a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ if and only if the set P_{l+h} constructed by the subroutine **path-ext**(P_l, V', z_0, h) contains a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ whose weight is the maximum over all (z_0, V')-separated $(v, l+h)$-paths in $P_l \odot V'$.

One direction is trivial: it suffices to ensure the existence of a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ if the set P_{l+h} contains a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$. We prove the other direction by induction on h. For the case $h = 1$, the algorithm **path-ext** proceeds in exactly the same way as that of the algorithm **find-**

paths in Figure 3. Since every $(v, l+1)$-path in $P_l \odot V'$ is (z_0, V')-separated, in this case the corresponding part of the proof for algorithm **find-paths** (i.e., Theorem A.4) is directly applied to derive the correctness of the above claim. Now consider the case $h > 1$. Suppose that there is a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$, and let

$$\rho_{l+h} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1}, \ldots, w_h \rangle$$

be a (z_0, V')-separated $(v, l+h)$-path of the maximum weight in $P_l \odot V'$, where $\langle u_1, \ldots, u_l \rangle$ is a path in P_l, $w_j \in V'$ for all j, $h_1 = \lceil h/2 \rceil$, and $w_h = v$. Since the path ρ_{l+h} is (z_0, V')-separated, the set $S_h = \{\psi_{z_0,k^2,n}(w_1), \ldots, \psi_{z_0,k^2,n}(w_h)\}$ consists of exactly h elements in the set Z_{k^2}, and $(\{\psi_{z_0,k^2,n}(w_1), \ldots, \psi_{z_0,k^2,n}(w_{h_1})\}, \{\psi_{z_0,k^2,n}(w_{h_1+1}), \ldots, \psi_{z_0,k^2,n}(w_h)\})$ is a partition of S_h. By the definition of the (k^2, h)-universal set Ψ_h, there is a splitting function f_0 in Ψ_h such that $f_0(\psi_{z_0,k^2,n}(w_j)) = 0$ for $1 \leq j \leq h_1$, and $f_0(\psi_{z_0,k^2,n}(w_j)) = 1$ for $h_1 + 1 \leq j \leq h$. Therefore, when this splitting function f_0 is picked in step 3 of the algorithm **path-ext**(P_l, V', z_0, h), the set V_L obtained in step 3.1 contains the vertices w_1, …, w_{h_1}, and the set V_R obtained in step 3.2 contains the vertices w_{h_1+1}, …, w_h.

Note that the path $\rho_{l+h_1} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1} \rangle$ is a (z_0, V_L)-separated $(w_{h_1}, l+h_1)$-path in $P_l \odot V_L$. By the induction hypothesis, the set $P^L_{l+h_1}$ obtained in step 3.3 contains a (z_0, V_L)-separated $(w_{h_1}, l+h_1)$-path $\overline{\rho}_{l+h_1}$ in $P_l \odot V_L$ whose weight is at least as large as that of ρ_{l+h_1}. Now the concatenation of the path $\overline{\rho}_{l+h_1}$ and the path $\langle w_{h_1+1}, \ldots, w_h \rangle$ is a (z_0, V_R)-separated $(w_h, (l+h_1)+(h-h_1))$-path (i.e., a $(v, l+h)$-path) in $P^L_{l+h_1} \odot V_R$. Thus, by our induction hypothesis again, the set P^R_{l+h} obtained in step 3.5 contains a (z_0, V_R)-separated $(v, l+h)$-path $\overline{\rho}_{l+h}$ in $P^L_{l+h_1} \odot V_R$ whose weight is at least as large as the sum of the weights of the paths $\overline{\rho}_{l+h_1}$ and $\langle w_{h_1+1}, \ldots, w_h \rangle$. Since the weight of $\overline{\rho}_{l+h_1}$ is not smaller than that of ρ_{l+h_1}, we conclude that the weight of the path $\overline{\rho}_{l+h}$ is not smaller than that of ρ_{l+h}. Finally, since the path $\overline{\rho}_{l+h}$ is a

concatenation of a $(w, l+h_1)$-path ρ'_{l+h_1} in $P^L_{l+h_1}$ and a path in $G[V_R]$ that is (z_0, V_R)-separated, where by the induction hypothesis, ρ'_{l+h_1} is a (z_0, V_L)-separated path in $P_l \odot V_L$, we derive that $\overline{\rho}_{l+h}$ is actually a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ (note that for two vertices $w \in V_L$ and $w' \in V_R$, the values $\psi_{z_0,k^2,n}(w)$ and $\psi_{z_0,k^2,n}(w')$ are always different because they are mapped to different values under the splitting function f_0). Since the weight of $\overline{\rho}_{l+h}$ is not smaller than the weight of ρ_{l+h}, and by our assumption, the path ρ_{l+h} has the maximum weight over all (z_0, V')-separated $(v, l+h)$-paths in $P_l \odot V'$, we conclude that $\overline{\rho}_{l+h}$ must also be a (z_0, V')-separated $(v, l+h)$-path of the maximum weight in $P_l \odot V'$. Therefore, the collection P_{l+h} returned by the subroutine **path-ext**(P_l, V', z_0, h) must contain a (z_0, V')-separated $(v, l+h)$-path of the maximum weight. This completes the proof for the claim.

Now the correctness of the algorithm **D-paths** can be easily derived from the above claim: by our assumption on the integer z_0, the k-path ρ_k of the maximum weight in the graph G is actually a (z_0, V)-separated (v, k)-path in $\{\rho_\emptyset\} \odot V$ (i.e., in the graph G). Therefore, the above claim concludes that the set P_k obtained in step 4.1 of the algorithm **D-paths** by calling the subroutine **path-ext**$(\{\rho_\emptyset\}, V, z_0, k)$ must contain a (v, k)-path of the maximum weight. In consequence, the path ρ_0 returned in step 5 of the algorithm **D-paths**(G, k) must be a k-path of the maximum weight.

For the running time of the algorithm, note that the running time of the algorithm **D-paths** is dominated by step 4, which is a q_0-time iteration of the subroutine **path-ext**, where $q_0 = O(n)$. Let $T(h, m)$ be the running time of the recursive subroutine **path-ext**(P_l, V', z, h), where m is the number of edges in the input graph G to the main algorithm **D-paths**(G, k). By Theorem C.6, the (k^2, h)-universal set Ψ_h has at most $k^2 2^{h+12\log^2 h+2}$ splitting functions. Therefore, the value $T(h, m)$ satisfies the

following recurrence relations (where c_1 and c_2 are constants):

$$\begin{aligned} T(1,m) &\leq c_1 m; \\ T(h,m) &\leq k^2 2^{h+12\log^2 h+2}[c_2 m + T(\lceil h/2 \rceil, m) + T(\lfloor h/2 \rfloor, m)]. \end{aligned} \tag{4.6}$$

Let $X_0 = k^2 2^{12\log^2 k+2} = 2^{12\log^2 k+2\log k+2}$. Since $k \geq h$ for all values h used in the subroutine **path-ext**(P_l, V', z, h), we derive from (4.6) that

$$\begin{aligned} T(1,m) &\leq c_1 m; \\ T(h,m) &\leq 2^h X_0[c_2 m + T(\lceil h/2 \rceil, m) + T(\lfloor h/2 \rfloor, m)]. \end{aligned} \tag{4.7}$$

Since the value X_0 is independent of the variables h and m, we can apply Corollary A.2 to the recurrence relations in (4.7), and obtain $T(h,m) = O(4^h h^\alpha m)$, where α is any number larger than $\log_2(X_0(4+1)/2)$. In particular, if we pick $\alpha = \log_2 X_0 + 2 = 12\log^2 k + 2\log k + 4$, we have

$$h^\alpha \leq k^\alpha = 2^{\alpha \log k} = 2^{O(\log^3 k)},$$

which gives

$$T(h,m) = O(4^h 2^{O(\log^3 k)} m) = 4^{k+O(\log^3 k)} m.$$

Combining this with our previous discussion, we conclude that the running time of the algorithm **D-paths** is bounded by $4^{k+O(\log^3 k)} nm$. □

Using the same technique, we can develop improved deterministic algorithms for the matching and packing problems discussed in section A.

Theorem D.2 *There is a deterministic algorithm that solves the* PW-r-SET PACKING *problem in time* $4^{rk+O(\log^3(rk))} n^2$*, where* n *is the number of* r*-sets in the input instance.*

PROOF. The deterministic algorithm for the PW-r-SET PACKING problem is a

derandomization of the algorithm given in Figure 4. The procedure is very similar to that described in Theorem D.1 for the PW-PATH problem. The only difference is that here we use an $((rk)^2, rh)$-universal set in which a splitting function splits the (implicit) h-packing of the maximum weight into two packings of sizes $\lceil h/2 \rceil$ and $\lfloor h/2 \rfloor$, respectively. We leave the details for interested readers to verify. □

Corollary D.3 *There is a deterministic algorithm that solves the* PW-r-D MATCHING *problem in time* $4^{rk+O(\log^3(rk))}n^2$*, where* n *is the number of points in the input instance.*

Corollary D.4 *Let* H *be a fixed graph of* r *vertices. There is a deterministic algorithm that solves the* PW-H-GRAPH PACKING *problem in time* $4^{rk+O(\log^3(rk))}n^{r+1}$*, where* n *is the number of vertices in the input graph* G*.*

Using Theorem C.6, we can also derandomize our algorithm for the PW-SET SPLITTING problem in section B, and obtain a deterministic parameterized algorithm of running time $O^*(4^{k+o(k)})$ for the PW-SET SPLITTING problem. This also provides the first proof for the fixed parameter tractability of the problem.

Theorem D.5 *The* PW-SET SPLITTING *problem can be solved by a deterministic algorithm of running time* $O(N^2 4^{k+6\log^2 k+6\log k}) = O(N^2 4^{k+o(k)})$*, where* N *is the instance size of the problem.*

PROOF. Let $(X, \mathcal{F}, k)$ be an instance of the PW-SET SPLITTING problem, where without loss of generality, let the set X be Z_n. Then in $O(n2^{2k+12\log^2(2k)+12\log(2k)}) = O(n4^{k+o(k)})$ time, we can construct an $(n, 2k)$-universal set $\mathcal{P}$ based on Theorem C.6.

We use each splitting function in $\mathcal{P}$ to partition the ground set X, and see if the corresponding partition of X splits at least k subsets in $\mathcal{F}$. If so, we record the collection of the k subsets of the largest weight that are split by this partition. We

repeat this process for all splitting functions in $\mathcal{P}$. The output of our algorithm is either "No" if no partition of X is constructed in this process that splits k subsets in $\mathcal{F}$, or the collection of the k subsets of the largest weight over all collections recorded in this process.

If the answer to the instance $(X, \mathcal{F}, k)$ is "No", then the above algorithm obviously returns "No" because the algorithm does not return "No" only if it actually constructs a partition of X that splits k subsets in $\mathcal{F}$. On the other hand, if the answer to the instance is not "No", then there is a partition of X that splits k subsets $S_1, \ldots, S_k$ in $\mathcal{F}$ whose total weight is the largest over all k split subsets caused by partitions of X. As we explained in the previous section, there is a set W of at most $2k$ elements in X and a partition (W_1, W_2) of W such that $W_1 \cap S_i \neq \emptyset$ and $W_2 \cap S_i \neq \emptyset$, for all $1 \leq i \leq k$. Therefore, when we perform the above process using a splitting function f in $\mathcal{P}$ that implements (W_1, W_2) (note by Lemma C.5, f is an $(n, |W|)$-universal set even if $|W| < 2k$), the corresponding partition of X will split all these k subsets $S_1, \ldots, S_k$, and record the collection of k subsets of largest weight in $\mathcal{F}$ that can be split by a partition of the ground set X. □

E. Chapter Conclusion

In section A of the chapter, we developed improved randomized algorithms for the PW-PATH, PW-r-D MATCHING and PW-r-SET PACKING problems. The time complexity for the PW-PATH problem is improved from $O^*(5.5^k)$ to $O^*(4^k)$, and time complexities for the PW-r-D MATCHING and PW-r-SET PACKING problems are improved from $O^*(5.5^{rk})$ to $O^*(4^{(r-1)k})$. These improvements are based on our newly developed technique, divide-and-conquer, which is a general technique that has been used to solve the PW-PATH, PW-r-D MATCHING and PW-r-SET PACKING problems, and will

also be applied to other parameterized NP-hard problems.

In section B, we designed a randomized algorithm, that is based on the subset partition technique and has a running time bounded by $O^*(2^k)$ for the PW-SET SPLITTING problem. This algorithm is significantly better than the previous best known algorithm of running time bounded by $O^*(2.65^k)$ (which is a deterministic algorithm that only works for the simpler P-SET SPLITTING problem). In section B, we also proposed an effective technique based on a probabilistic method that allows us to develop a simpler and more efficient (deterministic) kernelization algorithm for the P-SET SPLITTING problem. This new technique has also led to a better kernel if many subsets in the instance have more than two elements.

In section C, we introduced (n, k)-universal sets which were first constructed by Naor, Schulman, and Srinivasan [93]. As the orginal construction was presented in an extended abstract [93], in which many details were omitted, for a more general problem. For the completeness of our discussion, we reproduced a slightly different and simpler construction and its analysis specifically for (n, k)-universal sets. The presentation in section C is more precise with all needed details provided.

In section D, we used the (n, k)-universal sets to de-randomize our algorithms in sections A and B. This leads to improved deterministic algorithms for the PW-PATH, PW-r-D MATCHING, PW-r-SET PACKING, and PW-SET SPLITTING problems.

CHAPTER V

COLOR CODING AND ITERATIVE EXPANSION

A k-color coding scheme is a collection of k-coloring functions (color a set with k colors) such that every subset of size k in the ground set of size n is colored properly, i.e. no two elements in the subset are assigned with the same color, by at least one k-coloring function in the scheme. Hence if we enumerate all coloring functions in the scheme, the unknown solution subset of size k must be colored properly by at least one coloring function. The goal to color the solution subset properly is that it is helpful to find the solution through dynamic programming. The time complexity in the coloring step is decided by the number of k-coloring functions in the color coding scheme and the time complexity in the dynamic programming step is decided by the number of colors used in the scheme.

For some problems, if a solution of size k is given, it is helpful to find a solution of size $k+1$. Hence, we can begin from a solution of size 1. Using the solution of size 1, we find a solution of size 2. Then using the solution of size 2, we find a solution of size 3, and so on until we find a solution of the proper size. This method is called the iterative expansion. In this chapter, we will introduce how to construct a k-color coding scheme of size $O^*(6.4^k)$ and how iterative expansion technique helps us to solve the p-3-d matching problem.

A. Introduction

The 3-d matching problem is one of the six "basic" NP-complete problems according to Garey and Johnson [56]. Since Downey and Fellows' initial work [44], the parameterized version of the 3-d matching problem, or the p-3-d matching problem has attracted considerable interests in recent years, where one is looking for a matching

of size k. The research has resulted in a number of new algorithmic techniques and an impressive list of improved algorithms [29, 22, 38, 44, 50, 66, 74, 75, 76, 86, 119]. There have been three popular new techniques in developing improved algorithms for P-3-D MATCHING:

- *Greedy localization.* This method tries to apply local optimization to effectively reduce the search space when one is looking for a matching of size k. Using this method, Chen *et al.* [22] obtained an $O^*((5.7k)^k)$ time algorithm for the problem, and Jia *et al.* [66] extended the technique to P-3-SET PACKING that is a generalization of P-3-D MATCHING.

- *Color coding.* This method considers the construction of a collection of partitions of a universal set so that the symbols in the desired matching are effectively separated. Using this method, Koutis [75] developed an $O^*(10.88^{3k})$ time randomized algorithm for P-3-D MATCHING. The algorithm can be de-randomized using a method suggested in [3], resulting in an $O^*(c_1^{3k})$ time deterministic algorithm, where c_1 is a very large constant. Fellows *et al.* [50] revised the technique and presented an improved randomized algorithm of running time $O^*(5.44^{3k})$ and an improved deterministic algorithm of running time $O^*(c_2^{3k})$, where one can deduce that $c_2 \geq 132$. Chen *et al.* [29] used an improved color coding scheme and developed an improved deterministic algorithm of running time $O^*(12.8^{3k})$.

- *Randomized divide-and-conquer.* This more recent technique uses randomization to effectively partition the desired matching so that the classical divide-and-conquer method can be applied. The method was independently proposed by Kneis *et al.* [74] and by Chen *et al.* [29], who both developed an $O^*(2.51^{3k})$ time randomized algorithm for the P-3-D MATCHING problem. A de-randomization process was proposed in [74] that leads to an $O^*(16^{3k})$ time deterministic algo-

rithm, and another de-randomization process was proposed in [29] that leads to an $O^*(4^{3k+o(k)})$ time deterministic algorithm. Very recently, Wang and Feng [119] further refined the method, and developed an $O^*(3.52^{3k})$ time deterministic algorithm for the problem.

We remark that recently, a fourth method has been announced by Koutis [76], who proposed an algebraic method that leads to a randomized algorithm of running time $O^*(2^{3k})$ for P-3-D MATCHING. However, it is unknown whether his algorithm can be de-randomized to result in an improved deterministic algorithm for the problem. Moreover, we point out that all these techniques are also applicable to the P-3-SET PACKING problem, which is a generalization of the P-3-D MATCHING problem. On the other hand, it is unknown whether any of these techniques can take any advantage of the structure of P-3-D MATCHING and lead to more efficient algorithms for P-3-D MATCHING.

Table I summaries these algorithmic results and the corresponding employed techniques for the P-3-D MATCHING problem. We have also included our results in this chapter for comparison. The bound $O^*(3.52^{3k})$ given in [119] stands as the best previous upper bound for deterministic algorithms for P-3-D MATCHING. Our new results presented in the current chapter give an $O^*(2.80^{3k})$ time deterministic algorithm for P-3-D MATCHING.

Our approach, which is named *iterative expansion*, proceeds based on the integration and improvements of a number of previous techniques. First, we present an $O^*(6.4^k)$ time deterministic construction of a k-color coding scheme, which significantly improves the previous construction [3], and also directly implies improvements on all previous deterministic algorithms for a variety of problems that are based on the color coding method, including those for P-3-D MATCHING as given in [75, 50].

Table I. Algorithms for the P-3-D MATCHING problem

References	randomized	deterministic	main techniques
Downey and Fellows [44]	—	$O^*((3k)!(3k)^{9k+1})$	H
Chen *et al.* [22]	—	$O^*((5.7k)^k)$	G
Koutis [75]	$O^*(10.88^{3k})$	$O^*(2^{O(k)})$	C, D
Fellows *et al.* [50]	$O^*(5.44^{3k})$	$O^*(132^{3k})$	C, D
Chen *et al.* [29]	$O^*(5.44^{3k})$	$O^*(12.8^{3k})$	C, D
Kneis *et al.* [74]	$O^*(2.51^{3k})$	$O^*(16^{3k})$	R
Chen *et al.* [29]	$O^*(2.51^{3k})$	$O^*(4^{3k+o(k)})$	R
Wang and Feng [119]	$O^*(3.52^{3k})$	$O^*(3.52^{3k})$	C, R
Koutis [76]	$O^*(2^{3k})$	—	A
This chapter	$O^*(2.32^{3k})$	$O^*(2.80^{3k})$	C, D, G, I

Techniques: [A] algebraic methods; [C] color coding; [D] dynamic programming; [G] greedy localization; [H] hashing; [I] iterative expansion; [R] randomized divide-and-conquer

Secondly, by a more careful investigation on the structures of P-3-D MATCHING, we prove that if a triple set contains a matching of $k+1$ triples, then every matching of k triples is overlapping with some matching of $k+1$ triples by at least $2/3$ of its symbols. This structural result significantly improves the one given in [22] that was the basis for the greedy localization method. Moreover, we replace the enumeration phase in the greedy localization method by a more efficient phase that uses the improved color coding and dynamic programming method. Finally, by taking advantage of symbol order in a triple, we show that the dynamic programming phase in the above process

can be further improved. The combination of all these improvements leads to an improved $O^*(2.80^{3k})$ time deterministic algorithm for P-3-D MATCHING.

Finally, we remark that all previous parameterized algorithms for P-3-D MATCHING and P-3-SET PACKING have the same time complexity, although it is obvious that P-3-SET PACKING is a nontrivial generalization of P-3-D MATCHING. For the first time, we take advantage of the structure of P-3-D MATCHING and present a faster algorithm for P-3-D MATCHING.

B. An Improved Color Coding Scheme

For an integer n, denote by Z_n the set $\{0, 1, \ldots, n-1\}$. If n is a prime number, then Z_n is a field under addition and multiplication modulo n. A function f on Z_n is *injective from a subset S of Z_n* if for any $x, y \in S$, $x \neq y$, we have $f(x) \neq f(y)$. A *k-coloring* of Z_n is a function mapping Z_n to Z_k, and a *coloring* of Z_n is a k-coloring of Z_n for some integer k. A collection $\mathcal{C}$ of k-colorings of Z_n is a *k-color coding scheme* if for every subset S_k of k elements in Z_n, there is a k-coloring f in $\mathcal{C}$ that is injective from S_k (in this case, we also say that *the coloring f colors S_k properly*). The *size* of the k-color coding scheme $\mathcal{C}$ is the number of colorings in $\mathcal{C}$.

The concept of the color coding method was proposed by Alon, Yuster, and Zwick in their seminal work [3], and the efficiency of the method depends directly on the size of the color coding scheme. An explicit construction of a k-color coding scheme was suggested in [3] based on the construction of a perfect hash function given in [108], which has size $\Omega(c^k)$, where $c \geq 4000$ (this bound was not made explicit in [3] but is estimated based on the construction in [108]).

In this section, we present a k-color coding scheme of significantly improved size.

1. A special collection of color coding schemes

We start with a few simple lemmas on color coding schemes.

Lemma B.1 *For any integers n and k, $n \geq k$, there is a k-color coding scheme for Z_n of size bounded by $\binom{n}{k}$.*

PROOF. For each subset W of k elements in Z_n, construct a k-coloring f_W for Z_n that assigns each element in W a distinct color and colors all other elements in Z_n arbitrarily. The k-coloring f_W is obviously injective from W. The collection $\{f_W \mid W \subseteq Z_n,\ |W| = k\}$ of $\binom{n}{k}$ k-colorings is a k-color coding scheme for the set Z_n. □

Lemma B.2 *For $n \geq 2$, there exist* (1) *a 1-color coding scheme of size 1 for Z_n;* (2) *an n-color coding scheme of size 1 for Z_n; and* (3) *a 2-color coding scheme of size $\leq \lceil \log n \rceil$ for Z_n.*

PROOF. Facts (1) and (2) are obvious. We prove Fact (3) by induction on n. The fact can be easily verified for the cases of $n \leq 4$. Inductively, suppose that for $n \leq 2^g$, $g \geq 2$, Fact (3) holds true. Now we consider the case $2^g < n \leq 2^{g+1}$.

Partition the n elements in Z_n into two subsets Z' and Z'' such that $|Z'| = 2^g$ and $|Z''| = n - 2^g \leq 2^g$. By the inductive hypothesis, there exist a 2-color coding scheme $\mathcal{F}' = \{f'_1, \ldots, f'_{g'}\}$ of size g' for Z' and a 2-color coding scheme $\mathcal{F}'' = \{f''_1, \ldots, f''_{g''}\}$ of size g'' for Z'', where $g' \leq \lceil \log |Z'| \rceil = g$ and $g'' \leq \lceil \log |Z''| \rceil \leq g$. Without loss of generality, we assume that $g' \geq g''$.

Construct g' 2-colorings for Z_n:

$$(f'_1, f''_1), (f'_2, f''_2), \ldots, (f'_{g''}, f''_{g''}), (f'_{g''+1}, f''_{g''}), \ldots, (f'_{g'}, f''_{g''}), \tag{5.1}$$

where the coloring (f'_i, f''_j) colors the elements in Z' using the 2-coloring f'_i, and the elements in Z'' using the 2-coloring f''_j (both f'_i and f''_j use the color set $\{0,1\}$). We also introduce a new 2-coloring f_0 for Z_n that assigns 0 to all elements in Z' and 1 to all elements in Z''. It is straightforward to verify that the 2-coloring f_0 plus the g' 2-colorings in (5.1) makes a 2-color coding scheme for the set Z_n, whose size is $g' + 1 \le g + 1 = \lceil \log n \rceil$. This proves the lemma. □

For general k, we have the following recurrence relation.

Lemma B.3 *Let $n = n_1 + \cdots + n_r$, where all $n_j \ge 1$ are integers. Let $\tau(n', k')$ be an upper bound for the size of a k'-color coding scheme for the set $Z_{n'}$, where $n' < n$ and $k' \le k$. Then there is a k-color coding scheme for the set Z_n whose size is bounded by*

$$\sum_{\substack{k_1+\cdots+k_r=k \\ 0\le k_1\le n_1,\ldots,0\le k_r\le n_r}} \left(\frac{\tau(\#[k_j \le 1], \#[k_j = 1])}{\binom{\#[k_j\le 1]}{\#[k_j=1]}} \prod_{k_j \ge 2} \tau(n_j, k_j) \right),$$

where $\#[k_j \le 1]$ and $\#[k_j = 1]$ are the numbers of k_j's in the list $[k_1, \ldots, k_r]$ such that $k_j \le 1$ and $k_j = 1$, respectively.

PROOF. Arbitrarily partition the set Z_n into r disjoint subsets Y_1, ..., Y_r, where $|Y_j| = n_j$ for all j. Let L be the collection of all lists $[k_1, \ldots, k_r]$ of r integers satisfying $k_1 + \cdots + k_r = k$ and $0 \le k_j \le n_j$ for all j. We say that two lists $[k_1, \ldots, k_r]$ and $[k'_1, \ldots, k'_r]$ in L are *conjugate* if for every j, either $k_j \ge 2$ or $k'_j \ge 2$ will imply $k_j = k'_j$. It is clear that this conjugation is an equivalence relation and partitions the lists in L into equivalence classes. A conjugation equivalence class will be called a $(k_1, \ldots, k_r)$-class for any list $[k_1, \ldots, k_r]$ in the class. Each $(k_1, \ldots, k_r)$-class contains exactly $\binom{\#[k_j\le 1]}{\#[k_j=1]}$ lists in L.

Fix a $(k_1, \ldots, k_r)$-class. For each j such that $k_j \ge 2$, let $\mathcal{F}_{n_j,k_j}$ be a k_j-color coding scheme of size bounded by $\tau(n_j, k_j)$ for the set Z_{n_j}. Moreover, let $\mathcal{F}$ be a

$(\#[k_j = 1])$-color coding scheme of size bounded by $\tau(\#[k_j \leq 1], \#[k_j = 1])$ for the set $Z_{\#[k_j \leq 1]}$. Suppose that the color sets used by all these schemes are disjoint. We construct a set of at most

$$\tau(\#[k_j \leq 1], \#[k_j = 1]) \prod_{k_j \geq 2} \tau(n_j, k_j)$$

k-colorings for the set Z_n: each of these k-colorings consists of a k_j-coloring from the scheme $\mathcal{F}_{n_j,k_j}$ for the set Y_j for each $k_j \geq 2$, plus a $(\#[k_j = 1])$-coloring from the scheme $\mathcal{F}$ that treats each set Y_j with $k_j \leq 1$ as a single element and assigns all elements in Y_j with the same color.

We apply the above process to each $(k_1, \ldots, k_r)$-class, which gives a collection of

$$\sum_{0 \leq k_1 \leq n_1, \ldots, 0 \leq k_r \leq n_r}^{k_1 + \cdots + k_r = k} \left(\frac{\tau(\#[k_j \leq 1], \#[k_j = 1])}{\binom{\#[k_j \leq 1]}{\#[k_j = 1]}} \prod_{k_j \geq 2} \tau(n_j, k_j) \right)$$

k-colorings for the set Z_n (note that each $(k_1, \ldots, k_r)$-class contains exactly $\binom{\#[k_j \leq 1]}{\#[k_j = 1]}$ lists in the collection L). To complete the proof of the lemma, it remains to show that this collection makes a k-color coding scheme for the set Z_n.

Let W be an arbitrary subset of k elements in Z_n. Suppose that for each j, W has exactly k_j elements in the set Y_j. Note that $[k_1, \ldots, k_r]$ is a list in the collection L. For each $k_j \geq 2$, since $\mathcal{F}_{n_j,k_j}$ is a k_j-color coding scheme for Y_j, one k_j-coloring f_j in $\mathcal{F}_{n_j,k_j}$ must be injective from the k_j elements of W that are in Y_j. On the other hand, since $\mathcal{F}$ is a $(\#[k_j = 1])$-color coding scheme for the set $Z_{\#[k_j \leq 1]}$, one $(\#[k_j = 1])$-coloring f in $\mathcal{F}$ assigns each of the $\#[k_j = 1]$ sets Y_j with $k_j = 1$ a distinct color. Therefore, the combination of these k_j-colorings f_j and the $(\#[k_j = 1])$-coloring f, which is one of the k-colorings constructed above, makes a k-coloring for the set Z_n that is injective from the subset W. This completes the proof of the lemma. $\square$

By Lemma B.2 and Lemma B.3, for small values of n and k, we can construct a k-color coding scheme for the set Z_n and derive an upper bound $\tau(n,k)$ for its size. We did this for special pairs (n,k) for small values of n and k, where $n = k(k-1)$, using a computer program based on Lemma B.3. Our computation results are given in Table II (the last column in the table will be used for later discussion).

The main result of this subsection is the following lemma, which will be used in bounding the size of our final color coding scheme.

Lemma B.4 *Let $[c_0, c_1, \ldots, c_r]$ be a list of non-negative integers such that $\sum_{j=0}^{r} c_j = k$ and $\sum_{j=0}^{r} c_j(c_j-1) \leq 4k$. Then there is a collection $\{\mathcal{F}_{c_0}, \mathcal{F}_{c_1}, \ldots, \mathcal{F}_{c_r}\}$ of color coding schemes, where $\mathcal{F}_{c_j}$ is a c_j-color coding scheme for the set $Z_{c_j(c_j-1)}$, such that $\prod_{c_j \geq 2} |\mathcal{F}_{c_j}| \leq 2.4142^k$.*

PROOF. For $2 \leq c_j \leq 18$, we use the c_j-color coding scheme $\mathcal{F}_{c_j}$ for the set $Z_{c_j(c_j-1)}$ given in Table II, whose size is bounded by $\tau(c_j(c_j-1), c_j)$ in the third column of the table. For $c_j > 18$, we simply use the trivial c_j-color coding scheme $\mathcal{F}_{c_j}$ for the set $Z_{c_j(c_j-1)}$ given in Lemma B.1, whose size is bounded by $\tau(c_j(c_j-1), c_j) = \binom{c_j(c_j-1)}{c_j}$.

Let $B_{c_j} = (\tau(c_j(c_j-1), c_j))^{4/c_j(c_j-1)}$. It is easy to verify from Table II that $B_{c_j} \leq 2.4142$ for $c_j \leq 18$ (see the fourth column in the table). For $c_j > 18$, by the definition of $\tau(c_j(c_j-1), c_j)$, $B_{c_j} = (\tau(c_j(c_j-1), c_j))^{4/c_j(c_j-1)} = \binom{c_j(c_j-1)}{c_j}^{4/c_j(c_j-1)}$. Consider

Table II. Upper bound on the size of a k-color coding scheme for Z_n

k	$n = k(k-1)$	upper bound $\tau(n,k)$	$B_k = (\tau(n,k))^{4/n}$
2	2	1	1
3	6	3	2.0801
4	12	12	2.2895
5	20	82	2.4142
6	30	434	2.2474
7	42	2,937	2.1394
8	56	16,960	2.0050
9	72	115,251	1.9108
10	90	655,756	1.8136
11	110	4,731,907	1.7488
12	132	33,489,268	1.6906
13	156	260,723,566	1.6437
14	182	1,426,381,707	1.5893
15	210	13,008,846,025	1.5584
16	240	58,465,192,360	1.5117
17	272	676,712,910,839	1.4928
18	306	6,079,615,220,515	1.4693

$$\begin{aligned} f(x) &= \binom{x(x-1)}{x}^{\frac{1}{x(x-1)}} \\ &= \left[\frac{[x(x-1)][x(x-1)-1]\cdots[x(x-1)-x+1]}{x!}\right]^{\frac{1}{x(x-1)}} \\ &\leq \left[\frac{[x(x-1)-\frac{x-1}{2}]^x}{\sqrt{2\pi x}(x/e)^x}\right]^{\frac{1}{x(x-1)}} \\ &< \left[\frac{(\frac{2x(x-1)-(x-1)}{2})^x}{(x/e)^x}\right]^{\frac{1}{x(x-1)}} \\ &= \left(\frac{e(2x-1)(x-1)}{2x}\right)^{\frac{1}{x-1}}, \end{aligned}$$

where in the first inequality, we have used the inequalities $ab \leq ((a+b)/2)^2$ and $x! \geq \sqrt{2\pi x}(x/e)^x$.

Let $g(x) = [e(2x-1)(x-1)/(2x)]^{1/(x-1)}$. It can be verified that when $x \geq 7$, $g(x)$ is strictly decreasing. In particular, for $c_j \geq 19$, we have

$$B_{c_j} = (f(c_j))^4 < (g(c_j))^4 \leq (g(19))^4 \leq 2.3599.$$

Thus, $B_{c_j} \leq 2.4142$ for all c_j. Combining this with $\sum_{j=0}^{r} c_j(c_j-1) \leq 4k$, we obtain

$$\begin{aligned} &\prod_{c_j \geq 2} |\mathcal{F}_{c_j}| \leq \prod_{c_j \geq 2} \tau(c_j(c_j-1), c_j) = \prod_{c_j \geq 2} B_{c_j}^{c_j(c_j-1)/4} \leq \prod_{c_j \geq 2} 2.4142^{c_j(c_j-1)/4} \\ = \;& 2.4142^{\sum_{c_j \geq 2} c_j(c_j-1)/4} = 2.4142^{\sum_{j=0}^{r} c_j(c_j-1)/4} \leq 2.4142^k. \end{aligned}$$

This completes the proof of the lemma. □

2. A k-coloring algorithm with a given set of parameters

In the rest of the discussion in this section, we fix integers n and k, where $n \geq k$. At the moment, we assume that k is divisible by 4 (this constraint will be removed in

our final construction). We will concentrate on k-color coding schemes for the set Z_n.

Let p_0 and p be prime numbers satisfying $n \leq p_0 < 2n$ and $k^2 < p < 2k^2$ (such prime numbers exist by Bertrand's Conjecture [63]). The prime numbers p_0 and p can be obtained in time $O(n\sqrt{n})$ and $O(k^2\sqrt{k^2}) = O(k^3)$, respectively, using a trivial primality testing algorithm.

We present a k-coloring algorithm for the set Z_n. The algorithm is associated with a set of parameters satisfying the following conditions:

C0. an integer a_0, where $0 \leq a_0 \leq p_0 - 1$;

C1. a pair of integers (a, b), where $0 < a \leq p - 1$ and $0 \leq b \leq p - 1$;

C2. an ordered list $C = [c_0, c_1, \ldots, c_{k'}]$ of non-negative integers, where $k' = k/4 - 1$, $\sum_{j=0}^{k'} c_j = k$, and $\sum_{j=0}^{k'} c_j(c_j - 1) \leq 4k$. Let $C_{>1}$ be the sublist of C by removing all $c_j \leq 1$;

C3. an ordered list $L = [(a_1, b_1), (a_2, b_2), \ldots, (a_r, b_r)]$ of pairs of integers, where $0 < a_i \leq p - 1$, $0 \leq b_i \leq p - 1$, and $r \leq \log |C_{>1}|$;

C4. a mapping from the elements in the list $C_{>1}$ to the elements in the list L such that at least one half of the c_j's in $C_{>1}$ are mapped to (a_1, b_1), at least one half of the c_j's that are not mapped to (a_1, b_1) are mapped to (a_2, b_2), at least one half of the c_j's that are not mapped to (a_1, b_1) and (a_2, b_2) are mapped to (a_3, b_3), and so on.

C5. an ordered list of colorings $[f_{c_0}, f_{c_1}, \ldots, f_{c_{k'}}]$, where for each $c_j \geq 2$, f_{c_j} is a c_j-coloring from the c_j-color coding scheme $\mathcal{F}_{c_j}$ for $Z_{c_j(c_j-1)}$ in Lemma B.4 (for $c_j \leq 1$, f_{c_j} is irrelevant).

We also define two functions as follows. For an integer m, let p_m be the smallest prime number such that $m \leq p_m < 2m$. For two given integers s and a, where

$1 < s < m$ and $0 \le a \le p_m - 1$, we define a function $\psi_{a,s,m}$ from Z_m to Z_s by

$$\psi_{a,s,m}(x) = (ax \bmod p_m) \bmod s, \tag{5.2}$$

and for three given integers s, a, b, where $1 < s < m$, $0 < a \le p_m - 1$, and $0 \le b \le p_m - 1$, we define a function $\phi_{a,b,s,m}$ from the set Z_m to the set Z_s by

$$\phi_{a,b,s,m}(x) = ((ax + b) \bmod p_m) \bmod s. \tag{5.3}$$

Our k-coloring algorithm on the set Z_n is given in Figure 9.

Coloring
input: parameters as specified in **C0**–**C5**;
output: a k-coloring of the set Z_n;

0. **for each** $x \in Z_n$ **do** $\bar{x} = \psi_{a_0,k^2,n}(x)$;
1. **for** $j = 0$ **to** $k' = k/4 - 1$ **do** $U_j = \{x \mid x \in Z_n,\ \phi_{a,b,k/4,k^2}(\bar{x}) = j\}$;
2. **for each** U_j such that $c_j > 1$, suppose that in **C4**, c_j is mapped to (a_i, b_i) **do**
   for $t = 0$ **to** $c_j(c_j - 1) - 1$ **do** $U_{j,t} = \{x \mid x \in Z_n,\ \phi_{a_i,b_i,c_j(c_j-1),k^2}(\bar{x}) = t\}$;
   create c_j new colors $\tau_{j,0}$, $\tau_{j,1}$, …, τ_{j,c_j-1};
   assign all elements in $U_{j,t}$ with color $\tau_{j,s}$ if the c_j-coloring f_{c_j} in **C5** for $Z_{c_j(c_j-1)}$ assigns color s to the element t;
3. **for each** $c_j = 1$, create a new color τ_j and assign all elements in U_j the color τ_j;
4. assign all elements in $\bigcup_{c_j=0} U_j$ arbitrary colors using the colors created in steps 2–3.

Fig. 9. A coloring algorithm

We make some remarks on the algorithm **Coloring**:

(1) the function $\psi_{a_0,k^2,n}$ in step 0 is from Z_n to Z_{k^2}, so $\bar{x} \in Z_{k^2}$;

(2) the function $\phi_{a,b,k/4,k^2}$ in step 1 is from Z_{k^2} to $Z_{k/4}$, so $\phi_{a,b,k/4,k^2}(\bar{x}) \in Z_{k/4}$;

(3) for each j such that $c_j > 1$, the function $\phi_{a_i,b_i,c_j(c_j-1),k^2}$ in step 2 is from Z_{k^2} to $Z_{c_j(c_j-1)}$,

so $\phi_{a_i,b_i,c_j(c_j-1),k^2}(\bar{x}) \in Z_{c_j(c_j-1)}$;

(4) by steps 2–3, for each $c_j \ge 1$, we create c_j new colors. By **C2**, $\sum_{j=0}^{k/4-1} c_j = k$.

Therefore,

the total number of colors used by the algorithm **Coloring** is exactly k. In consequence,

the algorithm produces a k-coloring for the set Z_n.

For each given set of parameters satisfying conditions **C0**–**C5**, the algorithm **Coloring** produces a k-coloring for the set Z_n. A collection $\mathcal{F}$ of k-colorings of Z_n can be obtained by running over all possible sets of parameters that satisfy the conditions. In the next subsection, we consider the size of this collection $\mathcal{F}$.

3. The size of the collection $\mathcal{F}$

To derive an upper bound for the size of the collection $\mathcal{F}$ constructed in the previous subsection, we discuss the number of possible combinations of the parameters satisfying conditions **C0**–**C5**.

Condition **C0**: the parameter a_0 satisfies $0 \leq a_0 \leq p_0 - 1$, where $p_0 < 2n$. Therefore, there are at most $2n - 1 = O(n)$ possible values for the parameter a_0.

Condition **C1**: the parameters a and b satisfy $0 < a \leq p - 1$ and $0 \leq b \leq p - 1$, where $p < 2k^2$. Therefore, there are at most $O(k^4)$ pairs of integers (a, b) satisfying condition **C1**.

Condition **C2**: we represent each list $C = [c_0, \ldots, c_{k'}]$ satisfying condition **C2** using a single binary string B_C of length $5k/4 - 1$ in which there are exactly $k/4 - 1$ 0-bits. The $k/4 - 1$ 0-bits in B_C divide B_C into $k/4$ "segments" such that the j-th segment contains exactly c_j 1-bits (in particular, the segment between two consecutive 0's in B_C corresponds to $c_j = 0$). It is easy to verify that any list C satisfying condition **C2** is uniquely represented by such a binary string B_C. Note that the number of binary strings of length $5k/4 - 1$ with exactly $k/4 - 1$ 0-bits is equal to

$\binom{5k/4-1}{k/4-1} \leq 1.8692^k$. We conclude that the total number of different lists satisfying condition **C2** is bounded by 1.8692^k. Note that all these lists can be systematically enumerated based on the binary string representation described above.

Condition **C3**: since $r \leq \log(|C_{>1}|) \leq \log(k/4) = \log k - 2$, there are at most $\log k - 2$ pairs in each list L satisfying condition **C3**. By condition **C3**, for each pair (a_i, b_i), there are at most $O(p^2) = O(k^4)$ possible different cases. Thus, the total number of lists L satisfying condition **C3** is $O(k^{4\log k - 8})$.

Condition **C4**: now we discuss that when a list $C = [c_0, \ldots, c_{k'}]$ satisfying condition **C2** and a list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying condition **C3** are given, how many different mappings from $C_{>1}$ to L can be there that satisfy condition **C4**. Let $q = |C_{>1}| \leq k/4$. We use a binary string $A_{>1}$ to represent a mapping from $C_{>1}$ to L, as follows. The binary string $A_{>1}$ has q 0-bits, which divide $A_{>1}$ into q segments, each starting with a 0-bit. For each j, the j-th segment of form 01^{i-1} in $A_{>1}$ represents the mapping from the j-th element in $C_{>1}$ to the integer pair (a_i, b_i) in L. By Condition **C4**, at least one half of the segments in $A_{>1}$ have no 1-bit, at least one half of the remaining segments in $A_{>1}$ have the form 01, and at least one half of the remaining segments that are not of the form 0 or 01 in $A_{>1}$ have the form 011, and so on. Therefore, the length of the binary string $A_{>1}$ is bounded by

$$\frac{q}{2} + 2\frac{q}{2^2} + 3\frac{q}{2^3} + \cdots < 2q \leq \frac{k}{2}.$$

In consequence, the number of different mappings from the list $C_{>1}$ to the list L satisfying condition **C4** is bounded by $2^{k/2} = 1.4143^k$.

Condition **C5**: in the list $[f_{c_0}, f_{c_1}, \ldots, f_{c_{k'}}]$ of colorings, for each $c_j \geq 2$, f_{c_j} is a c_j-coloring from the c_j-color coding scheme $\mathcal{F}_{c_j}$ for $Z_{c_j(c_j-1)}$ given in Lemma B.4. Moreover, by Lemma B.2(1), for $c_j = 1$, we have $|\mathcal{F}_{c_j}| = 1$. Therefore, the total number of different lists $[f_{c_0}, f_{c_1}, \ldots, f_{c_{k'}}]$ satisfying condition **C5** is equal to

$\prod_{j=0}^{k'} |\mathcal{F}_{c_j}| = \prod_{c_j \geq 2} |\mathcal{F}_{c_j}|$, which, by Lemma B.4, is bounded by 2.4142^k (note that the list $[c_0, c_1, \ldots, c_{k'}]$ satisfies condition **C2**).

Summarizing the above discussion, we obtain the following theorem.

Theorem B.5 *Running the algorithm* **Coloring** *in Figure 9 over all possible parameters satisfying conditions* **C0**–**C5** *gives a collection $\mathcal{F}$ of $O(6.383^k k^{4\log k-4} n)$ k-colorings for the set Z_n. These k-colorings can be constructed in $O(6.383^k k^{4\log k-4} n^2)$ time.*

PROOF. By the above analysis, the total number of possible combinations of the parameters satisfying conditions **C0**–**C5** is bounded by

$$O(n) \cdot O(k^4) \cdot 1.8692^k \cdot O(k^{4\log k-8}) \cdot 1.4143^k \cdot 2.4142^k = O(6.383^k k^{4\log k-4} n).$$

From the above discussion, these k-colorings can be constructed systematically. Since each k-coloring of the set Z_n can be represented using n digits in Z_k, the collection $\mathcal{F}$ can be constructed in time $O(6.383^k k^{4\log k-4} n^2)$. □

4. The collection $\mathcal{F}$ is a k-color coding scheme for Z_n

We derive in this subsection that the collection $\mathcal{F}$ of k-colorings for the set Z_n in Theorem B.5 is a k-color coding scheme for the set Z_n. For this, we need to prove that for any subset W of k elements in Z_n, there is a combination of parameters satisfying conditions **C0**–**C5** on which the algorithm **Coloring** produces a k-coloring for the set Z_n that is injective from W. In the following discussion, we fix a subset $W = \{w_1, \ldots, w_k\}$ of Z_n.

First we consider the function $\psi_{a,s,n}$ used in step 0 of the algorithm **Coloring**. The function $\psi_{a,s,n}$ has been thoroughly studied for its use in the construction of

universal hashing functions [54, 108]. This is the construction suggested in [3] for the implementation of k-color coding schemes. In particular, the following result holds:

Theorem B.6 ([54]) *For the subset W of k elements in Z_n, there is an integer a_0, $0 \leq a_0 \leq p_0 - 1$, such that the function $\psi_{a_0,k^2,n}$ is injective from W.*

Therefore, there is an integer a_0 satisfying condition **C0** such that if we let $\overline{w}_i = \psi_{a_0,k^2,n}(w_i)$ for all $1 \leq i \leq k$, then the set $\overline{W} = \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_k\}$ is a subset of k elements in Z_{k^2}. For each $1 \leq i \leq k$, define a subset W_i of Z_n by $W_i = \{x \mid x \in Z_n \ \& \ \psi_{a_0,k^2,n}(x) = \overline{w}_i\}$. Let $\mathcal{W} = \{W_1, \ldots, W_k\}$ be the collection of these subsets.

Now consider the function ϕ_{a,b,s,k^2} used in steps 1–2 in the algorithm **Coloring**, which has been studied by Carter and Wegman for other purposes [19]. To discuss its use in the construction of k-color coding schemes, we first study some basic properties of the function.

Consider the following two sets of ordered pairs of integers (recall that p is a prime number satisfying $k^2 < p < 2k^2$):

$$\begin{aligned} F_1(p) &= \{(a,b) \mid 0 < a \leq p-1 \text{ and } 0 \leq b \leq p-1\}, \\ F_2(p) &= \{(r,q) \mid 0 \leq r,q \leq p-1 \text{ and } r \neq q\}. \end{aligned}$$

Fix two distinct integers x and y, $0 \leq x, y \leq p-1$, we construct a mapping as follows:

$$\pi_{x,y}: \ \ (a,b) \longrightarrow ((ax+b) \bmod p, \ \ (ay+b) \bmod p).$$

We have the following lemma.

Lemma B.7 *For any two integers x and y such that $0 \leq x, y \leq p-1$ and $x \neq y$, the mapping $\pi_{x,y}$ is a one-to-one mapping from $F_1(p)$ to $F_2(p)$.*

PROOF. Since p is a prime number, the set Z_p is a field in terms of addition

and multiplication modulo p. For a pair (a, b) in $F_1(p)$, from $(ax + b) \bmod p = (ay + b) \bmod p$, we would get $x = y$ (recall that p is a prime and $a \neq 0$). Therefore, the mapping $\pi_{x,y}$ maps each element in $F_1(p)$ to an element in $F_2(p)$. Moreover, for a pair (r, q) in $F_2(p)$, where $r \neq q$, the linear equations $(ax + b) \bmod p = r$ and $(ay + b) \bmod p = q$ have a unique solution (a, b), where $a, b \in Z_p$ and $a \neq 0$, i.e., $(a, b) \in F_1(p)$. The lemma now follows directly from the fact that both sets $F_1(p)$ and $F_2(p)$ have exactly $p(p-1)$ elements. □

Let $0 < a \leq p-1$, $0 \leq b \leq p-1$, and $1 < s \leq k^2$. For the subset $\overline{W}$ of k elements in Z_{k^2} and for each integer j, $0 \leq j \leq s-1$, denote by $B(a, b, s, \overline{W}, j)$ the number of integers x in $\overline{W}$ such that $\phi_{a,b,s,k^2}(x) = j$. We have the following lemma.

Lemma B.8 *Suppose that* $p \bmod s = h$. *Then for the subset* $\overline{W}$ *of* k *elements in* Z_{k^2}, *we have*

$$\sum_{(a,b)\in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a, b, s, \overline{W}, j)}{2} = \frac{k(k-1)(p-h)(p-(s-h))}{2s}. \tag{5.4}$$

Proof. Let $p = gs + h$, where g is an integer. Then $Z_p = \{0, 1, \ldots, gs + h - 1\}$. Let

$$A_0 = \sum_{(a,b)\in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a, b, s, \overline{W}, j)}{2} = \sum_{j=0}^{s-1} \sum_{(a,b)\in F_1(p)} \binom{B(a, b, s, \overline{W}, j)}{2}.$$

The value A_0 is equal to the number of different ways of picking an ordered pair (a, b) in $F_1(p)$, and two different elements x and y in $\overline{W}$ such that $\phi_{a,b,s,k^2}(x) = \phi_{a,b,s,k^2}(y)$, or equivalently

$$((ax + b) \bmod p) \bmod s = ((ay + b) \bmod p) \bmod s.$$

By Lemma B.7, for two different elements x and y in $\overline{W}$, the mapping

$$\pi_{x,y}: \ \ (a,b) \longrightarrow ((ax+b) \bmod p, \ \ (ay+b) \bmod p)$$

is a one-to-one mapping from $F_1(p)$ to $F_2(p)$. Therefore, the value A_0 is also equal to the number of different ways of picking an ordered pair (r,q) in $F_2(p)$ and two different elements x and y in $\overline{W}$ such that

$$r \bmod s = q \bmod s.$$

The number of different ways to pick two different elements x and y in $\overline{W}$ is equal to $k(k-1)/2$. Therefore, the value A_0 is equal to $k(k-1)/2$ times the number of different ways of picking an ordered pair (r,q) in $F_2(p)$ such that $r \bmod s = q \bmod s$.

For each j, if $0 \le j \le h-1$, there are $g+1$ elements q in Z_p such that $q \bmod s = j$; while if $h \le j \le s-1$, there are g elements q in Z_p such that $q \bmod s = j$. Therefore, for each j, $0 \le j \le h-1$, there are $g(g+1)$ ordered pairs (r,q) in $F_2(p)$ such that $r \bmod s = q \bmod s = j$; while for each j, $h \le j \le s-1$, there are $g(g-1)$ ordered pairs (r,q) in $F_2(p)$ such that $r \bmod s = q \bmod s = j$. In summary, there are totally $hg(g+1) + (s-h)g(g-1) = g(p-(s-h)) = (p-h)(p-(s-h))/s$ ordered pairs (r,q) in $F_2(p)$ such that $r \bmod s = q \bmod s$.

Therefore, we have proved

$$A_0 = \sum_{(a,b) \in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a,b,s,\overline{W},j)}{2} = \frac{k(k-1)(p-h)(p-(s-h))}{2s}.$$

This completes the proof of the lemma. □

Corollary B.9 *Let $1 < s \le k^2$. For the subset $\overline{W}$ of k elements in the set Z_{k^2}, there*

is an ordered pair (a, b) *in* $F_1(p)$ *such that*

$$\sum_{j=0}^{s-1} \binom{B(a,b,s,\overline{W},j)}{2} < \frac{k(k-1)}{2s}.$$

PROOF. Since there are exactly $p(p-1)$ ordered pairs in $F_1(p)$, from Lemma B.8, there is at least one ordered pair (a, b) in $F_1(p)$ such that

$$\sum_{j=0}^{s-1} \binom{B(a,b,s,\overline{W},j)}{2} \leq \frac{k(k-1)(p-h)(p-(s-h))}{2sp(p-1)}.$$

Since $p = gs+h$ is a prime number, we have $1 \leq h \leq s-1$ and $1 \leq s-h \leq s-1$. Therefore, both $(p-h)$ and $(p-(s-h))$ are not larger than $p-1$. In particular, $(p-h)(p-(s-h))$ is strictly smaller than $p(p-1)$. Now the corollary follows. □

We remark that Corollary B.9 gives a significant improvement over the bound given in [54], which is the bound used to implement the color coding scheme suggested in [3]. In particular, the bound derived in [54] uses the function ψ_{a,s,k^2}, and the corresponding bound is k^2/s. We will see that the bound improvement from k^2/s to $k(k-1)/(2s)$ will induce a very significant improvement on the size of k-color coding schemes.

Lemma B.10 *For the subset* $\overline{W}$ *of* k *elements in* Z_{k^2}*, there is a pair* (a, b) *in* $F_1(p)$ *such that*

$$\sum_{j=0}^{k/4-1} B(a,b,k/4,\overline{W},j)(B(a,b,k/4,\overline{W},j)-1) < 4k.$$

PROOF. Let $s = k/4$. From Corollary B.9, there is a pair (a, b) in $F_1(p)$, such that

$$\sum_{j=0}^{k/4-1} \binom{B(a,b,k/4,\overline{W},j)}{2} < 2(k-1),$$

which directly implies the lemma. □

Corollary B.11 *For the subset $\overline{W}$ of k elements in Z_{k^2}, there is a pair (a, b) satisfying condition* **C1**, *such that if we let $\overline{W}_j = \{\overline{w} \mid \overline{w} \in \overline{W}\ \&\ \phi_{a,b,k/4,k^2}(\overline{w}) = j\}$ and let $c_j = |\overline{W}_j|$ for all $0 \le j \le k' = k/4 - 1$, then the list $C = [c_0, \ldots, c_{k'}]$ satisfies condition* **C2**.

Since each element w_i in the set W corresponds uniquely to an element $\overline{w}_i$ in the set $\overline{W}$, according to Corollary B.11, there is a pair (a, b) satisfying condition **C1**, such that each set U_j, $0 \le j \le k' = k/4 - 1$, constructed in step 1 of the algorithm **Coloring** contains exactly c_j elements in W, and the list $C = [c_0, \ldots, c_{k'}]$ satisfies condition **C2**.

Lemma B.12 *Let $\overline{\mathcal{W}}$ be a collection of some of the subsets $\overline{W}_j$ with $c_j = |\overline{W}_j| > 1$, as given in Corollary B.11. Then there is a pair (a, b) satisfying condition* **C1** *such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}$, the function $\phi_{a,b,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$.*

Proof. Fix a subset $\overline{W}_j$ in $\overline{\mathcal{W}}$, where $c_j = |\overline{W}_j| > 1$. Applying Lemma B.8 to $\overline{W}_j$ and let $s = c_j(c_j - 1)$ (where we have let $p \bmod s = h > 0$), we get:

$$\begin{aligned}\sum_{(a,b)\in F_1(p)} \sum_{i=0}^{s-1} \binom{B(a,b,s,\overline{W}_j,i)}{2} &= \frac{c_j(c_j-1)(p-h)(p-(s-h))}{2s} \\ &< \frac{c_j(c_j-1)p(p-1)}{2s} = \frac{p(p-1)}{2}.\end{aligned}$$

Since the set $F_1(p)$ totally has $p(p-1)$ pairs, the above relation shows that for at least one half of the pairs (a, b) in $F_1(p)$, the equality

$$\sum_{i=0}^{s-1} \binom{B(a,b,s,\overline{W}_j,i)}{2} = 0$$

holds, i.e., $B(a, b, s, \overline{W}_j, i) \le 1$ for all i. Therefore, for at least one half of the pairs

(a, b) in $F_1(p)$, the function $\phi_{a,b,c_j(c_j-1),k^2}$ is injective from the subset $\overline{W}_j$. Applying this analysis to each subset $\overline{W}_j$ in $\overline{\mathcal{W}}$ and using a simple counting argument, we derive that there is at least one pair (a, b) in $F_1(p)$ (i.e., a pair (a, b) satisfying condition **C1**) such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}$, the function $\phi_{a,b,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$. □

Corollary B.13 *Let $\overline{\mathcal{W}}_{>1}$ be the collection of all subsets $\overline{W}_j$ in Corollary B.11 with $c_j = |\overline{W}_j| > 1$. Then there is an ordered list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying condition* **C3** *such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_1,b_1,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$, for at least one half of the remaining subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_2,b_2,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$, and for at least one half of the remaining subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_3,b_3,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$, and so on.*

Proof. Applying Lemma B.12 to $\overline{\mathcal{W}}_{>1}$, we get a pair (a_1, b_1) satisfying condition **C1** such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_1,b_1,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$. Let $\overline{\mathcal{W}}'_{>1}$ be the remaining subsets in $\overline{\mathcal{W}}_{>1}$. Applying Lemma B.12 to $\overline{\mathcal{W}}'_{>1}$, we get a pair (a_2, b_2) satisfying condition **C1** such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}'_{>1}$, the function $\phi_{a_2,b_2,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$; and so on. This process stops after at most $r \leq \log q$ steps, where q is the total number of subsets in $\overline{\mathcal{W}}_{>1}$. The list of pairs $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ constructed this way satisfies condition **C3**. □

From Corollary B.13, we get immediately

Corollary B.14 *Let $\overline{W}_j$ be the subset, $0 \leq j \leq k'$, as given in Corollary B.11, $c_j = |\overline{W}_j|$, and $C = [c_0, \ldots, c_{k'}]$. Then there is a list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying condition* **C3** *and a mapping from $C_{>1}$ to L satisfying condition* **C4**, *such*

that for all j, if $c_j > 1$ is mapped to (a_i, b_i), then the function $\phi_{a_i,b_i,c_j(c_j-1),k^2}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$.

Therefore, for the pair (a', b') and the list $C' = [c'_0, \ldots, c'_{k'}]$ in Corollary B.11, which satisfy conditions **C1** and **C2**, respectively, and for the list $L' = [(a'_1, b'_1), \ldots, (a'_r, b'_r)]$ and the mapping from $C'_{>1}$ to L' in Corollary B.14, which satisfy conditions **C3** and **C4**, respectively, each function $\phi_{a'_i,b'_i,c'_j(c'_j-1),k^2}$ is injective from the subset $\overline{W}_j$ to $Z_{c'_j(c'_j-1)}$ for all j. For each j, let W'_j be the image of $\overline{W}_j$ under the function $\phi_{a'_i,b'_i,c'_j(c'_j-1),k^2}$, then $W'_j \subseteq Z_{c'_j(c'_j-1)}$ and $|W'_j| = c'_j$. Now since $\mathcal{F}_{c'_j}$ is a c'_j-color coding scheme for the set $Z_{c'_j(c'_j-1)}$, one $f_{c'_j}$ of the c'_j-colorings in $\mathcal{F}_{c'_j}$ is injective from W'_j. According to the algorithm **Coloring**, when this c'_j-coloring $f_{c'_j}$ is used for the algorithm, the c'_j elements in W whose images are in $\overline{W}_j$ are colored with distinct colors. Running this for all j, we conclude that there is a list $[f_{c'_0}, f_{c'_1}, \ldots, f_{c'_{k'}}]$, where $f_{c'_j}$ is a c'_j-coloring in the c'_j-color coding scheme $\mathcal{F}_{c'_j}$, satisfying condition **C5** such that all elements in the subset W are colored with distinct colors.

Summarizing the above discussion, we conclude

Theorem B.15 *For each subset W of k elements in Z_n, there is a combination of parameters satisfying conditions* **C0**–**C5** *on which the algorithm* **Coloring** *produces a k-coloring for Z_n that is injective from W.*

Combining Theorem B.15 with Theorem B.5, and let $n = k^2$, we get

Theorem B.16 *Let k be an integer divisible by 4. There is a k-color coding scheme of size $O(6.383^k k^{4\log k - 2})$ for the set Z_{k^2}, which can be constructed in time $O(6.383^k k^{4\log k})$.*

Using Theorem B.6, we can easily extend Theorem B.16 to general values of n.

Theorem B.17 *For any integer n, and an integer k divisible by 4, where $n \geq k$, there is a k-color coding scheme of size $O(6.383^k k^{4\log k-2} n)$ for the set Z_n, which can be constructed in time $O(6.383^k k^{4\log k-2} n^2)$.*

5. Extension to general k

To extend Theorem B.17 to general values of k, suppose that $k = 4k' - h$, where $1 \leq h \leq 3$. We first construct a $(4k')$-color coding scheme $\mathcal{F}'$ of size

$$O(6.383^{4k'}(4k')^{4\log(4k')-2} n) = O(6.383^k k^{4\log(k+h)-2} n)$$

for the set Z_n. Now for each $(4k')$-coloring f in $\mathcal{F}'$, we construct $\binom{4k'}{h} = O(k^3)$ k-colorings for Z_n by selecting every subset of h colors in f and replacing them arbitrarily by the remaining $k = 4k' - h$ colors. This gives a collection $\mathcal{F}$ of

$$O(k^3 6.383^k k^{4\log(k+h)-2} n) = O(6.4^k n)$$

k-colorings for the set Z_n. To see that this is a k-color coding scheme for the set Z_n, let W be any subset of k elements in Z_n. Let W' be a subset of $4k'$ elements in Z_n obtained from W by adding arbitrarily h elements. Since $\mathcal{F}'$ is a $(4k')$-color coding scheme for Z_n, there is a $(4k')$-coloring f' in $\mathcal{F}'$ that is injective from W'. In particular, this $(4k')$-coloring f' is also injective from W. Now the k-coloring f in $\mathcal{F}$ obtained from f' by removing the other h colors is injective from W. This shows that the collection $\mathcal{F}$ is a k-color coding scheme for the set Z_n. This proves the main result of this section, as given in the following theorem.

Theorem B.18 *For any integers n and k, where $n \geq k$, there is a k-color coding scheme of size $O(6.4^k n)$ for the set Z_n, which can be constructed in time $O(6.4^k n^2)$.*

C. An Improved Matching Algorithm

In this section, we present an improved algorithm for the P-3-D MATCHING problem. The improved algorithm is based on a method that will be named *iterative expansion*. The method is developed based on a structural property of P-3-D MATCHING that is a significant improvement over the one used in the previous greedy localization method [22, 66], on the improved color coding scheme presented in the previous section, and on a new dynamic programming process that significantly improves those suggested in the literature for solving P-3-D MATCHING [29, 50].

A set M of points in the 3-dimensional Euclidean space $\mathbb{R}^3$ is a *matching* if no two points in M agree in any coordinate. A *k-matching* is a matching consisting of exactly k points in $\mathbb{R}^3$. Each point $t = (x, y, z)$ in $\mathbb{R}^3$ is called a *triple*. For a triple $t = (x, y, z)$ in $\mathbb{R}^3$, denote by $\mathrm{Val}(t)$ the set $\{x, y, z\}$, and let $\mathrm{Val}^1(t) = \{x\}$, $\mathrm{Val}^2(t) = \{y\}$, and $\mathrm{Val}^3(t) = \{z\}$. Let S be a set of triples in $\mathbb{R}^3$. Denote $\mathrm{Val}(S) = \bigcup_{t \in S} \mathrm{Val}(t)$, and $\mathrm{Val}^i(S) = \bigcup_{t \in S} \mathrm{Val}^i(t)$ for $i = 1, 2, 3$.

The P-3-D MATCHING problem is formally defined as follows.

> P-3-D MATCHING: Given a set S of n points in the 3-dimensional Euclidean space $\mathbb{R}^3$ and a parameter k, either construct a k-matching in S, or report that no k-matching exists in S.

Instead of working on this problem, we will concentrate on the following related problem.

> 3-D MATCHING AUGMENTATION: Given a pair (S, M_k), where S is a set of n triples, and M_k is a k-matching in S, either construct a $(k+1)$-matching in S, or report that no such a matching exists.

The following lemma shows how the problems P-3-D MATCHING and 3-D MATCH-

ING AUGMENTATION are related.

Lemma C.1 *For any constant $c > 1$,* 3-D MATCHING AUGMENTATION *is solvable in time $O^*(c^k)$ if and only if* P-3-D MATCHING *is solvable in time $O^*(c^k)$.*

PROOF. Suppose that the original P-3-D MATCHING problem can be solved in time $O^*(c^k)$. Then for an instance (S, M_k) of 3-D MATCHING AUGMENTATION, we simply solve the instance $(S, k+1)$ of P-3-D MATCHING in time $O^*(c^{k+1}) = O^*(c^k)$, which gives a correct solution to the instance (S, M_k) of 3-D MATCHING AUGMENTATION.

On the other hand, suppose that 3-D MATCHING AUGMENTATION can be solved in time $O^*(c^k)$ by an algorithm A. For a given instance (S, k) of P-3-D MATCHING, we trivially pick a matching M_1 of one triple in S, then iteratively apply the algorithm A to the instance (S, M_i) for each i, where M_i is an i-matching, to get an $(i+1)$-matching M_{i+1} in S. If the algorithm A reports that "S has no $(i+1)$-matching" on an instance (S, M_i) for some $i \leq k-1$, then obviously there is no k-matching in the set S. On the other hand, the algorithm A on the instance (S, M_{k-1}) will construct a k-matching for the set S. The running time of this process is bounded by

$$O^*(c^1) + O^*(c^2) + \cdots + O^*(c^{k-1}) = O^*(c^k).$$

Therefore, the P-3-D MATCHING problem can be solved in time $O^*(c^k)$. □

By Lemma C.1, we can focus on 3-D MATCHING AUGMENTATION. Note that the formulation of 3-D MATCHING AUGMENTATION provides the basis for iterative expansion, where we iteratively expand a given solution to obtain a larger solution until we get a solution of the desired size. In the rest of this section, we study the problem structures and algorithmic techniques that make the iterative expansion process efficient for 3-D MATCHING AUGMENTATION.

Lemma C.2 *Let (S, M_k) be an instance of* 3-D MATCHING AUGMENTATION*, where M_k is a k-matching in the triple set S. If S also has $(k+1)$-matchings, then there exists a $(k+1)$-matching M_{k+1} in S such that every triple in M_k contains at least two symbols in Val(M_{k+1}).*

PROOF. We prove the lemma by contradiction. Suppose that the lemma does not hold. Then for every $(k+1)$-matching M in S, there is a triple in M_k that contains at most one symbol in Val(M). Let M_{k+1} be a $(k+1)$-matching in S such that the number of common triples in M_k and M_{k+1} is maximized over all $(k+1)$-matchings in S.

By our assumption, there is a triple ρ in M_k that contains at most one symbol in Val(M_{k+1}). We distinguish two different cases.

Case 1. Exactly one symbol a in the triple ρ is in Val(M_{k+1}). Then let ρ' be the triple in M_{k+1} that contains the symbol a. Since no other symbol in ρ is in Val(M_{k+1}), if we replace the triple ρ' in M_{k+1} by the triple ρ, we get a new $(k+1)$-matching that has one more common triple (i.e., the triple ρ) with the k-matching M_k (note that the triple ρ' cannot be in M_k because ρ' and ρ share a common symbol a while ρ contains another two symbols not in Val(M_{k+1})). This contradicts our assumption that the $(k+1)$-matching M_{k+1} maximizes the number of common triples with M_k.

Case 2. No symbol in ρ is in Val(M_{k+1}). Since M_k contains k triples while M_{k+1} contains $k+1$ triples, there must be a triple ρ'' in M_{k+1} that is not in M_k. Since ρ contains no symbol in Val(M_{k+1}), replacing ρ'' in M_{k+1} by ρ gives a new $(k+1)$-matching that has one more common triple (i.e., the triple ρ) with M_k, again contradicting the assumption that the $(k+1)$-matching M_{k+1} maximizes the number of common triples with M_k.

This contradiction shows that the triple ρ in M_k that contains at most one symbol

in $\mathrm{Val}(M_{k+1})$ cannot exist. The lemma then follows. □

Lemma C.2 significantly improves a result given in [22], which observed that for each k-matching M_k in S, there is a $(k+1)$-matching M_{k+1} (if $(k+1)$-matchings exist in S) such that every triple in M_k contains at least one symbol in M_{k+1}. More importantly, the result of Lemma C.2 significantly narrows down the search space when we look for the $(k+1)$-matching M_{k+1}: there are at most $k+3$ symbols in $\mathrm{Val}(M_{k+1})$ that are not in $\mathrm{Val}(M_k)$.

Definition Let S be a triple set, let p and q be any two indices in the index set $\{1,2,3\}$, and let f be a coloring of the set $\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)$. A matching M in the set S is (p,q)-*properly colored* by f if no two symbols in $\mathrm{Val}^p(M) \cup \mathrm{Val}^q(M)$ are colored with the same color under f.

In the following, we present a dynamic programming process that constructs a (p,q)-properly colored k-matching. This process is an improvement over those proposed in previous work on P-3-D MATCHING. In previous work [50, 75], it is known that if a coloring f of the set $\mathrm{Val}(S)$ is given such that the symbols in $\mathrm{Val}(M)$ are colored properly by f for some k-matching M, then a k-matching can be constructed by a dynamic programming process. On the other hand, our next lemma shows that as long as the symbols in any two columns of a k-matching are properly colored, a k-matching can be constructed by a dynamic programming process.

Theorem C.3 *Let p and q be two indices in the index set $\{1,2,3\}$. There is an algorithm of running time $O^*(\sum_{i=0}^{k} \binom{g}{2i})$ that, on an integer k, a triple set S, and a g-coloring f on the set $Val^p(S) \cup Val^q(S)$, constructs a (p,q)-properly colored k-matching in S if such a matching exists.*

Algorithm 3DMatch$(S, k, f, g; p, q)$
input: A triple set S, an integer k, a g-coloring f of the symbols in $\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)$
output: A (p, q)-properly colored k-matching in S if such a matching exists

1. remove any triples in S in which any two symbols have the same color under f;
2. let the set of remaining triples be S';
3. $r = \{1, 2, 3\} - \{p, q\}$;
4. let the symbols in $\mathrm{Val}^r(S')$ be x_1, x_2, ..., x_m;
5. $\mathcal{Q}_{old} = \{\emptyset\}$; $\mathcal{Q}_{new} = \{\emptyset\}$;
6. **for** $i = 1$ **to** m **do**
6.1. **for** each set C of symbol pairs in $\mathcal{Q}_{old}$ **do**
6.2. **for** each triple t in S' with $\mathrm{Val}^r(t) = x_i$ **do**
6.3. **if** no symbol in C is of the same color as a symbol in $\mathrm{Val}^p(t) \cup \mathrm{Val}^q(t)$ **then**
6.4. $C' = C \cup \{(\mathrm{Val}^p(t), \mathrm{Val}^q(t))\}$;
6.5. **if** C' contains no more than k symbol pairs and $\mathcal{Q}_{new}$ contains no set of symbol pairs that uses exactly the same colors as that used by C' **then**
6.6. add C' to $\mathcal{Q}_{new}$;
6.7. $\mathcal{Q}_{old} = \mathcal{Q}_{new}$;
7. **if** $\mathcal{Q}_{old}$ contains a set C_k of k symbol pairs **then**
extend C_k to a (p, q)-properly colored k-matching M_k, and **return** M_k.

Fig. 10. Dynamic programming for the P-3-D MATCHING problem

PROOF. Consider the algorithm in Figure 10. By steps 6.3–6.6 of the algorithm, for every set C in the collection $\mathcal{Q}_{old}$, all symbols in C are from $\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)$, and no two symbols in C are of the same color. We say that a set $C = \{w_1, \ldots, w_h\}$ of h symbol pairs is *extendable to an h-matching* in S if there is an h-matching $M = \{t_1, \ldots, t_h\}$ in S such that for all j, the pair $(\mathrm{Val}^p(t_j), \mathrm{Val}^q(t_j))$ is identical to the symbol pair w_j. For each i, let S'_i be the set of triples t in S' such that $\mathrm{Val}^r(t)$ is in $\{x_1, x_2, \ldots, x_i\}$. For a matching M, denote by $cl(M)$ the set of colors used by the symbols in $\mathrm{Val}^p(M) \cup \mathrm{Val}^q(M)$.

We prove the following claim by induction on i:

Claim. For each i, $0 \le i \le m$, and for all $h \le k$, there is a (p, q)-properly

colored h-matching M_h in S'_i if and only if after the i-th execution of the loop 6.1–6.7 of algorithm **3DMatch**$(S, k, f, g; p, q)$, the collection $\mathcal{Q}_{old}$ contains a set C_h of h symbol pairs such that the set of colors used for the symbols in C_h is exactly the set $cl(M_h)$. Moreover, each set C_h of h symbol pairs in the collection $\mathcal{Q}_{old}$ after the i-th execution of the loop 6.1–6.7 is extendable to a (p, q)-properly colored h-matching in S'_i.

The case $i = 0$ is obvious because we initially set $\mathcal{Q}_{old}$ to $\{\emptyset\}$. Now we consider a general case for $i \geq 1$. First note that the claim is always true for $h = 0$ because the collection $\mathcal{Q}_{old}$ always contains the empty set $\emptyset$ while the set S'_i always contains a 0-matching (which by definition is (p, q)-properly colored).

Suppose that after the i-th execution of the loop 6.1–6.7, the collection $\mathcal{Q}_{old}$ contains a set C_h of h symbol pairs, where $h \geq 1$. Suppose that the set C_h was created during the j-th execution of the loop 6.1–6.7, where $j \leq i$, by adding a symbol pair $(\mathrm{Val}^p(t), \mathrm{Val}^q(t))$ to a set C_{h-1} of $h - 1$ symbol pairs, where t is a triple with $\mathrm{Val}^r(t) = x_j$ and the set C_{h-1} is contained in $\mathcal{Q}_{old}$ after the $(j - 1)$-st execution of the loop 6.1–6.7. By the inductive hypothesis, the set C_{h-1} is extendable to an $(h - 1)$-matching M_{h-1} in S'_{j-1}, which is obviously (p, q)-properly colored. Since no symbol in C_{h-1} uses the same color as a symbol in $\mathrm{Val}^p(t) \cup \mathrm{Val}^q(t)$, and the matching M_{h-1} does not contain the symbol x_j, the set $M_h = M_{h-1} \cup \{t\}$ makes a (p, q)-properly colored h-matching in S'_j. Since $j \leq i$ and $S'_j \subseteq S'_i$, we conclude that the set S'_i contains a (p, q)-properly colored h-matching M_h such that the symbols in the set C_h use exactly the color set $cl(M_h)$. Moreover, it is obvious that the symbol set C_h is extendable to the h-matching M_h.

To prove the other direction, suppose that the set S'_i contains a (p, q)-properly colored h-matching M_h. We distinguish two different cases.

Case 1. There is a (p, q)-properly colored h-matching M'_h in S'_j for some $j < i$ such that $cl(M'_h) = cl(M_h)$. In this case, by the inductive hypothesis, after the j-th execution of the loop 6.1–6.7, the collection $\mathcal{Q}_{old}$ contains a set C_h of h symbol pairs such that (1) the set of colors used for the symbols of C_h is exactly the set $cl(M'_h)$; and (2) C_h is extendable to an h-matching in S'_j. Since $j < i$, $S'_j \subseteq S'_i$, and we never remove symbol pairs from $\mathcal{Q}_{old}$, we conclude that in this case, after the i-th execution of the loop 6.1–6.7, the set C_h is still contained in the collection $\mathcal{Q}_{old}$ such that (1) the set of colors used for the symbols of C_h is exactly the set $cl(M'_h) = cl(M_h)$; and (2) C_h is extendable to an h-matching in $S'_j \subseteq S'_i$.

Case 2. There is no (p, q)-properly colored h-matching M'_h in S'_j for any $j < i$ such that $cl(M'_h) = cl(M_h)$. Then by the inductive hypothesis, after the j-th execution of the loop 6.1–6.7 for any $j < i$, the collection $\mathcal{Q}_{old}$ contains no set C of symbol pairs such that the symbols in C use exactly the color set $cl(M_h)$. Let the (p, q)-properly colored h-matching M_h be $M_h = \{t_1, \ldots, t_h\}$, where for each j, $\mathrm{Val}^r(t_j) = x_{d_j}$, with $d_1 < \cdots < d_{h-1} < d_h$. In this case, we must have $d_h = i$ and $\mathrm{Val}^r(t_h) = x_i$. Let $y = \mathrm{Val}^p(t_h)$ and $z = \mathrm{Val}^q(t_h)$. Since $d_{h-1} < d_h = i$, the triple set $M_{h-1} = M_h - \{t_h\}$ is a (p, q)-properly colored $(h-1)$-matching in S'_{i-1}. By the inductive hypothesis, after the $(i-1)$-st execution of the loop 6.1–6.7 in the algorithm, the collection $\mathcal{Q}_{old}$ contains a set C_{h-1} of $h-1$ symbol pairs such that the set of colors used for the symbols in C_{h-1} is exactly the set $cl(M_{h-1})$. Now in the i-th execution of the loop 6.1–6.7 when the set C_{h-1} and the triple t_h are examined in step 6.3, a set C of symbol pairs using the color set $cl(M_{h-1}) \cup \{f(y), f(z)\} = cl(M_h)$ will be created. Therefore, after the i-th execution of the loop 6.1–6.7, a set C_h of h symbol pairs using the color set $cl(M_h)$ must be contained in the collection $\mathcal{Q}_{old}$. Suppose that the set C_h was created during the i-th execution by adding a symbol pair $(\mathrm{Val}^p(t'_h), \mathrm{Val}^q(t'_h))$ to a set C'_{h-1} of $h-1$ symbol pairs, where the triple t'_h satisfies $\mathrm{Val}^r(t'_h) = x_i$ and C'_{h-1} is

contained in the collection $\mathcal{Q}_{old}$ after the $(i-1)$-st execution of the loop 6.1–6.7 (note that t'_h and C'_{h-1} are not necessarily t_h and C_{h-1}, respectively). By the inductive hypothesis, the set C'_{h-1} is extendable to a (p,q)-properly colored $(h-1)$-matching M'_{h-1} in S'_{i-1}. In consequence, the set C_h is extendable to the (p,q)-properly colored h-matching $M'_{h-1} \cup \{t'_h\}$ in S'_i. This completes the proof of the claim.

By the claim and let $i = m$, we conclude that when the algorithm **3DMatch**$(S, k, f, g; p, q)$ reaches step 7, the collection $\mathcal{Q}_{old}$ contains a set C_k of k symbol pairs if and only if the triple set S contains a (p,q)-properly colored k-matching M_k, and C_k is extendable to a (p,q)-properly colored k-matching. To construct such a k-matching from C_k, we can use a method suggested in [22]. Formally, from the set C_k of symbol pairs, we construct a bipartite graph $B_k = (V_L \cup V_R, E)$, where V_L contains k vertices, corresponding to the k symbol pairs in C_k, and V_R is the set of all symbols in $\mathrm{Val}^r(S)$. There is an edge in B_k from a vertex (y,z) in V_L to a vertex x in V_R if and only if the symbols y, z, and x form a triple in S. It is easy to see that a (p,q)-properly colored k-matching in S extended from C_k corresponds to a graph matching of k edges in the graph B_k, which can be constructed in polynomial time [31]. This completes the proof for the correctness of the algorithm.

For the time complexity, note that the collection $\mathcal{Q}_{old}$ keeps at most one set of symbol pairs for each set of $2i$ colors, thus, totally contains at most $\sum_{i=0}^{k} \binom{g}{2i}$ sets of symbol pairs. For each set C of symbol pairs in $\mathcal{Q}_{old}$, steps 6.2–6.6 run in polynomial time. Therefore, each execution of the loop 6.1–6.7 of the algorithm runs in time $O^*(\sum_{i=0}^{k} \binom{g}{2i})$. In consequence, the running time of the algorithm **3DMatch**$(S, k, f, g; p, q)$ is bounded by $O^*(\sum_{i=0}^{k} \binom{g}{2i})$. □

The next lemma shows that when compared to previously proposed color coding methods for P-3-D MATCHING, it is much more efficient to construct a collection

of colorings such that some $(k+1)$-matching becomes (p,q)-properly colored by a coloring in the collection.

Lemma C.4 *Let (S, M_k) be an instance of* 3-D MATCHING AUGMENTATION, *where S is a triple set and M_k is a k-matching in S. There are two indices $p, q \in \{1,2,3\}$ and a collection $\mathcal{C}$ of $O^*(6.4^{2k/3})$ $(\lceil 8k/3+2 \rceil)$-colorings for the set $Val^p(S) \cup Val^q(S)$ such that if S contains $(k+1)$-matchings, then at least one $(k+1)$-matching of S is (p,q)-properly colored by a coloring in $\mathcal{C}$. Moreover, the collection $\mathcal{C}$ can be constructed in time $O^*(6.4^{2k/3})$.*

PROOF. Suppose that the triple set S has $(k+1)$-matchings. By Lemma C.2, there is a $(k+1)$-matching M_{k+1} in S such that each triple in the k-matching M_k contains at least two symbols in M_{k+1}. In particular, there are two indices p and q in $\{1,2,3\}$, such that at least $\lceil 4k/3 \rceil$ symbols in the set $\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k)$ are contained in the set $\mathrm{Val}^p(M_{k+1}) \cup \mathrm{Val}^q(M_{k+1})$. Since the set $\mathrm{Val}^p(M_{k+1}) \cup \mathrm{Val}^q(M_{k+1})$ has exactly $2k+2$ symbols, at most $\lceil 2k/3+2 \rceil$ symbols in $\mathrm{Val}^p(M_{k+1}) \cup \mathrm{Val}^q(M_{k+1})$ are missing in the set $\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k)$. Let D be the set of symbols in $(\mathrm{Val}^p(M_{k+1}) \cup \mathrm{Val}^q(M_{k+1})) - (\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k))$, then $|D| \leq \lceil 2k/3+2 \rceil$. Let D_0 be any set of $\lceil 2k/3+2 \rceil$ symbols in $(\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)) - (\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k))$ such that $D \subseteq D_0$.

According to Theorem B.18, we can construct, in $O^*(6.4^{\lceil 2k/3+2 \rceil}) = O^*(6.4^{2k/3})$ time, a collection $\mathcal{C}_0$ of $O^*(6.4^{2k/3})$ $(\lceil 2k/3+2 \rceil)$-colorings for the set $(\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)) - (\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k))$ in which at least one $(\lceil 2k/3+2 \rceil)$-coloring f_0 properly colors the set D_0. In particular, f_0 also properly colors the set D. Each $(\lceil 2k/3+2 \rceil)$-coloring f in $\mathcal{C}_0$ induces a $(\lceil 8k/3+2 \rceil)$-coloring on the set $\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)$, in which each of the $2k$ symbols in $\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k)$ is colored by a distinct color, and the symbols in $(\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)) - (\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k))$ are colored by the color-

ing f. Therefore, in time $O^*(6.4^{2k/3})$, we can construct a collection $\mathcal{C}$ of $O^*(6.4^{2k/3})$ $(\lceil 8k/3+2\rceil)$-colorings for the set $\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)$. In particular, let f_0' be the $(\lceil 8k/3+2\rceil)$-coloring in $\mathcal{C}$ that is induced from the $(\lceil 2k/3+2\rceil)$-coloring f_0 in $\mathcal{C}_0$, where f_0 properly colors the set $D = (\mathrm{Val}^p(M_{k+1}) \cup \mathrm{Val}^q(M_{k+1})) - (\mathrm{Val}^p(M_k) \cup \mathrm{Val}^q(M_k))$. By definition, the $(\lceil 8k/3+2\rceil)$-coloring f_0' in the collection $\mathcal{C}$ (p,q)-properly colors the $(k+1)$-matching M_{k+1}. This completes the proof of the lemma. □

Now we are ready for our main result in this section.

Theorem C.5 *The* 3-D MATCHING AUGMENTATION *problem can be solved in time* $O^*(2.80^{3k})$.

PROOF. Let (S, M_k) be an instance of the 3-D MATCHING AUGMENTATION problem, where S is a triple set and M_k is a k-matching in S. Consider the following algorithm:

> **3DMA.** For each pair p and q of indices in $\{1,2,3\}$, construct a collection $\mathcal{C}_{p,q}$ of $O^*(6.4^{2k/3})$ $(\lceil 8k/3+2\rceil)$-colorings for the set $\mathrm{Val}^p(S) \cup \mathrm{Val}^q(S)$, as given in Lemma C.4. Then for each coloring f in the collection $\mathcal{C}_{p,q}$, call the algorithm **3DMatch**$(S, k+1, f, \lceil 8k/3+2\rceil; p, q)$ to see if a $(k+1)$-matching (p,q)-properly colored by f can be constructed.

By Lemma C.4, if the triple set S contains $(k+1)$-matchings, then there are indices p and q in $\{1,2,3\}$ and a $(\lceil 8k/3+2\rceil)$-coloring f_0 in $\mathcal{C}_{p,q}$ such that a $(k+1)$-matching M of S is (p,q)-properly colored by f_0. By Theorem C.3, the algorithm **3DMatch**$(S, k+1, f_0, \lceil 8k/3+2\rceil; p, q)$ on this particular coloring f_0 and the indices p and q will return a $(k+1)$-matching in S. Therefore, the algorithm **3DMA** solves the 3-D MATCHING AUGMENTATION problem correctly.

Now we analyze the complexity of the algorithm **3DMA**. By Theorem C.3, the running time of each call on the algorithm **3DMatch**$(S, k+1, f, \lceil 8k/3+2\rceil; p, q)$

is $O^*(\sum_{i=0}^{k+1} \binom{\lceil 8k/3+2 \rceil}{2i}) = O^*(2^{8k/3})$. Since there are totally $O^*(6.4^{2k/3})$ ($\lceil 8k/3+2 \rceil$)-colorings in the collection $\mathcal{C}_{p,q}$, the algorithm on each pair of indices p and q in $\{1,2,3\}$ runs in time

$$O^*(6.4^{2k/3}) \cdot O^*(2^{8k/3}) = O^*(2.80^{3k}).$$

Now the theorem follows since there are only three different pairs of indices in $\{1,2,3\}$.

□

Combining Theorem C.5 with Lemma C.1, we get immediately

Corollary C.6 *The* P-3-D MATCHING *problem can be solved in time* $O^*(2.80^{3k})$.

Corollary C.6 gives a significant step towards the improvements on deterministic algorithms for P-3-D MATCHING. The best previous upper bound for P-3-D MATCHING is $O^*(3.52^{3k})$ by Wang and Feng [119]. Moreover, we remark that if we replace the deterministic construction of the color coding scheme given in Theorem B.18 by a randomized construction as described in [3], we will obtain a randomized algorithm of running time $O^*(2.32^{3k})$ for the P-3-D MATCHING problem. This randomized algorithm was reported in 2006 in a preliminary version of the current paper [86], and stood as the best randomized algorithm for P-3-D MATCHING until very recently when Koutis announced his randomized algorithm of running time $O^*(2^{3k})$ for the P-3-D MATCHING and P-3-SET PACKING problems [76].

D. Chapter Conclusion

In this chapter, we studied and improved a number of algorithmic techniques, including color coding, dynamic programming, and greedy localization, which have been very useful in recent research in the development of improved algorithms for the P-3-D MATCHING and other parameterized problems.

A k-color coding scheme is a collection of k-coloring functions such that every subset of size k can be colored properly, i.e. no two elements in the subset are assigned with the same color, by at least one coloring function in the scheme. Hence if we enumerate all k-coloring functions in the scheme, the unknown solution subset of size k will be colored properly by at least one k-coloring function in the scheme, which in this case, we can find the solution efficiently through dynamic programming. The time in coloring step is decided by the size, i.e. the number of k-coloring functions, of the scheme. In this chapter, we constructed a k-color coding scheme of size $O^*(6.4^k)$ which is a significantly progress comparing with the previous best scheme of size $O^*(16^k)$. With our better scheme, all previous algorithms based on color coding technique can be improved.

The iterative expansion method seems especially suitable for parameterized problems whose corresponding optimization problems are to find the solution of maximum value. In general, the method starts from a trivial small solution, and iteratively "expands" a given solution into a larger one until a solution of the desired size is obtained. For example, in solving the P-3-D MATCHING problem, we started from a matching of 1 triple. Using the matching of 1 triple, we found a matching of 2 triples. Then using the matching of 2 triples, we found a matching of 3 triples, and so on, until we found a matching of k triples. The reason that we want to use iterative expansion method for the P-3-D MATCHING problem is that every matching of i triples will have $2i$ symbols in a matching of $i+1$ triples if there is a matching of $i+1$ triples. Hence, we can take advantage of a matching of i triples to find a matching of $i+1$ triples.

In addition to the improved color coding scheme of size $O^*(6.4^k)$ and iterative expansion method, we also used a property for the MATCHING problem to develop our algorithm in the chapter. As symbol sets from different columns are not intersected, after sorting all triples by the order of symbols from on column, we only need to color

symbols of a matching from remaining two columns properly. This technique can greatly save the time for the coloring and the time for the dynamical programming. This is also the key technique that we can developed a more efficient algorithm for the P-3-D MATCHING problem than the P-3-SET PACKING problem.

Combining above three techniques, we improved the running time of the deterministic algorithm for the P-3-D MATCHING problem to $O^*(2.80^{3k})$. We believe that this group of methods should be of general interest for the development of parameterized algorithms.

CHAPTER VI

BRANCH AND ITERATIVE COMPRESSION*

We have introduced branch and bound techniques before. The basic idea of the branch and bound is very simple. Given an instance of a problem, we branch the instance into two or more simpler sub-instances that the solution of the original instance can be constructed from solutions of all sub-instances. Then we branch each sub-instance again and repeat this process until we reach sub-instances that can be solved easily, i.e. that can be solved in polynomial time.

The iterative compression technique was proposed by Reed et al. [107]. The technique is suitable for parameterized NP-hard problems whose corresponding NP-hard optimization problems are to find the solution of minimum value. The basic idea is that instead of finding a solution of size k directly, which is very hard, we solve a serial of problems that find a solution of size k under the condition that a solution of size $k+1$ is given. One example is the p-feedback vertex set problem on undirected graph G. If we remove $n-k$ vertices from graph G, the remaining graph $G1$ will have an FVS of size at most k. We add one vertex back to $G1$ to get graph $G2$. Then $G2$ will have an FVS of size at most $k+1$. If we can find an FVS of size k in $G2$, we add another vertex back to $G2$, and so on, until we add all vertices back.

In this chapter, we will combine iterative compression technique with branch and bound techniques to solve the pd-feedback vertex set, p-feedback vertex

---

*Reprinted with permission from "A fixed-parameter algorithm for the directed feedback vertex set problem" by Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, Igor Razgon, 2008. *Journal of the ACM*, (accepted), Copyright 2009 by ACM.

*Reprinted with permission from "Improved algorithms for feedback vertex set problems" by Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, Yngve Villanger, 2008. *Journal of Computer and System Sciences*, 74, 1188-1198, Copyright 2009 by Elsevier.

SET and PW-FEEDBACK VERTEX SET problems.

A. The Feedback Vertex Set Problem on Directed Graphs

1. Introduction

Let G be a directed graph. A *feedback vertex set* F (briefly, dFVS) for G is a set of vertices in G such that every directed cycle in G contains at least one vertex in F, or equivalently, that the removal of F from the graph G leaves a directed acyclic graph (i.e., a DAG). The (parameterized directed) FEEDBACK VERTEX SET problem, i.e. PD-FEEDBACK VERTEX SET problem, is defined as follows: Given a directed graph G and a parameter k, either construct a dFVS of at most k vertices for G or report that no such set exists.

The PD-FEEDBACK VERTEX SET problem is a classic NP-complete problem that appeared in the first list of NP-complete problems in Karp's seminal paper [71], and has a variety of applications in areas such as operating systems [112], database systems [55], and circuit testing [84]. In particular, the PD-FEEDBACK VERTEX SET problem has played an essential role in the study of *deadlock recovery* in database systems and in operating systems [112, 55]. In such a system, the status of system resource allocations can be represented as a directed graph G (i.e., the *system resource-allocation graph*), and a directed cycle in G represents a deadlock in the system. Therefore, in order to recover from deadlocks, we need to abort a set of processes in the system, i.e., to remove a set of vertices in the graph G, so that all directed cycles in G are broken. Equivalently, we need to find a dFVS in the graph G. In practice, one may expect and desire that the number of vertices removed from the graph G, which is the number of processes to be aborted in the system, be small. This motivates the study of *parameterized algorithms* for the PD-FEEDBACK VERTEX SET problem that find a

dFVS of k vertices in a directed graph of n vertices and run in time $f(k)n^{O(1)}$ for a fixed function f; thus, the algorithms become practically efficient when the value k is small.

This work has been part of a systematic study of the theory of fixed-parameter tractability [44], which has received considerable attention in recent years. The fixed-parameter tractability of the PD-FEEDBACK VERTEX SET problem was posted as an open problem in the very first papers on the study of fixed-parameter tractability [41, 43]. After numerous significant efforts, however, the problem still remained open. In the past fifteen years, the problem has been constantly and explicitly posted as an open problem in a large number of publications in the literature (see [62] for a recent survey on this study). The problem has become a well-known and outstanding open problem in parameterized computation and complexity.

In this section, we develop new algorithmic techniques that lead to the conclusion that the PD-FEEDBACK VERTEX SET problem is fixed-parameter tractable, and thus resolve the above open problem in parameterized computation and complexity. We first show that the PD-FEEDBACK VERTEX SET problem can be reduced in time $f(k)n^{O(1)}$ for some function f to a special version of the multi-cut problem, which will be called the PD-SKEW SEPARATOR problem. We then develop an algorithm that shows the fixed-parameter tractability of the PD-SKEW SEPARATOR problem. The combination of these two results gives an algorithm with running time $4^k k! n^{O(1)}$ for the PD-FEEDBACK VERTEX SET problem, which proves its fixed-parameter tractability.

The relationship between the PD-FEEDBACK VERTEX SET problem and multi-cut problems has been studied in the research of approximation algorithms for the D-FEEDBACK VERTEX SET problem [48, 83]. However, our problem formulations and the corresponding techniques are significantly different from those studied in the approximation algorithms. In particular, our formulations and techniques seem especially

suitable for developing faster and more effective exact algorithms (of exponential-time) for NP-hard multi-cut problems. First of all, instead of seeking a multi-cut that separates a given set of *terminal vertices*, as formulated in most multi-cut problems, our problem is more general: we wish to construct a multi-cut that separates a *collection of terminal vertex-subsets*. This more general version of the multi-cut problem enables us to effectively reduce the search space size when we are searching for an *optimal* solution of a given problem instance. Secondly, unlike most multi-cut problems whose solutions are multi-cuts that are in general symmetric to the given terminal vertices, the multi-cuts for the PD-SKEW SEPARATOR problem are asymmetric to the terminal vertex-subsets. Thirdly, we develop an (exponential-time) reduction that effectively reduces the problem of multi-cuts for multiple terminal vertex-subsets to the problem of minimum cuts from a single source vertex to a single sink vertex. Note that the latter is solvable in polynomial time via algorithms for the MAXIMUM FLOW problem. Such an exponential-time reduction is obviously very different from the polynomial-time processes used in the development of polynomial-time approximation algorithms. Finally, unlike most parameterized algorithms that are focused on effectively decreasing the sole parameter value k, our algorithm for the PD-SKEW SEPARATOR problem adds another dimension of bounds in terms of the size of a minimum cut between two properly chosen terminal vertex-subsets. This dimension of bounds has become crucial in our development of the algorithm for the PD-SKEW SEPARATOR problem because it effectively bounds the number of branches in which the parameter value k is not decreased.

Before we move to the technical discussion of our algorithms, we remark that the P-FEEDBACK VERTEX SET problem (the parameterized FEEDBACK VERTEX SET problem on undirected graphs) has also been an interesting and active research topic in parameterized computation and complexity. Since the first fixed-parameter tractable

algorithm for the P-FEEDBACK VERTEX SET problem was published almost twenty years ago [12], there has been an impressive list of improved algorithms for the problem. Currently the best algorithm for the P-FEEDBACK VERTEX SET problem runs in time $O(5^k kn^2)$ [21], where we will discuss this new result in next section. The PD-FEEDBACK VERTEX SET problem seems very different from the problem on undirected graphs (i.e., the P-FEEDBACK VERTEX SET problem). This fact has also been reflected in the study of approximation algorithms for the problems. The FEEDBACK VERTEX SET on undirected graphs is polynomial-time approximable with a ratio 2. This holds true even for weighted graphs [7]. On the other hand, it still remains open whether the FEEDBACK VERTEX SET problem on directed graph (i.e. the D-FEEDBACK VERTEX SET problem) has a constant-ratio polynomial-time approximation algorithm. The current best polynomial-time approximation algorithm for the problem on directed graphs has a ratio $O(\log \tau \log \log \tau)$, where τ is the size of a minimum dFVS for the input graph [48].

2. Preliminaries

Let $G = (V, E)$ be a directed graph and let $e = [u, v]$ be a (directed) edge in G. We say that the edge e *goes out* from the vertex u and *comes into* the vertex v. The edge e is called an *outgoing edge* of the vertex u, and an *incoming edge* of the vertex v. These concepts can be extended from single vertices to general vertex sets. Thus, for two vertex sets S_1 and S_2, we can say that an edge goes out from S_1 and comes into S_2 if the edge goes out from a vertex in S_1 and comes into a vertex in S_2. Moreover, we say that an edge *goes out from* S_1 if the edge goes out from a vertex in S_1 and comes into a vertex not in S_1, and that an edge *comes into* S_2 if the edge goes out from a vertex not in S_2 and comes into a vertex in S_2.

A *path* P from a vertex v_1 to a vertex v_h in the graph G is a sequence $\{v_1, \ldots, v_h\}$

of vertices in G such that $[v_i, v_{i+1}]$ is an edge in G for all $1 \leq i \leq h-1$. The path P is *simple* if no vertex is repeated in P. The path P is a *cycle* if $v_1 = v_h$, and the cycle is *simple* if no other vertices are repeated. We say that a path is from a vertex set S_1 to a vertex set S_2 if the path is from a vertex in S_1 to a vertex in S_2. The graph G is a *DAG* (i.e., directed acyclic graph) if it contains no cycles.

For a vertex subset $V' \subseteq V$ in the directed graph $G = (V, E)$, we denote by $G[V']$ the subgraph of G that is induced by the vertex subset V'. Without any ambiguity, we will denote by $G - V'$ the induced subgraph $G[V - V']$, and by $G - w$, where w is a vertex in G, the induced subgraph $G[V - \{w\}]$.

A vertex subset F in the directed graph G is a *feedback vertex set* (*dFVS*) if the graph $G - F$ is a DAG. Since a vertex v with a self-loop (i.e., an edge that both goes out from and comes into v) must be included in every dFVS for the graph G, we will assume, without loss of generality, that the graphs in our discussion have no self-loops.

Definition Let $[S_1, \ldots, S_l]$ and $[T_1, \ldots, T_l]$ be two collections of l vertex subsets in a directed graph $G = (V, E)$. A *skew separator* X for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ is a vertex subset in $V - \bigcup_{i=1}^{l}(S_i \cup T_i)$ such that for any pair of indices i and j satisfying $l \geq i \geq j \geq 1$, there is no path from S_i to T_j in the graph $G - X$.

The subsets S_1, ..., S_l will be called the *source sets* and the subsets T_1, ..., T_l will be called the *sink sets*. A vertex is a *non-terminal vertex* if it is not in $\bigcup_{i=1}^{l}(S_i \cup T_i)$. Note that by definition, all vertices in a skew separator must be non-terminal vertices. Moreover, a skew separator X is asymmetric to the source sets and the sink sets: a path from S_i to T_j with $i < j$ may exist in the graph $G - X$.

When there is only one source set S_1 and one sink set T_1, a skew separator for the

pair $([S_1], [T_1])$ becomes a regular cut for S_1 and T_1, i.e., a vertex set whose removal leaves a graph in which there is no path from the set S_1 to the set T_1. Therefore, a skew separator for the pair $([S_1], [T_1])$ is also called a *cut from* S_1 *to* T_1. A cut from S_1 to T_1 is a *min-cut* (i.e., a minimum cut) if it has the smallest cardinality over all cuts from S_1 to T_1.

The following lemma can be easily derived based on standard maximum flow techniques [109]. Thus, we omit its proof.

Lemma A.1 *There is an $O(kn^2)$ time algorithm that for two given vertex subsets S and T in a directed graph G of n vertices, and a parameter k, either constructs a min-cut from S to T whose size is bounded by k, or reports that the min-cut from S to T has a size larger than k.*

The algorithm for the PD-FEEDBACK VERTEX SET problem is obtained through careful development of algorithms for a series of problems. In the following, we give the formal definitions of these problems.

> PD-SKEW SEPARATOR: given $(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$, where G is a directed graph, $[S_1, \ldots, S_l]$ is a collection of l source sets and $[T_1, \ldots, T_l]$ is a collection of l sink sets in G, and a parameter k, such that
>
> (1) all sets S_1, ..., S_l, T_1, ..., T_l are pairwise disjoint;
>
> (2) for each i, $1 \leq i \leq l-1$, there is no edge coming into the source set S_i; and
>
> (3) for each j, $1 \leq j \leq l$, there is no edge going out from the sink set T_j,
>
> either construct a skew separator of at most k vertices for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$, or report that no such separator exists.

Note that in an instance of the PD-SKEW SEPARATOR problem, condition (2) on source sets and condition (3) on sink sets are not completely symmetric. Although the first $l-1$ source sets are not allowed to have incoming edges, the last source set S_l is allowed to have incoming edges. On the other hand, *all* sink sets are not allowed to have outgoing edges.

We remark that conditions (1)-(3) in the definition of the PD-SKEW SEPARATOR problem (plus the restriction that the skew separator can consist of only non-terminal vertices) may be relaxed, and our techniques for the problem may still be applicable. However, the above formulation of the problem will make our discussion simpler, and will also be sufficient for our solution to the PD-FEEDBACK VERTEX SET problem, which is the focus of the current section. We leave the investigation of the separator problems of more general forms to later research.

Let $G = (V, E)$ be a directed graph, and let (D_1, D_2) be a bi-partition of the vertex set V of G, i.e., $D_1 \cup D_2 = V$ and $D_1 \cap D_2 = \emptyset$. The bi-partition (D_1, D_2) is a *DAG-bipartition* for the graph G if both induced subgraphs $G[D_1]$ and $G[D_2]$ are DAGs. A vertex subset F in the graph G is a D_1-FVS if F is a dFVS for G and $F \subseteq D_1$.

> PD-DAG-BIPARTITION FEEDBACK VERTEX SET: given (G, D_1, D_2, k), where G is a directed graph, (D_1, D_2) is a DAG-bipartition for G, and k is the parameter, either construct a D_1-FVS of size bounded by k for the graph G, or report that no such D_1-FVS exists.

We will be also interested in a special version of the D-FEEDBACK VERTEX SET problem.

> D-FEEDBACK VERTEX SET REDUCTION: given a triple (G, F, k), where G is a directed graph and F is a dFVS of size $k+1$ for G, either construct

a dFVS of size bounded by k for G, or report that no such dFVS exists.

Finally, our central problem in this section is as follows.

PD-FEEDBACK VERTEX SET: given a pair (G, k), where G is a directed graph and k is the parameter, either construct a dFVS of size bounded by k for G, or report that no such dFVS exists.

3. Solving the PD-SKEW SEPARATOR problem

In this subsection, we study the complexity of the PD-SKEW SEPARATOR problem.

Let $(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$ be an instance of the PD-SKEW SEPARATOR problem. Define $T_{all} = \bigcup_{1 \leq i \leq l} T_i$. There are a few cases in which we can directly reduce the instance size:

Rule R1. There is no path from S_l to T_{all}, i.e., the size of a min-cut from S_l to T_{all} is 0: then we only need to find a skew separator of size k that separates S_i from T_j for all indices i and j satisfying $l-1 \geq i \geq j \geq 1$, i.e., we can work instead on the instance $(G, [S_1, \ldots, S_{l-1}], [T_1, \ldots, T_{l-1}], k)$. Note that in this case, by definition, if $l = 1$, then the solution to the instance $(G, [S_1, \ldots, S_{l-1}], [T_1, \ldots, T_{l-1}], k)$ is simply the empty set $\emptyset$;

Rule R2. There is an edge from S_l to T_{all}: then there is no way to even separate S_l from T_{all} – we can simply stop and claim that the given instance is a "No" instance;

Rule R3. There exists a non-terminal vertex w, an edge from S_l to w, and an edge from w to T_{all}: then the vertex w must be included in the skew separator in order to separate S_l and T_{all} – we can simply work on the instance $(G -$

$w, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k-1)$ and recursively find a skew separator of size $k-1$.

Note that in Rules R1 and R3, the reduced instances $(G, [S_1, \ldots, S_{l-1}], [T_1, \ldots, T_{l-1}], k)$ and $(G - w, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k-1)$ are still valid instances of the PD-SKEW SEPARATOR problem.

In the following discussion, assume that for the input instance $(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$, none of the rules above is applicable. In particular, since Rule R1 is not applicable, a min-cut from S_l to T_{all} has size larger than 0. Because Rules R1-R3 are not applicable, there must be a non-terminal vertex u_0 such that (1) there is an edge from S_l to u_0; and (2) there is no edge from u_0 to T_{all}. Such a vertex u_0 will be called an *S_l-extended vertex*. Fix an S_l-extended vertex u_0, let $S'_l = S_l \cup \{u_0\}$.

We start with the following simple but important lemma. The proof of this lemma is straightforward. Thus, we omit the proof.

Lemma A.2 *Let X be a subset of vertices in the graph G that does not contain the S_l-extended vertex u_0. Then X is a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ if and only if X is a skew separator for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$.*

Lemma A.2 also directly implies the following two useful corollaries.

Corollary A.3 *A skew separator for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$ is also a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$.*

Corollary A.4 *The size of a min-cut from S'_l to T_{all} in the graph G is at least as large as the size of a min-cut from S_l to T_{all} in G.*

Now we are ready for our main theorem in this subsection.

Theorem A.5 *If the size of a min-cut from S_l to T_{all} is equal to the size of a min-cut from S'_l to T_{all}, then the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ has a skew separator of size bounded by k if and only if the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$ has a skew separator of size bounded by k.*

PROOF. $\Leftarrow$: Suppose that the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$ has a skew separator X' of size bounded by k. By Corollary A.3, X' is also a skew separator for $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. In consequence, $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ has a skew separator of size bounded by k.

$\Rightarrow$: Suppose that the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ has a skew separator X of size bounded by k. If the skew separator X does not contain the S_l-extended vertex u_0, then by Lemma A.2, X is also a skew separator of size bounded by k for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$, and the theorem is proved. Therefore, we can assume that the set X contains the S_l-extended vertex u_0. We will define another set X' that does not contain u_0. We will show that $|X'| \leq |X|$ and that X' is a skew separator for $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. Then the theorem will immediately follow.

Let Y be a min-cut from S'_l to T_{all}. Then Y does not contain the S_l-extended vertex u_0. Moreover, since there is no edge coming into S_i from outside of S_i for all $i \leq l-1$, the set Y does not contain any vertex in $\bigcup_{i=1}^{l-1} S_i$. In consequence, the set Y consists of only non-terminal vertices. By Corollary A.3, Y is also a cut from S_l to T_{all}. Moreover, by the assumption of the theorem that the size of a min-cut from S_l to T_{all} is equal to the size of a min-cut from S'_l to T_{all}, Y is actually also a min-cut from S_l to T_{all}. Let $R_Y(S_l)$ be the set of vertices v such that either $v \in S_l$ or there is a path from S_l to v in the subgraph $G - Y$. In particular, $u_0 \in R_Y(S_l)$ because Y does not contain u_0 and there is an edge from S_l to u_0.

We introduce a number of sets as follows.

$$\begin{aligned} Z &= X \cap Y; \\ X_{in} &= X \cap R_Y(S_l); \\ X_{out} &= X - (X_{in} \cup Z). \end{aligned}$$

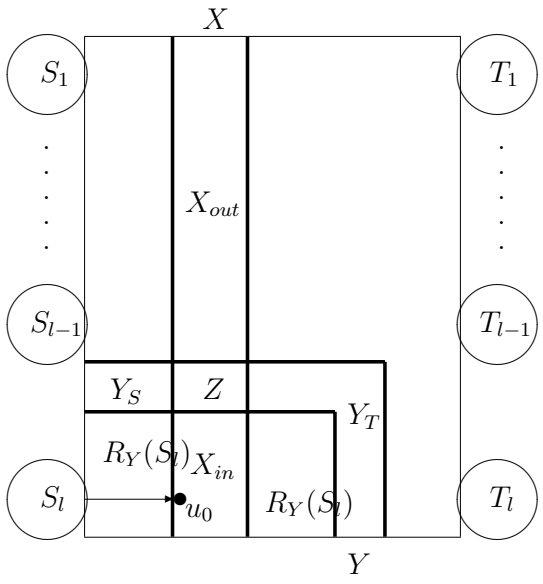
That is, the skew separator X for $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ is decomposed into three disjoint subsets Z, X_{in}, and X_{out} (note that by definitions, $R_Y(S_l)$ and Y do not intersect).

Let Y_T be the set of vertices v in the min-cut Y such that there is a path from v to T_{all} in the subgraph $G - X$. By definition, we have $Y_T \cap Z = \emptyset$. Let

$$Y_S = Y - (Y_T \cup Z).$$

Thus, the min-cut Y from S_l to T_{all} is decomposed into three disjoint subsets Z, Y_T, and Y_S. Figure 11 gives an intuitive illustration of the sets Z, X_{in}, X_{out}, Y_T, Y_S, and $R_Y(S_l)$.

We first show that the set $Y' = Y_S \cup Z \cup X_{in}$ is also a cut from S_l to T_{all}. If by contradiction Y' is not a cut from S_l to T_{all}, then there is a path P_1 from S_l to T_{all} in the subgraph $G - Y'$. The path P_1 must contain vertices in the set Y since Y is a cut from S_l to T_{all}. Let w be the first vertex on the path P_1 that is in Y when we traverse from S_l to T_{all} along the path P_1. Then w must be in Y_T since Y' contains both Y_S and Z. Now the partial path P_1' of P_1 from S_l to w (not including w) must be entirely contained in $R_Y(S_l)$ (note that the path P_1 does not intersect $Y_S \cup Z$). Moreover, the path P_1' contains neither vertices in $X_{in} \cup Z$ (by the definition of the set Y') nor vertices in X_{out} (since the sets X_{out} and $R_Y(S_l)$ are disjoint). In summary, the subpath P_1' from S_l to w contains no vertex in the set X. Moreover,



Y: a min-cut from S'_l to T_{all}

X: a skew separator for $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$

$R_Y(S_l)$: vertices reachable from S_l in $G - Y$

$Z = X \cap Y$

$X_{in} = X \cap R_Y(S_l)$

$X_{out} = X - (X_{in} \cup Z)$

Y_T: vertices in Y from which T_{all} is reachable in $G - X$

$Y_S = Y - (Y_T \cup Z)$

Fig. 11. Sets in the proof of Theorem A.5

by the definition of the set Y_T, and $w \in Y_T$, there is a path P''_1 from w to T_{all} in the subgraph $G - X$. Now the concatenation of the paths P'_1 and P''_1 would result in a path from S_l to T_{all} in the graph $G - X$, contradicting the fact that X is a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. This contradiction shows that the set Y' must be a cut from S_l to T_{all}.

Since Y is a min-cut from S_l to T_{all}, we have $|Y| \leq |Y'|$. By definition, $Y = Y_S \cup Z \cup Y_T$ and $Y' = Y_S \cup Z \cup X_{in}$. Also note that Y_S, Z, and Y_T are pairwise disjoint, and that Y_S, Z, and X_{in} are also pairwise disjoint. Therefore, we must have $|Y_T| \leq |X_{in}|$.

Consider the set $X' = X_{out} \cup Z \cup Y_T$. The set X' has the following properties: (1) X' consists of only non-terminal vertices (because both X and Y consist of only non-terminal vertices); (2) $|X'| \leq |X|$ (because $|Y_T| \leq |X_{in}|$), so the size of X' is bounded by k; and (3) the set X' does not contain the S_l-extended vertex u_0 (this is

because u_0 is in X_{in} and Y does not contain u_0). Therefore, if we can prove that X' is a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$, then by Lemma A.2, X' is also a skew separator of size bounded by k for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$. This will complete the proof of the theorem.

Therefore, what remains is to prove that the set $X' = X_{out} \cup Z \cup Y_T$ is a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. Let $R_Y(T_{all})$ be the set of vertices v such that either $v \in T_{all}$, or there is a path from v to T_{all} in the subgraph $G - Y$.

Suppose by contradiction that X' is not a skew separator for $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. Then there is a path P_2 in the subgraph $G - X'$ from S_i to T_j for some $i \geq j$. The path P_2 has the following properties:

1. The path P_2 must contain a vertex in $R_Y(S_l)$: since X is a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$, the path P_2 from S_i to T_j with $i \geq j$ must contain at least one vertex w_1 in $X = X_{in} \cup Z \cup X_{out}$. Now since the path P_2 is in the subgraph $G - X'$, where $X' = X_{out} \cup Z \cup Y_T$, the vertex w_1 must be in X_{in}, which is a subset of $R_Y(S_l)$;

2. The path P_2 must contain a vertex in Y_S: by Property 1, P_2 contains a vertex w_1 in $R_Y(S_l)$. From the vertex w_1 to T_{all} along the path P_2, there must be a vertex w_2 in $Y = Y_S \cup Z \cup Y_T$ since Y is a cut from S_l to T_{all} while w_1 is reachable from S_l in the subgraph $G - Y$. Now since $X' = X_{out} \cup Z \cup Y_T$, and the path P_2 is in the subgraph $G - X'$, the vertex w_2 on the path P_2 must be in the set Y_S;

3. The path P_2 must end at a vertex in $R_Y(T_{all})$; this is simply because P_2 is ended in T_{all}. Note that by definition, no vertex in Y_S can be in $R_Y(T_{all})$.

By Properties 2-3, the path P_2 contains a vertex not in $R_Y(T_{all})$ and ends at a vertex in $R_Y(T_{all})$. Thus, there must be an internal vertex w in the path such that w is not in

$R_Y(T_{all})$ but all vertices after w along the path P_2 (from S_i to T_j) are in $R_Y(T_{all})$. Note that no vertex w' after the vertex w along the path P_2 can be in the set X: w' in X would imply w' in X_{in} (since P_2 is a path in the subgraph $G-X'$), which would imply that there is another vertex after w' that is in Y thus is not in $R_Y(T_{all})$. Moreover, the vertex w must be in the set Y (otherwise, w would be in $R_Y(T_{all})$). Since P_2 is a path in $G-X'$ and $X' = X_{out} \cup Z \cup Y_T$, the vertex w must be in the set Y_S. However, this derives a contradiction: the subpath of P_2 from w to T_{all} shows that the vertex w should belong to the set Y_T (note that all vertices after w on the path are not in X), and the sets Y_S and Y_T are disjoint. This contradiction proves that the set X' must be a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. Since the size of the set X' is bounded by k and X' does not contain the S_l-extended vertex u_0, by Lemma A.2, the set X' is also a skew separator for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$, and the size of X' is bounded by k.

This completes the proof of the theorem. □

Theorem A.5 enables us to develop a parameterized algorithm for the PD-SKEW SEPARATOR problem. The algorithm is presented in Figure 12.

Theorem A.6 *The algorithm* $\mathbf{SMC}(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l), k]$ *solves the* PD-SKEW SEPARATOR *problem in* $O(4^k kn^3)$ *time, where* n *is the number of vertices in the input graph* G.

PROOF. We first prove the correctness of the algorithm. Let $(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$ be an input to the algorithm, which is an instance of the PD-SKEW SEPARATOR problem, where $G = (V, E)$ is a directed graph, $[S_1, \ldots, S_l]$ and $[T_1, \ldots, T_l]$ are the source sets and the sink sets, respectively, and k is the upper bound of the size of the skew separator we are looking for.

Algorithm SMC$(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$
input: an instance $(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$ of the PD-SKEW SEPARATOR problem.
output: a skew separator of size bounded by k for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$, or report "No" (i.e., no such separator exists).

1. **if** $l = 1$ **then** solve the problem in time $O(kn^2)$;
2. **if** Rule R2 applies or $k < 0$ **then** return "No";
3. **if** Rule R1 applies **then**
   return SMC$(G, [S_1, \ldots, S_{l-1}], [T_1, \ldots, T_{l-1}], k)$;
4. **if** Rule R3 applies on a vertex w **then**
   return $\{w\} \cup$ **SMC**$(G - w, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k-1)$; §
5. pick an S_l-extended vertex u_0; let $S'_l = S_l \cup \{u_0\}$;
6. let m be the size of a min-cut from S_l to $T_{all} = \bigcup_{i=1}^{l} T_i$;
7. **if** $m > k$ **then** return "No";
8. let m' be the size of a min-cut from S'_l to T_{all};
9. **if** $(m = m')$ **then**
9.1. **return SMC**$(G, [S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l], k)$;
9.2. **else** $X = \{u_0\} \cup$ **SMC**$(G - u_0, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k-1)$;
   if $X \neq$ "No" **then** return X;
9.3. **else return SMC**$(G, [S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l], k)$.

§ To simplify our description, we assume that a "No" plus anything gives a "No".

Fig. 12. An algorithm for the PD-SKEW SEPARATOR problem

If $l = 1$, then the problem becomes the construction of a min-cut of size bounded by k from S_1 to T_1, which can be solved in $O(kn^2)$ time by Lemma A.1. Steps 2-4 were justified in the discussions of Rules 2, 1, 3, respectively, at the beginning of this subsection (note that we have also consistently defined that an instance is a "No" instance if the parameter k has a negative value). Therefore, if the algorithm reaches step 5, then none of the Rules 1-3 are applicable. In particular, since Rule 1 is not applicable and the sets S_l and T_{all} are disjoint, there must be an edge $[v, w]$, where $v \in S_l$ and $w \notin S_l$. Since Rule 2 is not applicable, the vertex w is not in the set T_{all}. The vertex w also cannot be in any source set S_i for $i < l$ because there is no edge coming into S_i from outside of S_i. Therefore, the vertex w is a non-terminal vertex.

Finally, since Rule 3 is not applicable, there is no edge from w to T_{all}. Thus, w must be an S_l-extended vertex. This proves that at step 5, the algorithm can always find an S_l-extended vertex u_0.

In the case $m > k$ in step 7, i.e., the size m of a min-cut from S_l to T_{all} is larger than the parameter k, then even separating a single source set S_l from the sink sets $T_{all} = \bigcup_{j=1}^{l} T_j$ requires more than k vertices. Thus, no skew separator of size bounded by k can exist to separate S_i from T_j for all $l \geq i \geq j \geq 1$. Step 7 correctly handles this case by returning "No".

In the case $m = m'$ in step 9, i.e., the size m of a min-cut from S_l to T_{all} is equal to the size m' of a min-cut from S'_l to T_{all}, by Theorem A.5, the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ has a skew separator of size bounded by k if and only if the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$ has a skew separator of size bounded by k. Moreover, by Corollary A.3, a skew separator of size bounded by k for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$ is also a skew separator for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$. Therefore, in this case we can recursively call $\mathbf{SMC}(G, [S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l], k)$, and look instead for a skew separator of size bounded by k for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$, as handled by step 9.1.

In the case $m \neq m'$, then the algorithm branches into two subcases: step 9.2 includes the S_l-extended vertex u_0 in the skew separator and recursively looks for a skew separator of size bounded by $k-1$ in the remaining graph $G - u_0$ for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$; and step 9.3 excludes the S_l-extended vertex u_0 from the skew separator and recursively looks for a skew separator that does not contain u_0 and is of size bounded by k in the graph G for the pair $([S_1, \ldots, S_l], [T_1, \ldots, T_l])$ (which, by Lemma A.2, is a skew separator of size bounded by k for the pair $([S_1, \ldots, S_{l-1}, S'_l], [T_1, \ldots, T_{l-1}, T_l])$). This completes the verification of the correctness of the algorithm. Now we analyze its complexity.

The recursive execution of the algorithm can be described as a search tree $\mathcal{T}$. We first count the number of leaves in the search tree $\mathcal{T}$. Note that only steps 9.2-9.3 of the algorithm correspond to branches in the search tree $\mathcal{T}$. Let $D(k, m)$ be the total number of leaves in the search tree $\mathcal{T}$ for the algorithm $\mathbf{SMC}(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$, where m is the size of a min-cut from S_l to T_{all}. Then steps 9.2-9.3 induce the following recurrence relation:

$$D(k, m) \leq D(k-1, m_1) + D(k, m_2) \tag{6.1}$$

where m_1 is the size of a min-cut from S_l to T_{all} in the graph $G - u_0$ as given in step 9.2, and m_2 is the size of a min-cut from S'_l to T_{all} in the graph G as given in step 9.3. Note that $m - 1 \leq m_1 \leq m$ because removing the vertex u_0 from the graph G cannot increase the size of a min-cut from S_l to T_{all}, and can decrease the size of a min-cut for the two sets by at most 1. Moreover, by Corollary A.4, in step 9.3 we must have $m_2 \geq m + 1$. Summarizing these, we have

$$m - 1 \leq m_1 \leq m \qquad \text{and} \qquad m_2 \geq m + 1. \tag{6.2}$$

We prove, by induction on $t = 2k - m$, that $D(k, m) \leq 2^{2k-m}$. First note that we always have $t = 2k - m \geq 0$ because by the definitions of k and m we always have $k \geq m \geq 0$. In particular, in the initial case when $t = 2k - m = 0$, we must have $k = m = 0$; in this case the algorithm can solve the instance without further branching. Therefore, we have $D(k, m) = 1$ when $t = 2k - m = 0$. For the inductive step, note that by Inequalities (6.2), we have

$$t_1 = 2(k-1) - m_1 \leq 2(k-1) - (m-1) = 2k - m - 1,$$

and

$$t_2 = 2k - m_2 \leq 2k - (m+1) = 2k - m - 1.$$

Therefore, we can apply the inductive hypothesis on Inequality (6.1), which gives

$$\begin{aligned} D(k,m) &\leq D(k-1,m_1) + D(k,m_2) \\ &\leq 2^{2(k-1)-m_1} + 2^{2k-m_2} \\ &\leq 2^{2k-m-1} + 2^{2k-m-1} \\ &= 2^{2k-m}. \end{aligned} \tag{6.3}$$

This completes the inductive proof. Moreover, we also note that certain non-branching steps (i.e., steps 3, 4, and 9.1) may also change the values of k and m, thus changing the value $t = 2k - m$. However, none of these steps increases the value $t = 2k - m$: (i) step 3 keeps the value k unchanged and does not decrease the value m (because in this case the size of a min-cut from S_l to T_{all} is 0 that cannot be larger than the size of a min-cut from S_{l-1} to $\bigcup_{j=1}^{l-1} T_j$); (ii) step 4 decreases the value k by 1 and the value m by at most 1 (because removing a vertex from G can reduce the size of a min-cut from S_l to T_{all} by at most 1), which as a total will decrease the value $t = 2k - m$ by at least 1; (iii) by the condition assumed, step 9.1 keeps both the values k and m unchanged, thus unchanging the value $t = 2k - m$. As a result, the value $t = 2k - m$ after a branching step to the next branching step can never be increased.

Summarizing the above discussion, we conclude that the total number of leaves, $D(k, m)$, in the search tree $\mathcal{T}$ for the algorithm $\mathbf{SMC}(G, [S_1, \ldots, S_l], [T_1, \ldots, T_l], k)$, where m is the size of a min-cut from S_l to T_{all}, satisfies the following inequality

$$D(k,m) \leq 2^{2k-m} \leq 4^k.$$

The running time of each execution of the algorithm **SMC**, not counting the time for the recursive calls in the execution, is bounded by $O(kn^2)$, where n is the number of vertices in the input graph. In particular, by Lemma A.1, step 1 that looks

for a min-cut of size bounded by k from S_1 to T_1, steps 6-7 that determine if the size m of a min-cut from S_l to T_{all} is bounded by k, and steps 8-9 that determine if the size of a min-cut from S'_l to T_{all} is equal to m ($m \leq k$ at this point), all have their running time bounded by $O(kn^2)$.

Observe that for each recursive call in an execution of the algorithm **SMC**, either the number of source-sink pairs in the instance is decreased by 1 (step 3), or the number of non-terminal vertices in the instance is decreased by 1 (steps 4, 9.1, 9.2, and 9.3). When the number of source-sink pairs is equal to 1, the problem is solved in time $O(kn^2)$ by step 1, and when the number of non-terminal vertices is equal to 0, either step 2 or step 3 can be applied directly. In conclusion, along each root-leaf path in the search tree $\mathcal{T}$, there are at most $O(n)$ recursive calls to the algorithm **SMC**. Therefore, the running time along each root-leaf path in the search tree $\mathcal{T}$ is bounded by $O(kn^3)$.

Summarizing the above discussions, we conclude that the running time of the algorithm **SMC** is bounded by $O(4^k kn^3)$. This completes the proof of the theorem. □

4. Solving the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem

In this subsection, we describe how to use the results in the previous subsection to solve the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem.

Recall that an instance of PD-DAG-BIPARTITION FEEDBACK VERTEX SET is given as a tuple (G, D_1, D_2, k), where G is a directed graph, (D_1, D_2) is a DAG-bipartition of G, and k is the parameter, with the objective of finding a dFVS X for the graph G such that $X \subseteq D_1$ (recall that such a dFVS is called a D_1-FVS) and that the size of X is bounded by k.

Let $\pi = \{v_1, v_2, \ldots, v_h\}$ be a topologically sorted order of the vertices in the

induced DAG $G[D_2]$. We construct an instance of the PD-SKEW SEPARATOR problem as follows:

1. Let G' be the graph obtained from G by removing all edges in $G[D_2]$.

2. In the graph G', replace each vertex v_i in D_2 by a pair (t_i, s_i) of vertices such that all incoming edges into v_i are now coming into the vertex t_i, and that all outgoing edges from v_i are now going out from the vertex s_i. Let the resulting graph be G_π.

Note that in the resulting graph G_π, the vertices s_i, $1 \leq i \leq h$, have no incoming edges, and the vertices t_j, $1 \leq j \leq h$, have no outgoing edges. Moreover, since we have removed all edges between the vertices in $G[D_2]$, every edge going out from a vertex s_i must come into a vertex in the set D_1, and every edge coming into a vertex t_j must go out from a vertex in the set D_1. In particular, $(G_\pi, [\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}], k)$ is a valid instance for the PD-SKEW SEPARATOR problem, which will be called an instance of the PD-SKEW SEPARATOR *induced by the instance* (G, D_1, D_2, k) *of* PD-DAG-BIPARTITION FEEDBACK VERTEX SET *and the topologically sorted order* π *of the vertices in* $G[D_2]$.

Thus, each vertex v_i in the set D_2 in the graph G is now "split" into the two vertices s_i and t_i in the graph G_π. Moreover, there is a one-to-one mapping between the vertices in the set D_1 in the graph G and the non-terminal vertices in the graph G_π. Thus, in case of no ambiguity, we will use the same vertex name to refer to both a non-terminal vertex in the graph G_π and a vertex in the set D_1 in the graph G. In particular, a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π corresponds to a subset of D_1 in the graph G. We have the following important theorem.

Theorem A.7 *Let (G, D_1, D_2, k) be an instance of the* PD-DAG-BIPARTITION FEEDBACK VERTEX SET *problem, and let X be a D_1-FVS for the graph G. Then there is a topologically sorted order $\pi = \{v_1, \ldots, v_h\}$ of the vertices in $G[D_2]$ such that in the instance $(G_\pi, [\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}], k)$ induced by (G, D_1, D_2, k) and π: (1) X is a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π; and (2) every skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in G_π is a D_1-FVS for the graph G.*

PROOF. As assumed in the theorem, let (G, D_1, D_2, k) be an instance of the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem, and let X be a D_1-FVS for the graph G. Consider the subgraph $G - X$. Since X is a dFVS for G, the graph $G - X$ is a DAG. Therefore, the vertices in $G - X$ can be topologically sorted into an ordered list π' such that there is no edge in $G - X$ that goes out from a later vertex in π' and comes into an earlier vertex in π'. Let $\pi = \{v_1, \ldots, v_h\}$ be the order of the vertices in D_2 that is induced from the order π' (i.e., π is obtained from π' by removing the vertices not in D_2. Note that all vertices in X are in D_1). The order π is obviously a topologically sorted order for the DAG $G[D_2]$. We show that this order π of the vertices in D_2 and the corresponding instance $(G_\pi, [\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}], k)$ induced by (G, D_1, D_2, k) and π satisfy the conclusions of the theorem.

We first show that the set X is a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π. If this were not the case, then there would be a path P in the graph $G_\pi - X$ that starts from a vertex s_i and ends at a vertex t_j with $i \geq j$. Since no vertex in $\{s_1, \ldots, s_h\}$ has incoming edges and no vertex in $\{t_1, \ldots, t_h\}$ has outgoing edges, all internal vertices on the path P are non-terminal vertices in G_π. In consequence, all internal vertices on P are vertices in the set D_1 in the graph G. Therefore, the path P in $G_\pi - X$ corresponds to a path P' in the graph $G - X$ that

starts from the vertex v_i and ends at the vertex v_j, where $i \geq j$, with all internal vertices of P' in the set D_1. But it is impossible: (1) if $i = j$ then the path P' would be a cycle in the graph $G - X$, contradicting the assumption that X is a dFVS for the graph G; and (2) if $i > j$, then P' would become a path from v_i to v_j with $i > j$ in the graph $G - X$, contradicting the assumption that $\pi = \{v_1, \ldots, v_h\}$ is an order of the vertices in D_2 that is induced from the topologically sorted order π' of the vertices in the DAG $G - X$. In conclusion, the path P does not exist, and the set X is a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π.

Now we prove that every skew separator X' for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π is a D_1-FVS for the graph G. First of all, by definition, a skew separator consists of only non-terminals, thus, all vertices in X' are in the set D_1. Suppose for a contradiction that X' is not a D_1-FVS for the graph G. Then there is a cycle C in the graph $G - X'$. Without loss of generality, we can assume that C is a simple cycle. Since both the induced subgraphs $G[D_1]$ and $G[D_2]$ are DAGs, the cycle C must contain both vertices in D_1 and vertices in D_2. We consider two different cases.

Case 1. The cycle C contains a single vertex v_i in the set D_2. Then all other vertices in the cycle C are in the set D_1. But then the cycle C would correspond to a path P_1 in the graph $G_\pi - X'$ that starts with the vertex s_i and ends at the vertex t_i (with all internal vertices being non-terminal vertices). But this contradicts the assumption that X' is a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ that should have cut all paths from s_i to t_i.

Case 2. The cycle C contains more than one vertex in the set D_2. Let $\{v_{i_1}, v_{i_2}, \ldots, v_{i_d}, v_{i_1}\}$ be the order of the vertices in D_2 that we encounter when traversing along the cycle C (starting from an arbitrary vertex v_{i_1} in D_2), where $d > 1$. Then there must be an index j such that $i_j > i_{j+1}$ (where we take $i_{j+1} = i_1$ if $j = d$). Now

consider the subpath P_2 of C that starts from the vertex v_{i_j} and ends at the vertex $v_{i_{j+1}}$. The path P_2 cannot be a single edge from v_{i_j} to $v_{i_{j+1}}$ since $\pi = \{v_1, v_2, \ldots, v_h\}$ is a topologically sorted order for the vertices in the DAG $G[D_2]$ and $i_j > i_{j+1}$. Thus, the path P_2 contains at least one internal vertex. Since all internal vertices on the path P_2 are not in D_2 thus correspond to non-terminal vertices in the graph $G_\pi - X'$, the path P_2 would correspond to a path P_2' in the graph $G_\pi - X'$ that starts from the vertex s_{i_j} and ends at the vertex $t_{i_{j+1}}$, with $i_j > i_{j+1}$. Again this contradicts the assumption that X' is a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$, which should have cut all paths from s_{i_j} to $t_{i_{j+1}}$ when $i_j > i_{j+1}$.

This proves that the skew separator X' for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π must be a D_1-FVS for the graph G. This completes the proof of the theorem. □

Theorem A.7 enables us to reduce the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem to the PD-SKEW SEPARATOR problem. An algorithm for the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem is given in Figure 13.

Algorithm DBF(G, D_1, D_2, k)
input: an instance (G, D_1, D_2, k) of the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem.
output: a D_1-FVS of size bounded by k for G, or report "No" (i.e., no such D_1-FVS exists).

1. **for** each topologically sorted order $\pi = \{v_1, \ldots, v_h\}$ of the vertices in $G[D_2]$ **do**
1.1. construct the instance $(G_\pi, [\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}], k)$ of the PD-SKEW SEPARATOR problem induced by (G, D_1, D_2, k) and π;
1.2. let $X = \mathbf{SMC}(G_\pi, [\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}], k)$;
1.3. **if** X is a D_1-FVS of size bounded by k for G **then**
return X;
2. **return** "No".

Fig. 13. An algorithm for the PD-DAG-BIPARTITION FEEDBACK VERTEX SET problem

Theorem A.8 *The algorithm* $\mathbf{DBF}(G, D_1, D_2, k)$ *solves the* PD-DAG-BIPARTITION FEEDBACK VERTEX SET *problem in time* $O(4^k kn^3 h!)$*, where* h *is the number of vertices in the set* D_2*, and* n *is the number of vertices in the input graph* G*.*

PROOF. The running time of the algorithm is obvious: the **for**-loop in step 1 is executed at most $h!$ times, and the time for each execution is dominated by the subroutine call to the algorithm **SMC** in step 1.2. By Theorem A.6, the running time of each execution of step 1.2 is bounded by $O(4^k kn^3)$.

For the correctness of the algorithm, first note that the algorithm always returns "No" unless it actually constructs a D_1-FVS of size bounded by k for G in step 1.3. In particular, if the input instance (G, D_1, D_2, k) contains no D_1-FVS of size bounded by k for the graph G, then the algorithm always correctly reports "No".

On the other hand, suppose that there is a D_1-FVS X_0 of size bounded by k for the graph G. Then by Theorem A.7, there is a topologically sorted order $\pi = \{v_1, \ldots, v_h\}$ of the vertices in the DAG $G[D_2]$ such that in the instance $(G_\pi, [\{s_1\}, \ldots, \bigotimes\{s_h\}], [\{t_1\}, \ldots, \{t_h\}], k)$ of the PD-SKEW SEPARATOR problem induced by π and (G, D_1, D_2, k), the set X_0 is a skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π, and every skew separator for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in G_π is a D_1-FVS for the graph G. In particular, the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ has at least one skew separator of size bounded by k (e.g., X_0) in the graph G_π. Therefore, step 1.2 of the algorithm **DBF** must return a skew separator X of size bounded by k for the pair $([\{s_1\}, \ldots, \{s_h\}], [\{t_1\}, \ldots, \{t_h\}])$ in the graph G_π (the set X may be different from the set X_0), and this set X is a D_1-FVS for the graph G. In conclusion, if there is a D_1-FVS of size bounded by k for the graph G, then the algorithm $\mathbf{DBF}(G, D_1, D_2, k)$ will correctly return a D_1-FVS of size bounded by k in step 1.3. □

5. Solving the PD-FEEDBACK VERTEX SET problem

We present our algorithm for the PD-FEEDBACK VERTEX SET problem. We start with a more restricted version of the problem, the D-FEEDBACK VERTEX SET REDUCTION problem, defined as follows.

> D-FEEDBACK VERTEX SET REDUCTION: given a triple (G, F, k), where G is a directed graph and F is a dFVS of size $k+1$ for G, either construct a dFVS of size bounded by k for G, or report that no such dFVS exists.

Lemma A.9 *The* D-FEEDBACK VERTEX SET REDUCTION *problem on a triple* (G, F, k) *is solvable in time* $O(n^3 4^k k^3 k!)$*, where* n *is the number of vertices in the input graph* G*.*

PROOF. Let $G = (V, E)$ be the input directed graph with $n = |V|$ vertices, and let F be the input dFVS of size $k+1$ for the graph G. Every dFVS F' of size bounded by k for G can be split into two disjoint subsets F_1 and F_2, where F_2 consists of j vertices in F for some integer j, $0 \leq j \leq k$, and F_1 consists of at most $k - j$ vertices in $V - F$. Note that since we assume that no vertex in $F - F_2$ is in the dFVS F', the induced subgraph $G[F - F_2]$ must be a DAG. Therefore, for each j, $0 \leq j \leq k$, we enumerate all subsets of j vertices in F. For each such subset F_2 of F such that $G[F - F_2]$ is a DAG, we seek a subset F_1 of at most $k - j$ vertices in $V - F$ such that $F_1 \cup F_2$ makes a dFVS for the graph G.

Fix a subset F_2 of F, such that $|F_2| = j$ and that the induced subgraph $G[F - F_2]$ is a DAG. Note that the graph G has a dFVS $F_1 \cup F_2$ of size bounded by k, where $F_1 \subseteq V - F$, if and only if the subset F_1 of $V - F$ is a dFVS for the graph $G - F_2$ and the size of F_1 is bounded by $k - j$. Therefore, to solve the original problem, we can instead consider how to construct a dFVS F_1 for the graph $G - F_2$ such that

$|F_1| \leq k - j$ and $F_1 \subseteq V - F$.

Since F is a dFVS for G, we have that the induced subgraph $G[V - F] = G - F$ is a DAG. Moreover, by our assumption, the induced subgraph $G[F - F_2]$ is also a DAG. Note that $(V - F) \cup (F - F_2) = V - F_2$, which is the vertex set for the graph $G' = G - F_2$. Therefore, $(V - F, F - F_2)$ is a DAG-bipartition of the graph G'. Thus, a dFVS F_1 for the graph G' such that $|F_1| \leq k - j$ and $F_1 \subseteq V - F$, is actually a $(V - F)$-FVS of size bounded by $k - j$ for the graph G' with the DAG-bipartition $(V - F, F - F_2)$. Therefore, the set F_1 can be constructed by the algorithm $\mathbf{DBF}(G', V - F, F - F_2, k - j)$.

Since $|F| = k+1$ and $|F_2| = j$, we have $|F - F_2| = k+1-j$. Therefore, the DAG $G[F - F_2]$ contains exactly $k+1-j$ vertices. By Theorem A.8, the running time of the algorithm $\mathbf{DBF}(G', V - F, F - F_2, k - j)$ is bounded by $O(4^{k-j}(k - j)n^3(k+1-j)!)$. Now for all integers j, $0 \leq j \leq k$, we enumerate all subsets F_2 of j vertices in F and apply the algorithm $\mathbf{DBF}(G', V - F, F - F_2, k - j)$ for those F_2 such that $G[F - F_2]$ is a DAG. As we discussed above, the graph G has a dFVS of size bounded by k if and only if for some F_2 of j vertices in F, where $0 \leq j \leq k$, the algorithm $\mathbf{DBF}(G', V - F, F - F_2, k - j)$ produces a dFVS F_1 of size bounded by $k - j$ for the graph G'. The running time of this process is bounded by the order of

$$\sum_{j=0}^{k} \binom{k+1}{j} \left(4^{k-j}(k - j)n^3(k + 1 - j)!\right) = O(n^3 4^k k^3 k!).$$

This completes the proof of the lemma. □

The rest of our process for solving the original PD-FEEDBACK VERTEX SET problem is to apply the *iterative compression* method. The method was proposed by Reed, Smith, and Vetta [107] and has been used for solving the P-FEEDBACK VERTEX SET problem [36, 59, 60]. Here we extend the method and apply it to solve the PD-

FEEDBACK VERTEX SET problem.

Theorem A.10 *The* PD-FEEDBACK VERTEX SET *problem can be solved in time* $O(n^4 4^k k^3 k!)$.

PROOF. Let (G, k) be an instance of the PD-FEEDBACK VERTEX SET problem, where $G = (V, E)$ is a directed graph with $n = |V|$ vertices, and k is the parameter. Pick any subset V_0 of $k+1$ vertices in G, and let F_0 be any subset of k vertices in V_0. Note that the set F_0 is a dFVS of k vertices for the induced subgraph $G_0 = G[V_0]$ since the graph $G_0 - F_0$ consists of a single vertex (note that by our assumption, the graph G contains no self-loops).

Let $V - V_0 = \{v_1, v_2, \ldots, v_{n-k-1}\}$. Let $V_i = V_0 \cup \{v_1, \ldots, v_i\}$, and let $G_i = G[V_i]$ be the subgraph induced by V_i, for $i = 0, 1, \ldots, n-k-1$. Inductively, suppose that for an integer i, $0 \leq i < n-k-1$, we have constructed a dFVS F_i of size bounded by k for the induced subgraph G_i (this has been the case for $i = 0$). Without loss of generality, we can assume that the set F_i consists of exactly k vertices – otherwise we simply pick $k - |F_i|$ vertices (arbitrarily) from $G_i - F_i$ and add them to the set F_i. Now consider the set $F'_{i+1} = F_i + v_{i+1}$. Since $G_{i+1} - F'_{i+1} = G_i - F_i$ and F_i is a dFVS for G_i, the set F'_{i+1} is a dFVS of size $k+1$ for the induced subgraph G_{i+1}. In particular, the triple (G_{i+1}, F'_{i+1}, k) is a valid instance for the D-FEEDBACK VERTEX SET REDUCTION problem.

Apply Theorem A.9 to the instance (G_{i+1}, F'_{i+1}, k), which either returns a dFVS F_{i+1} of size bounded by k for the graph G_{i+1}, or claims that no such dFVS exists. It is easy to see that if the induced subgraph $G_{i+1} = G[V_{i+1}]$ does not have a dFVS of size bounded by k, then the original graph G cannot have a dFVS of size bounded by k. Therefore, in this case, we can simply stop and conclude that there is no dFVS of

size bounded by k for the original input graph G. On the other hand, suppose that a dFVS F_{i+1} of size bounded by k is constructed for the graph G_{i+1} in the above process, then the induction successfully proceeds from i to $i+1$ with a new pair (G_{i+1}, F_{i+1}).

In conclusion, the above process either stops at some point and correctly reports that the input graph G has no dFVS of size bounded by k, or eventually ends with a dFVS F_{n-k-1} of size bounded by k for the graph $G_{n-k-1} = G[V_{n-k-1}] = G$.

This process is involved in solving at most $n-k-1$ instances (G_i, F_i, k) of the D-FEEDBACK VERTEX SET REDUCTION problem, for $0 \leq i \leq n-k-2$. By Theorem A.9, the running time of the process is bounded by $O(n^3 4^k k^3 k!(n-k-1)) = O(n^4 4^k k^3 k!)$, and the process correctly solves the PD-FEEDBACK VERTEX SET problem. □

Remark. The running time of the algorithm in Theorem A.10 can be further improved by taking advantage of existing approximation algorithms for the PD-FEEDBACK VERTEX SET problem. Even, Naor, Schieber, and Sudan [48] have developed a polynomial time approximation algorithm for the D-FEEDBACK VERTEX SET problem that for a given directed graph G, produces an dFVS F of size bounded by $c \cdot \tau \log \tau \log \log \tau$ in time $O(n^2 M(n) \log^2 n)$, where c is a constant, τ is the size of a minimum dFVS for the graph G, and $M(n) = O(n^{2.376})$ is the complexity of the multiplication of two $n \times n$ matrices. Therefore, for a given instance (G, k) of the PD-FEEDBACK VERTEX SET problem, we can first apply the approximation algorithm in [48] to construct a dFVS F for the graph G. If $|F| > c \cdot k \log k \log \log k$, then we know that the graph G has no dFVS of size bounded by k. On the other hand, suppose that $|F| \leq c \cdot k \log k \log \log k$. Then we pick a subset F_0 of arbitrary k vertices in F, and let $G_0 = G - (F - F_0)$. The set F_0 is an dFVS of size k for the graph G_0. Now we can proceed exactly the same way as we did in Theo-

rem A.10: let $F - F_0 = \{v_1, v_2, \ldots, v_h\}$, where $h \leq c \cdot k \log k \log\log k - k$, and let $V_i = V_0 \cup \{v_1, \ldots, v_i\}$, and $G_i = G[V_i]$, for $i = 0, 1, \ldots, h$. By repeatedly applying the algorithm in Lemma A.9, we can either stop with a certain index i where the induced subgraph G_{i+1} has no dFVS of size bounded by k (thus the original input graph G has no dFVS of size bounded by k), or eventually construct a dFVS F_h of size bounded by k for the graph $G_h = G[V_h] = G$. This process calls for the execution of the algorithm in Lemma A.9 at most $h = O(k \log k \log\log k)$ times, and each execution takes time $O(n^3 4^k k^3 k!)$. In conclusion, the PD-FEEDBACK VERTEX SET problem can be solved in time $O(n^3 4^k k^4 k! \log k \log\log k + n^{4.376} \log^2 n)$, where the second term in the complexity is due to the approximation algorithm given in [48].

6. Remarks and future research

We presented a parameterized algorithm of running time $O(n^4 4^k k^3 k!)$ for the PD-FEEDBACK VERTEX SET problem, which shows that the problem is fixed-parameter tractable, and resolves an outstanding open problem in parameterized computation and complexity. Before we close the section we give a few remarks on our results and on directions for future research.

There is an edge version of the PD-FEEDBACK SET problem, which is called the PD-FEEDBACK ARC SET problem on directed graph: given a directed graph G and a parameter k, either construct a set of at most k edges in G whose removal leaves a DAG, or report that no such edge set exists. The PD-FEEDBACK ARC SET problem is also a well-known NP-complete problem [56]. As shown by Even, Naor, Schieber, and Sudan [48], the PD-FEEDBACK ARC SET problem and the PD-FEEDBACK VERTEX SET problem can be reduced in linear time from one to the other with the same parameter. Therefore, our results also imply an $O(n^4 4^k k^3 k!)$ time algorithm for the PD-FEEDBACK ARC SET problem.

The techniques developed in this section for solving the PD-SKEW SEPARATOR problem seem to be powerful and generally useful in the study of a variety of separator problems. For example, it has been used recently in developing improved algorithms for a multi-cut problem on undirected graphs in which a separator is sought to (uniformly) separate a set of given terminals [26]. It will be interesting to identify the conditions for the multi-cut problems under which these techniques (and their variations and generalizations) are applicable. In particular, it will be interesting to see if the techniques are applicable to derive the fixed-parameter tractability of the P-FEEDBACK VERTEX SET problem on *weighted and directed graphs*. Note that the fixed-parameter tractability of the problem on weighted and *undirected* graphs has been derived recently [21].

It will be interesting to develop new techniques that lead to faster parameterized algorithms for the PD-FEEDBACK VERTEX SET problem and other related problems. For example, is it possible that the PD-FEEDBACK VERTEX SET problem can be solved in time $O(c^k n^{O(1)})$ for a constant c? Another direction is to look at the *kernelization* of the PD-FEEDBACK VERTEX SET problem, by which we refer to a polynomial-time algorithm that on an instance (G, k) of the PD-FEEDBACK VERTEX SET problem, produces a (smaller) instance (G', k') of the problem, such that the size of the graph G' (the kernel) is bounded by a function $g(k)$ of k (but independent of the size of the original graph G), that $k' \leq k$, and that the graph G has a dFVS of size bounded by k if and only if the graph G' has a dFVS of size bounded by k'. Since now it is known that the PD-FEEDBACK VERTEX SET problem is fixed-parameter tractable, by a general theorem in parameterized complexity theory [44], such a kernelization algorithm exists for the PD-FEEDBACK VERTEX SET problem. However, how small can the size of the kernel G' be? In particular, can the kernel G' have its size bounded by a polynomial of the parameter k? We note that recently there has been progress

in the study of kernelization for the P-FEEDBACK VERTEX SET problem. Bodlaender [13] was able to give a kernel of size $O(k^3)$ for the P-FEEDBACK VERTEX SET problem, and Bodlaender and Penninkx [14] have shown that the P-FEEDBACK VERTEX SET problem has a kernel of size $O(k)$.

This is the end of the introduction for the PD-FEEDBACK VERTEX SET problem. In next section, we will introduce our new algorithm for the P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET problems.

B. The Feedback Vertex Set Problem on Undirected Graphs

1. Introduction

A *feedback vertex set* (FVS) F in G is a set of vertices in an undirected graph G whose removal results in an acyclic graph (or equivalently, every cycle in G contains at least one vertex in F). The problem of finding a minimum feedback vertex set in a graph is one of the classic NP-complete problems [71] and has many applications. The history of the problem can be traced back to the early '60s. For several decades, many different algorithmic approaches were tried on this problem, including approximation algorithms, linear programming, local search, polyhedral combinatorics, and probabilistic algorithms (see the survey of Festa et al. [51]). There are also exact algorithms that find a minimum FVS in a graph of n vertices in time $O(1.8899^n)$ [106] and in time $O(1.7548^n)$ [53]. In this section, we only discuss parameterized FEEDBACK VERTEX SET problem on *undirected* graphs, i.e. the P-FEEDBACK VERTEX SET. Hence unless we specify the graph is directed, all graphs we mention are undirected graphs.

The P-FEEDBACK VERTEX SET problem is the simplification of the PD-FEEDBACK VERTEX SET problem. As the importance of the dead-lock problem in application,

before people made progress in the PD-FEEDBACK VERTEX SET problem on directed graphs, the P-FEEDBACK VERTEX SET problem on *undirected* graphs has received considerable attention, The first group of parameterized algorithms of running time $f(k)n^{O(1)}$ for the P-FEEDBACK VERTEX SET problem was given by Bodlaender [11] and by Downey and Fellows [42]. Since then a chain of dramatic improvements was obtained by different researchers (see Figure 1 for references.)

Bodlaender, Fellows [11, 42]	$O(17(k^4)!n^{O(1)})$
Downey and Fellows [44]	$O((2k+1)^k n^2)$
Raman et al.[105]	$O(\max\{12^k, (4\log k)^k\}n^{2.376})$
Kanj et al.[67]	$O((2\log k + 2\log\log k + 18)^k n^2)$
Raman et al.[104]	$O((12\log k/\log\log k + 6)^k n^{2.376})$
Guo et al.[59]	$O((37.7)^k n^2)$
Dehne et al.[36]	$O((10.6)^k n^3)$

Fig. 14. The history of parameterized algorithms for the P-FEEDBACK VERTEX SET problem

Randomized parameterized algorithms have also been studied in the literature for the P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET problems. The best known randomized parameterized algorithms for the P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET problems are due to Becker et al. [9], who developed a randomized algorithm of running time $O(4^k kn^2)$ for the P-FEEDBACK VERTEX SET problem, and a randomized algorithm of running time $O(6^k kn^2)$ for the PW-FEEDBACK VERTEX SET problem. To our knowledge, no deterministic algorithm of running time $f(k)n^{O(1)}$ for any function f was known prior to our results for the PW-FEEDBACK VERTEX SET problem.

2. Our results

The main result of this subsection is an algorithm for the PW-FEEDBACK VERTEX SET problem, i.e. for a given integer k and a weighted graph G, either finds a minimum weight FVS in G of at most k vertices, or correctly reports that G contains no FVS of at most k vertices. The running time of our algorithm is $O(5^k kn^2)$. This improves and generalizes a long chain of results in parameterized algorithms. Let us remark that the running time of our (deterministic) algorithm comes close to that of the best randomized algorithm for the P-FEEDBACK VERTEX SET problem and is better than the running time of the previous best randomized algorithm for the PW-FEEDBACK VERTEX SET problem.

The general approach of our algorithm is based on the *iterative compression* method [107], which has been successfully used recently for improved algorithms for the P-FEEDBACK VERTEX SET and other problems [36, 59, 107]. This method makes it possible to reduce the original P-FEEDBACK VERTEX SET problem on general graphs to the P-FEEDBACK VERTEX SET problem on graphs with a special decomposition structure. The main contribution of the current section is the development of a general algorithmic technique that identifies a dual parameter in problem instances that limits the number of times where the original parameter k cannot be effectively reduced during a branch and search process. In particular, for the P-FEEDBACK VERTEX SET problem on graphs of the above special decomposition structure, a measure is introduced that nicely combines the original parameter and the dual parameter and bounds effectively the running time of a branch and search algorithm for the P-FEEDBACK VERTEX SET problem. This technique leads to a simpler but significantly more efficient parameterized algorithm for the P-FEEDBACK VERTEX SET problem. Moreover, the introduction of the measure greatly simplifies the processing of degree-2

vertices in a weighted graphs, and enables us to solve the PW-FEEDBACK VERTEX SET problem in the same complexity as that for the P-FEEDBACK VERTEX SET problem. Note that this is significant because no previous algorithms for the P-FEEDBACK VERTEX SET problem on unweighted graphs can be extended to weighted graphs mainly because of the lack of an effective method for handling degree-2 vertices. Finally, the technique of dual parameters seems to be of general usefulness for the development of parameterized algorithms, and has been used more recently in solving other parameterized problems [26, 27].

The remaining part of this section is organized as follows. In subection 3, we provide in full details a simpler algorithm and its analysis for unweighted graphs. This is done for clearer demonstration of our approach. We also indicate why this simpler algorithm does not work for weighted graphs. In subection 4, we obtain the main result of this section, the algorithm for the PW-FEEDBACK VERTEX SET problem. This generalization of the results from subection 3 is not straightforward and requires a number of new structures and techniques.

3. On feedback vertex sets on unweighted graphs

In this subsection, we consider the P-FEEDBACK VERTEX SET problem. We start with some terminologies. A *forest* is a graph that contains no cycles. A *tree* is a forest that is connected (therefore, a forest can be equivalently defined as a collection of disjoint trees). Let W be a subset of vertices in a graph $G = (V, E)$. We will denote by $G[W]$ the subgraph of G that is induced by the vertex set W. For simplicity we will use the notation $G - w$ and $G - W$ for respectively $G[V - \{w\}]$ and $G[V - W]$ where $w \in V$ and $W \subseteq V$. A pair (V_1, V_2) of vertex subsets in a graph $G = (V, E)$ is a *forest bipartition* of G if $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$, and both induced subgraphs

$G[V_1]$ and $G[V_2]$ are forests. For a vertex $u \in V$ the degree of u will be the number of edges incident to u.

Let G be a graph and let F be a subset of vertices in G. The set F is a *feedback vertex set* (shortly, FVS) of G if $G - F$ is a forest. The *size* of an FVS F is the number of vertices in F.

Our main problem is formally defined as follows.

> p-feedback vertex set: given a graph G and an integer k, either find an FVS of size at most k in G, or report that no such FVS exists.

Before we present our algorithm for the p-feedback vertex set problem, we first consider a special version of the problem, defined as follows:

> p-f-bipartition feedback vertex set: given a graph G, a forest bipartition (V_1, V_2) of G, and an integer k, either find an FVS of size at most k for the graph G *in the subset* V_1, or report that no such an FVS exists.

Note that the main difference between the p-f-bipartition feedback vertex set problem and the original p-feedback vertex set problem is that we require that the FVS in the p-f-bipartition feedback vertex set is contained in the given subset V_1.

Observe that certain structures in the input graph G can be easily processed and then removed from G. For example, if a vertex v has a self-loop (i.e., an edge whose both ends are incident to v), then the vertex v is necessarily contained in every FVS in G. Thus, we can directly include v in the objective FVS. If two vertices v and w are connected by multiple edges (i.e., there are more than one edge whose one end is v and the other end is w), then one of v and w must be contained in the objective FVS. Thus, we can branch into two recursive calls, one includes v, and the other includes

w, in the objective FVS. All these operations are more efficient than the algorithm of running time $O(5^k kn^2)$ developed in the current section. Therefore, for a given input graph G, we always first apply a preprocessing that applies the above operations and removes all self-loops and multiple edges in the graph G. In consequence, we can assume, without loss of generality, that the input graph G contains neither self-loops nor multiple edges.

Algorithm-1 Feedback(G, V_1, V_2, k)
Input: $G = (V, E)$ is a graph with a forest bipartition (V_1, V_2), k is an integer.
Output: An FVS F of G such that $|F| \leq k$ and $F \subseteq V_1$; or report "No" (i.e., no such an FVS exists).

1. **if** $(k < 0)$ or $(k = 0$ and G is not a forest$)$ **then return** "No";
2. **if** $(k \geq 0)$ and G is a forest **then return** $\emptyset$;
3. **if** a vertex w in V_1 has at least two neighbors in V_2 **then**
3.1. **if** two of the neighbors of w in V_2 belong to the same tree in $G[V_2]$ **then**
$F_1 = \textbf{Feedback}(G - w, V_1 - \{w\}, V_2, k-1)$;
if $F_1 =$ "No" **then return** "No" **else return** $F_1 \cup \{w\}$;
3.2. **else**
$F_1 = \textbf{Feedback}(G - w, V_1 - \{w\}, V_2, k-1)$;
$F_2 = \textbf{Feedback}(G, V_1 - \{w\}, V_2 \cup \{w\}, k)$;
if $F_1 \neq$ "No" **then return** $F_1 \cup \{w\}$ **else return** F_2;
4. **else** pick any vertex w that has degree ≤ 1 in $G[V_1]$;
4.1. **if** w has degree ≤ 1 in the original graph G **then**
return Feedback$(G - w, V_1 - \{w\}, V_2, k)$;
4.2. **else return Feedback**$(G, V_1 - \{w\}, V_2 \cup \{w\}, k)$.

Fig. 15. Algorithm for the P-FEEDBACK VERTEX SET problem

The algorithm, **Feedback**(G, V_1, V_2, k), for the P-F-BIPARTITION FEEDBACK VERTEX SET problem is given in Figure 15. We first discuss the correctness of the algorithm. The correctness of step 1 and step 2 of the algorithm is obvious. Now consider step 3. Let w be a vertex in V_1 that has at least two neighbors in V_2.

If the vertex w has two neighbors in V_2 that belong to the same tree T in the induced subgraph $G[V_2]$, then the tree T plus the vertex w contains at least one cycle.

Since our search for an FVS is restricted to V_1, the only way to break the cycles in $T \cup \{w\}$ is to include the vertex w in the objective FVS. Moreover, the objective FVS of size at most k exists in G if and only if the remaining graph $G - w$ has an FVS of size at most $k-1$ in the subset $V_1 - \{w\}$ (note that $(V_1 - \{w\}, V_2)$ is a valid forest bipartition of the graph $G - w$). Therefore, step 3.1 correctly handles this case.

If no two neighbors of the vertex w belong to the same tree in the induced subgraph $G[V_2]$, then the vertex w is either in the objective FVS or not in the objective FVS. If w is in the objective FVS, then we should be able to find an FVS F_1 in the graph $G-w$ such that $|F_1| \leq k-1$ and $F_1 \subseteq V_1 - \{w\}$ (again note that $(V_1 - \{w\}, V_2)$ is a valid forest bipartition of the graph $G - w$). On the other hand, if w is not in the objective FVS, then the objective FVS for G must be contained in the subset $V_1 - \{w\}$. Also note that in this case, the subgraph $G[V_2 \cup \{w\}]$ induced by the subset $V_2 \cup \{w\}$ is still a forest since no two neighbors of w in V_2 belong to the same tree in $G[V_2]$. In consequence, $(V_1 - \{w\}, V_2 \cup \{w\})$ still makes a valid forest bipartition for the graph G. Therefore, step 3.2 handles this case correctly.

Now we consider step 4. At this point, every vertex in V_1 has at most one neighbor in V_2. Moreover, since the induced subgraph $G[V_1]$ is a forest, there must be a vertex w in V_1 that has degree at most 1 in $G[V_1]$ (note that V_1 cannot be empty at this point since otherwise the algorithm would have stopped at step 2). If the vertex w also has degree at most 1 in the original graph G, then removing w does not help breaking any cycles in G. Therefore, the vertex w can be discarded. This case is correctly handled by step 4.1. Otherwise, the vertex w has degree at most 1 in the induced subgraph $G[V_1]$ but has degree larger than 1 in the original graph G. Observing that w has at most one neighbor in V_2, we can derive that the degree of w in the original graph G must be exactly 2. Moreover, w has exactly two neighbors u and v such that v is in V_1 and u is in V_2.

Since the vertex w has degree 2 in the original graph G, and the vertex v is adjacent to w, we have that every cycle in G that contains w has to contain v. In consequence, if w is contained in the objective FVS, then we can simply replace it by v. Therefore, in this case, we can safely assume that the vertex w is not in the objective FVS. This can be easily implemented by moving the vertex w from the set V_1 to the set V_2, and recursively working on the modified instance, as given in step 4.2 of the algorithm (note that $(V_1 - \{w\}, V_2 \cup \{w\})$ is a valid forest bipartition of the graph G, because by our assumption, the vertex w will be a degree-1 vertex in the induced subgraph $G[V_2 \cup \{w\}]$).

Now we are ready to present the following lemma.

Lemma B.1 *The algorithm* **Feedback**(G, V_1, V_2, k) *correctly solves the* p-f-bipartition feedback vertex set *problem. The running time of the algorithm is* $O(2^{k+l}n^2)$*, where* n *is the number of vertices in* G*, and* l *is the number of connected components in the induced subgraph* $G[V_2]$*.*

Proof. The correctness of the algorithm has been verified by the above discussion. Now we consider the complexity of the algorithm.

The recursive execution of the algorithm can be described as a search tree $\mathcal{T}$. We first count the number of leaves in the search tree $\mathcal{T}$. Note that only step 3.2 of the algorithm corresponds to branches in the search tree $\mathcal{T}$. Let $T(k, l)$ be the total number of leaves in the search tree $\mathcal{T}$ for the algorithm **Feedback**(G, V_1, V_2, k), where l is the number of connected components (i.e., trees) in the forest $G[V_2]$. Inductively, the number of leaves in the search tree $\mathcal{T}_1$ corresponding to the recursive call **Feedback**$(G - w, V_1 - \{w\}, V_2, k - 1)$ is at most $T(k - 1, l)$. Moreover, we assumed at step 3.2 that w has at least two neighbors in V_2 and that no two neighbors of w

in V_2 belong to the same tree in $G[V_2]$. Therefore, the vertex w "merges" at least two trees in $G[V_2]$ into a single tree in $G[V_2 \cup \{w\}]$. Hence, the number of trees in $G[V_2 \cup \{w\}]$ is at most $l-1$. In consequence, the number of leaves in the search tree $\mathcal{T}_2$ corresponding to the recursive call **Feedback**$(G, V_1 - \{w\}, V_2 \cup \{w\}, k)$ is at most $T(k, l-1)$. This gives the following recurrence relation:

$$T(k,l) \leq T(k-1,l) + T(k,l-1).$$

Also note that none of the (non-branching) recursive calls in the algorithm (steps 3.1, 4.1, and 4.2) would increase the values k and l, and that $T(0,l) = 1$ for all l and $T(k,0) = 1$ for all k (by steps 1-2). From all these facts, we can easily derive that $T(k,l) = O(2^{k+l})$.

Finally, observe that along each root-leaf path in the search tree $\mathcal{T}$, the total number of executions of steps 1, 2, 3, 3.1, 4.1, and 4.2 of the algorithm is $O(n)$ because each of these steps either stops immediately, or reduces the size of the set V_1 by at least 1 (and the size of V_1 is never increased during the execution of the algorithm). It remains to explain how each of the steps can be executed in $O(n)$ time.

Before the first call to the **Feedback** algorithm, we use $O(n^2)$ time, because this will happen only once. The three graphs, $G_1 = G[V_1], G_2 = G[V_2]$, and $G_{12} = (V, E - (E(G_1) \cup E(G_2)))$ can be trivially constructed in $O(n^2)$ time. G_1 and G_2 are forests, and G_{12} is a bipartite graph with the two vertex sets V_1 and V_2 as independent sets.

Steps 1, 2, 4.1, and 4.2 can be easily performed in $O(n)$ time. For step 3, we simply search for a vertex of V_1 that has degree at least 2 in G_{12}, and for step 4 we search in G_1 for a leaf (vertex of degree at most 1). The condition for step 3.1 is that no two neighbors of w belong to the same tree in $G[V_2]$, which can be checked by

simply marking each neighbor of w, and doing a search in the forest $G[V_2]$.

Each of the steps 3.1, 3.2, 4.1, and 4.2 changes one or more of the graphs G_1, G_2, G_{12}, and we have to argue that these manipulations can also be done in $O(n)$ time. Looking closely at these steps, we can observe that only two operations are required. The first is to delete a vertex in V_1, which corresponds to deleting the vertex and all incident edges in G_1 and G_{12}. The second operation is to move a vertex w from V_1 to V_2, which corresponds to deleting w from G_1 and updating G_2 and G_{12} as follows: add w to $V(G_2)$ and to V_2 of G_{12}, and read the set of edges incident to w in G, and add edges between w and vertices in V_2 to G_2 and between w and V_1 to G_{12}. Using double linked lists and pointers it is possible to delete a vertex and all incident edges in $O(n)$ time, and to insert edges in $O(1)$ time.

Therefore, the computation time along each root-leaf path in the search tree $\mathcal{T}$ is $O(n^2)$. In conclusion, the running time of the algorithm **Feedback**(G, V_1, V_2, k) is $O(2^{k+l}n^2)$. This completes the proof of the lemma. □

Following the idea of *iterative compression* proposed by Reed et al. [107], we formulate the following problem:

> FEEDBACK VERTEX SET REDUCTION: given a graph G and an FVS F of size $k+1$ for G, either construct an FVS of size at most k for G, or report that no such an FVS exists.

Lemma B.2 *The* FEEDBACK VERTEX SET REDUCTION *problem on an n-vertex graph G can be solved in time $O(5^k n^2)$.*

PROOF. We use the algorithm **Feedback** to solve the FEEDBACK VERTEX SET REDUCTION problem. Let F be the FVS of size $k+1$ in the graph $G = (V, E)$. Every FVS F' of size at most k for G is a union of a subset F_1 of at most $k - j$ vertices in

$V - F$ and a subset F_2 of j vertices in F, for some integer j, $0 \leq j \leq k$. Note that since we assume that no vertex in $F - F_2$ is in the FVS F', the induced subgraph $G[F - F_2]$ must be a forest. For each j, $0 \leq j \leq k$, we enumerate all subsets of j vertices in F. For each such subset F_2 in F such that $G[F - F_2]$ is a forest, we seek a subset F_1 of at most $k - j$ vertices in $V - F$ such that $F_1 \cup F_2$ is an FVS in G.

Fix a subset F_2 in F, where $|F_2| = j$. Note that the graph G has an FVS $F_1 \cup F_2$ of size at most k, where $F_1 \subseteq V - F$, if and only if the subset F_1 of $V - F$ is an FVS for the graph $G - F_2$ and $|F_1| \leq k - j$. Therefore, to solve the original problem, we construct an FVS F_1 for the graph $G - F_2$ such that $|F_1| \leq k - j$ and $F_1 \subseteq V - F$.

Since F is an FVS for G, we have that the induced subgraph $G[V - F] = G - F$ is a forest. Moreover, by our assumption, the induced subgraph $G[F - F_2]$ is also a forest. Note that $(V - F) \cup (F - F_2) = V - F_2$, which is the vertex set for the graph $G' = G - F_2$. Therefore, $(V - F, F - F_2)$ is a forest bipartition of the graph G'. Thus, an FVS F_1 for the graph G' such that $|F_1| \leq k - j$ and $F_1 \subseteq V - F$ can be constructed by the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$.

Since $|F| = k + 1$ and $|F_2| = j$, we have that $|F - F_2| = k + 1 - j$. Therefore, the forest $G[F - F_2]$ contains at most $k + 1 - j$ connected components. By Lemma B.1, the running time of the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$ is $O(2^{(k-j)+(k+1-j)}n^2) = O(4^{k-j}n^2)$. Now for all integers j, $0 \leq j \leq k$, we enumerate all subsets F_2 of j vertices in F and apply the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$ for those F_2 such that $G[F - F_2]$ is a forest. As we discussed above, the graph G has an FVS of size at most k if and only if for some $F_2 \subseteq F$, the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$ produces an FVS F_1 for the graph G'. The running time of this procedure is

$$\sum_{j=0}^{k} \binom{k+1}{j} \cdot O(4^{k-j}n^2) = \sum_{j=0}^{k} \binom{k+1}{k-j+1} O(4^{k-j+1}n^2) = O(5^k n^2).$$

This completes the proof of the lemma. □

Finally, by combining Lemma B.2 with iterative compression, we obtain the main result of this subsection.

Theorem B.3 *The* P-FEEDBACK VERTEX SET *problem on an n-vertex graph is solvable in time $O(5^k kn^2)$.*

PROOF. To solve the P-FEEDBACK VERTEX SET problem, for a given graph $G = (V, E)$, we start by applying Bafna et al.'s 2-approximation algorithm for the MINIMUM FEEDBACK VERTEX SET problem [7]. This algorithm runs in $O(n^2)$ time, and either returns an FVS F' of size at most $2k$, or verifies that no FVS of size at most k exists. If no FVS is returned, the algorithm is terminated with the conclusion that no FVS of size at most k exists. In the case of the opposite result, we use any subset V' of k vertices in F', and put $V_0 = V' \cup (V - F')$. Of course, the induced subgraph $G[V_0]$ has an FVS of size k, namely the set V' ($G[V_0 - V']$ is a forest). Let $F' - V' = \{v_1, v_2, \ldots, v_{|F'|-k}\}$, and let $V_i = V_0 \cup \{v_1, \ldots, v_i\}$ for $i \in \{0, 1, \ldots, |F'| - k\}$. Inductively, suppose that we have constructed an FVS F_i for the graph $G[V_i]$, where $|F_i| = k$. Then the set $F'_{i+1} = F_i \cup \{v_{i+1}\}$ is obviously an FVS for the graph $G[V_{i+1}]$ and $|F'_{i+1}| = k + 1$.

Now the pair $(G[V_{i+1}], F'_{i+1})$ is an instance for the FEEDBACK VERTEX SET REDUCTION problem. Therefore, in time $O(5^k n^2)$, we can either construct an FVS F_{i+1} of size k for the graph $G[V_{i+1}]$, or report that no such an FVS exists. Note that if the graph $G[V_{i+1}]$ does not have an FVS of size k, then the original graph G cannot have an FVS of size k. In this case, we simply stop and claim the non-existence of an FVS of size k for the original graph G. On the other hand, with an FVS F_{i+1} of size k for the graph $G[V_{i+1}]$, our induction proceeds to the next graph $G[V_{i+1}]$, until we reach

the graph $G = G[V_{|F'|-k}]$. This process runs in time $O(5^k kn^2)$ since $|F'| - k \leq k$, and solves the P-FEEDBACK VERTEX SET problem. □

4. Feedback vertex set on weighted graphs

In this subsection, we discuss the PW-FEEDBACK VERTEX SET problem. A weighted graph $G = (V, E)$ is an undirected graph, where each vertex $u \in V$ is assigned a *weight* that is a positive real number. The weight of a vertex set $A \subseteq V$ is the sum of the vertex weights of all vertices in A. We denote by $|A|$ the cardinality of A. The PW-FEEDBACK VERTEX SET problem is formally defined as follows:

> PW-FEEDBACK VERTEX SET: given a weighted graph G and an integer k, either find an FVS F of minimum weight for G such that $|F| \leq k$, or report that no FVS of size at most k exists in G.

The algorithm for the weighted case has several similarities with the unweighted case, but has also a significant difference. The difference is that step 4.2 of **Algorithm-1** becomes invalid for weighted graphs. A degree-2 vertex w in the set V_1 cannot simply be excluded from the objective FVS. If the weight of w is smaller than that of its parent v in $G[V_1]$, it may become necessary to include w instead of v in the objective FVS.

To overcome this difficulty, we introduce a new partition structure of the vertices in a weighted graph.

Definition A triple (V_0, V_1, V_2) is an *independent-forest partition* (*IF-partition*) of a graph $G = (V, E)$ if (V_0, V_1, V_2) is a partitioning of V (i.e., $V_0 \cup V_1 \cup V_2 = V$, and V_0, V_1, and V_2 are pairwise disjoint), such that

(1) $G[V_1]$ and $G[V_2]$ are forests;

(2) $G[V_0]$ is an independent set;

(3) Every vertex u in V_0 is of degree 2 in G, with both neighbors in V_2.

The following problem is the analogue of the P-F-BIPARTITION FEEDBACK VERTEX SET problem on unweighted graphs.

> PW-IF-PARTITION FEEDBACK VERTEX SET: given a weighted graph G, an IF-partition (V_0, V_1, V_2) of G, and an integer k, either find an FVS F of minimum weight for G that satisfies the conditions $|F| \leq k$ and $F \subseteq V_0 \cup V_1$, or report that no such an FVS exists.

To develop and analyze our algorithm for the PW-IF-PARTITION FEEDBACK VERTEX SET problem, we need the following concept of *measure* for the problem instances. For a vertex subset V' in the graph G, we will denote by $\#c(V')$ the number of connected components in the induced subgraph $G[V']$.

Definition Let (G, V_0, V_1, V_2, k) be an instance of the PW-IF-PARTITION FEEDBACK VERTEX SET problem with an IF-partition (V_0, V_1, V_2). The *deficiency* of the instance (G, V_0, V_1, V_2, k) is defined as

$$\tau(k, V_0, V_1, V_2) = k - (|V_0| - \#c(V_2) + 1),$$

Intuitively, $\tau(k, V_0, V_1, V_2)$ of the instance (G, V_0, V_1, V_2, k) is an upper bound on the number of vertices in the objective FVS that are in the set V_1 (this will

become clearer during our discussion below). Our algorithm for the PW-IF-PARTITION FEEDBACK VERTEX SET problem is based on the following observation: once we have correctly determined all vertices in the objective FVS that are in the set V_1, the problem will become solvable in polynomial time, as shown in the following lemma.

Lemma B.4 *Let (G, V_0, V_1, V_2, k) be an instance of the* PW-IF-PARTITION FEEDBACK VERTEX SET *problem with an IF-partition (V_0, V_1, V_2) of an n-vertex graph G. If $V_1 = \emptyset$, or $V_2 = \emptyset$, or $\tau(k, V_0, V_1, V_2) \leq 0$, then a solution to the instance (G, V_0, V_1, V_2, k) can be constructed in time $O(n^2)$.*

PROOF. First of all, note that if $k < 0$, then the solution to the instance is "No": we cannot remove a negative number of vertices from G. Thus, in the following discussion, we assume that $k \geq 0$.

If $V_2 = \emptyset$, then by the definition, V_0 should also be an empty set. Thus, the graph $G = G[V_1]$ is a forest, and the solution to the instance (G, V_0, V_1, V_2, k) is the empty set $\emptyset$.

Now consider the case $V_1 = \emptyset$. Then we need to find a minimum-weight subset of at most k vertices in the set V_0 whose removal from the graph $G = G[V_0 \cup V_2]$ results in a forest. We will solve this problem by creating a new graph $\mathcal{H}$, such that an optimal solution for (G, V_0, V_1, V_2, k) corresponds to the edges which are not used in a maximum weight spanning tree of $\mathcal{H}$.

Construct a new graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where each vertex μ in $\mathcal{V}$ corresponds to a connected component in the induced subgraph $G[V_2]$, and each edge $[\mu, \nu]$ in $\mathcal{E}$ corresponds to a vertex v in the set V_0 such that the two neighbors of v are in the connected components in $G[V_2]$ that correspond to the two vertices μ and ν, respectively, in $\mathcal{H}$. Intuitively, the graph $\mathcal{H}$ can be obtained from the graph $G =$

$G[V_0 \cup V_2]$ by "shrinking" each connected component in $G[V_2]$ into a single vertex and "smoothing" each degree-2 vertex in V_0 (note that the graph $\mathcal{H}$ may contain multiple edges and self-loops). Moreover, we give each edge in $\mathcal{H}$ a weight that is equal to the weight of the corresponding vertex in V_0. Thus, the graph $\mathcal{H}$ is a graph with edge weights. Observe that there is a one-to-one correspondence between the connected components in the graph $\mathcal{H}$ and the connected components in the graph G. Moreover, since each connected component in the induced subgraph $G[V_2]$ is a tree, a connected component in the graph $\mathcal{H}$ is a tree if and only if the corresponding connected component in the graph G is a tree. Most importantly, removing a vertex in V_0 in the graph G corresponds to removing the corresponding edge in the graph $\mathcal{H}$. Therefore, the problem of constructing a minimum-weight vertex set in V_0 whose removal from G results in a forest is equivalent to the problem of constructing a minimum-weight edge set in the graph $\mathcal{H}$ whose removal from $\mathcal{H}$ results in a forest.

Let $\mathcal{H}_1, \ldots, \mathcal{H}_s$ be the connected components of the graph $\mathcal{H}$, where for each i, the component $\mathcal{H}_i$ has n_i vertices and m_i edges. An edge set $\mathcal{E}_i$ in $\mathcal{H}_i$ whose removal from $\mathcal{H}_i$ results in a forest is of the minimum weight if and only if the complement graph $\mathcal{H}_i - \mathcal{E}_i$ is a spanning tree of the maximum weight in $\mathcal{H}_i$. Thus, the union $\mathcal{E}' = \bigcup_{i=1}^{s}(\mathcal{H}_i - \mathcal{T}_i)$ is a minimum-weight edge set whose removal from $\mathcal{H}$ results in a forest, where for each i, $\mathcal{T}_i$ is a maximum-weight spanning tree in the graph $\mathcal{H}_i$. Since the maximum-weight spanning tree $\mathcal{T}_i$ in $\mathcal{H}_i$ can be constructed in time $O(n_i^2)$ by modifying the well-known minimum spanning tree algorithms (the algorithms work even for graphs with self-loops and multiple edges) [31], we conclude that the minimum-weight edge set $\mathcal{E}'$ in $\mathcal{H}$ can be constructed in time $\sum_{i=1}^{s} O(n_i^2) = O(n^2)$. Also note that the number of edges in the set $\mathcal{E}'$ is equal to $\sum_{i=1}^{s}(m_i - n_i + 1) = |\mathcal{E}| - |\mathcal{V}| + s$.

Correspondingly, in case $V_1 = \emptyset$, each minimum-weight FVS in V_0 for the graph G contains exactly $|\mathcal{E}| - |\mathcal{V}| + s$ vertices, and such an FVS can be constructed in time $O(n^2)$. Note that $|\mathcal{E}| = |V_0|$, $|\mathcal{V}|$ is equal to the number $\#c(V_2)$ of connected components in the induced subgraph $G[V_2]$, and s (which is the number of connected components in $\mathcal{H}$) is equal to the number $\#c(G)$ of connected components in the graph $G = G[V_0 \cup V_2]$. Thus, each minimum-weight FVS in V_0 for the graph G contains exactly $|V_0| - \#c(V_2) + \#c(G)$ vertices, and such a minimum-weight FVS can be constructed in time $O(n^2)$. Therefore, for the given instance (G, V_0, V_1, V_2, k) of the PW-IF-PARTITION FEEDBACK VERTEX SET problem with $V_1 = \emptyset$, the solution is "No" if $k < |V_0| - \#c(V_2) + \#c(G)$; and the solution is an $O(n^2)$ time constructible FVS of $|V_0| - \#c(V_2) + \#c(G)$ vertices in V_0 if $k \geq |V_0| - \#c(V_2) + \#c(G)$.

This completes the proof that when $V_1 = \emptyset$, a solution to the instance (G, V_0, V_1, V_2, k) can be constructed in time $O(n^2)$.

Now consider the case $\tau(k, V_0, V_1, V_2) \leq 0$. If $V_2 = \emptyset$, then by the first part of the proof, the lemma holds. Thus, we assume that $V_2 \neq \emptyset$. As analyzed above, to break every cycle in the induced subgraph $G[V_0 \cup V_2]$ we have to remove at least $|V_0| - \#c(V_2) + \#c(V_0 \cup V_2)$ vertices in the set V_0. Therefore, if $\tau(k, V_0, V_1, V_2) \leq 0$, then $k \leq |V_0| - \#c(V_2) + 1 \leq |V_0| - \#c(V_2) + \#c(V_0 \cup V_2)$ (note that $V_2 \neq \emptyset$ so $\#c(V_0 \cup V_2) \geq 1$). Thus, in this case, all k vertices in the objective FVS must be in the set V_0 in order to break all cycles in the induced subgraph $G[V_0 \cup V_2]$, and no vertex in the objective FVS can be in the set V_1. Hence, if the induced subgraph $G[V_1 \cup V_2]$ contains a cycle, then the solution to the instance is "No". On the other hand, suppose that $G[V_1 \cup V_2]$ is a forest, then the graph G has another IF-partition (V_0', V_1', V_2'), where $V_0' = V_0$, $V_1' = \emptyset$, and $V_2' = V_1 \cup V_2$. It is easy to verify that in this case the instance (G, V_0', V_1', V_2', k) with the IF-partition (V_0', V_1', V_2') has the same solution set as the instance (G, V_0, V_1, V_2, k) with the IF-partition (V_0, V_1, V_2). Since

$V_1' = \emptyset$, by the second part of the proof, a solution to the instance (G, V_0', V_1', V_2', k) with the IF-partition (V_0', V_1', V_2') can be constructed in time $O(n^2)$. This completes the proof of the lemma. □

We are now in a position to introduce our main algorithm, which is given in Figure 16 and solves the PW-IF-PARTITION FEEDBACK VERTEX SET problem. The subroutine **min-w**(S_1, S_2) on two vertex subsets S_1 and S_2 in the algorithm returns among S_1 and S_2 the one with a smaller weight (or any one of them if the weights are tied). To simplify our descriptions, we take the conventions that "No" is a special vertex set of an infinitely large weight and that any set plus "No" gives a "No". Therefore, the value of **min-w**(S_1, S_2) will be (1) "No" if both S_1 and S_2 are "No"; (2) S_1 if S_2 is "No"; (3) S_2 if S_1 is "No"; and (4) the one of smaller weight among S_1 and S_2 if both S_1 and S_2 are not "No".

For each tree in the forest $G[V_1]$, we fix a root so that we can talk about the "lowest leaf" in a tree in $G[V_1]$.

Lemma B.5 *In time $O(2^{\tau(k,V_0,V_1,V_2)}n^2)$, the algorithm* **W-Feedback**(G, V_0, V_1, V_2, k) *correctly solves the* PW-IF-PARTITION FEEDBACK VERTEX SET *problem, where n is the number of vertices in the graph G.*

PROOF. We first verify the correctness of the algorithm. Step 1 of the algorithm is justified by Lemma B.4. Justifications for steps 2, 3, 4, 4.1, and 4.2 are exactly the same as that for steps 1, 4.1, 3, 3.1, and 3.2 in **Algorithm-1** for unweighted graphs. Now consider step 5. When the algorithm reaches step 5, the following conditions hold:

(1) the sets V_1 and V_2 are not empty;

(2) every vertex in the set V_1 has degree at least 2 in the graph G; and

Algorithm-2 W-Feedback(G, V_0, V_1, V_2, k)
Input: $G = (V, E)$ is a graph with an IF-partition (V_0, V_1, V_2), k is an integer.
Output: a minimum-weight FVS F of G such that $|F| \leq k$ and $F \subseteq V_0 \cup V_1$;
or report "No" (i.e., no such an FVS exists).

1. **if** $(V_1 = \emptyset)$ or $(V_2 = \emptyset)$ or $(\tau(k, V_0, V_1, V_2) \leq 0)$ **then**
solve the problem in time $O(n^2)$;
2. **if** $(k < 0)$ or $(k = 0$ and G is not a forest$)$ **then return** "No";
3. **if** a vertex w in V_1 has degree less than 2 in G **then**
return W-Feedback$(G - w, V_0, V_1 - \{w\}, V_2, k)$;
4. **else if** a vertex w in V_1 has at least two neighbors in V_2 **then**
4.1 **if** two neighbors of w are in the same tree of $G[V_2]$ **then**
return $(\{w\} \cup$ **W-Feedback**$(G - w, V_0, V_1 - \{w\}, V_2, k - 1))$;
4.2 **else**
$F_1 =$ **W-Feedback**$(G - w, V_0, V_1 - \{w\}, V_2, k - 1)$;
$F_2 =$ **W-Feedback**$(G, V_0, V_1 - \{w\}, V_2 \cup \{w\}, k)$;
return min-w$(F_1 \cup \{w\}, F_2)$;
5 **else** pick a lowest leaf w_1 in any tree T in $G[V_1]$;
let w be the parent of w_1 in T, and let $w_1, \ldots, w_t$ be the children of w in T;
5.1 **if** (w has a neighbor in V_2) or (w has more than one child in T) **then**
$F_1 =$ **W-Feedback**$(G - w, V_0, V_1 - \{w\}, V_2, k - 1)$;
$F_2 =$ **W-Feedback**$(G, V_0 \cup \{w_1, \ldots, w_t\}, V_1 - \{w, w_1, \ldots, w_t\}, V_2 \cup \{w\}, k)$;
return min-w$(F_1 \cup \{w\}, F_2)$;
5.2 **else**
if the weight of w_1 is larger than the weight of w **then**
return W-Feedback$(G, V_0, V_1 - \{w_1\}, V_2 \cup \{w_1\}, k)$;
else return W-Feedback$(G, V_0 \cup \{w_1\}, V_1 - \{w, w_1\}, V_2 \cup \{w\}, k)$.

Fig. 16. Algorithm for the PW-FEEDBACK VERTEX SET problem

(3) every vertex in the set V_1 has at most one neighbor in the set V_2.

Condition (1) holds because of step 1; condition (2) holds because of step 3; and condition (3) holds because of step 4.

By condition (1) and because the induced subgraph $G[V_1]$ is a forest, step 5 can always pick the vertex w_1. By conditions (2) and (3), the vertex w_1 has a unique neighbor in V_2. Also by conditions (2) and (3), the vertex w_1 must have a parent w in the tree T in $G[V_1]$. In consequence, the vertex w_1 has degree exactly 2 in G. Finally, since w_1 is the lowest leaf in the tree T, all children $w_1, \ldots, w_t$ of w in the

tree T are also leaves in T. By conditions (2) and (3) again, each child w_i of w has a unique neighbor in the set V_2, and every child w_i of w has degree exactly 2 in the graph G.

Step 5.1 simply branches on the vertex w. To include the vertex w in the objective FVS, we simply remove w from the graph G (and from the set V_1), and recursively look for an FVS in $V_0 \cup (V_1 - \{w\})$ of size at most $k-1$. Note that in this case, the sets V_0 and V_2 are unchanged, and the triple $(V_0, V_1 - \{w\}, V_2)$ obviously makes a valid IF-partition for the graph $G - w$. On the other hand, to exclude the vertex w from the objective FVS, we move w from V_1 to V_2. First note that since the vertex w has at most one neighbor in V_2, the induced subgraph $G[V_2 \cup \{w\}]$ is still a forest. Moreover, since all children $w_1, \ldots, w_t$ of w have degree 2 in the graph G and each w_i has a unique neighbor in the set V_2, after moving w from V_1 to V_2, all these degree-2 vertices $w_1, \ldots, w_t$ have their both neighbors in the set $V_2 \cup \{w\}$. Therefore, these vertices $w_1, \ldots, w_t$ now can be moved to the set V_0. In particular, the triple $(V_0 \cup \{w_1, \ldots, w_t\}, V_1 - \{w, w_1, \ldots, w_t\}, V_2 \cup \{w\})$ is a valid IF-partition of the vertex set of the graph G. This recursive branching is implemented by the two recursive calls in step 5.1.

If we reach step 5.2, then the two conditions in step 5.1 do not hold. Therefore, in addition to conditions (1)-(3), the following two conditions also hold:

(4) the vertex w has no neighbor in V_2; and

(5) the vertex w has a unique child w_1 in the tree T.

By conditions (2), (4), and (5), the vertex w has degree exactly 2 in the graph G (and w is not the root of the tree T). Therefore, the vertices w_1 and w are two adjacent degree-2 vertices in the graph G. Observe that in this case, a cycle in the graph G contains the vertex w_1 *if and only if* it also contains the vertex w.

Therefore, we can safely assume that the one of larger weight among w_1 and w is not in the objective FVS. If the larger weight vertex is w_1, then the first recursive call in step 5.2 is executed, which moves w_1 from set V_1 to set V_2 (note that the triple $(V_0, V_1 - \{w_1\}, V_2 \cup \{w_1\})$ is a valid IF-partition of G because w_1 has a unique neighbor in V_2). If the larger weight vertex is w, then the second recursive call in step 5.2 is executed, which moves w from V_1 to V_2. Note that since both neighbors of the degree-2 vertex w_1 are in the set $V_2 \cup \{w\}$, after adding w to the set V_2, we can also move the vertex w_1 from V_1 to V_0. Thus, the triple $(V_0 \cup \{w_1\}, V_1 - \{w, w_1\}, V_2 \cup \{w\})$ is a valid IF-partition of the graph G.

We also remark that by our assumption, the input graph G contains neither multiple edges nor self-loops. Moreover, the graph in each of the recursive calls in the algorithm is either the original G, or G with a vertex deleted. Therefore, the graph in each of the recursive calls in the algorithm also contains neither multiple edges nor self-loops.

Since all possible cases are covered in the algorithm, we conclude that when the algorithm **W-Feedback** stops, it must output a correct solution to the given instance (G, V_0, V_1, V_2, k).

To analyze the running time, as in the unweighted case, we first count the number of leaves in the search tree corresponding to the execution of the algorithm. Let $T(k, V_0, V_1, V_2)$ be the number of leaves in the search tree for algorithm **W-Feedback**(G, V_0, V_1, V_2, k). We prove by induction on the value $\tau(k, V_0, V_1, V_2)$ that $T(k, V_0, V_1, V_2) \leq max(1, 2^{\tau(k, V_0, V_1, V_2)})$. First of all, if $\tau(k, V_0, V_1, V_2) \leq 0$, then by step 1 of the algorithm, we have $T(k, V_0, V_1, V_2) = 1$.

First consider the branching steps, i.e., step 4.2 and step 5.1. In case step 4.2 of the algorithm is executed, we have recursively

$$\begin{aligned} & T(k, V_0, V_1, V_2) \\ & \leq T(k-1, V_0, V_1 - \{w\}, V_2) + T(k, V_0, V_1 - \{w\}, V_2 \cup \{w\}). \end{aligned} \tag{6.4}$$

Since

$$\begin{aligned} \tau_1 &= \tau(k-1, V_0, V_1 - \{w\}, V_2) \\ &= (k-1) - (|V_0| - \#c(V_2) + 1) \\ &= \tau(k, V_0, V_1, V_2) - 1 \\ &< \tau(k, V_0, V_1, V_2), \end{aligned}$$

and

$$\begin{aligned} \tau_2 &= \tau(k, V_0, V_1 - \{w\}, V_2 \cup \{w\}) \\ &= k - (|V_0| - \#c(V_2 \cup \{w\}) + 1) \\ &\leq k - (|V_0| - (\#c(V_2) - 1) + 1) \\ &= \tau(k, V_0, V_1, V_2) - 1 \\ &< \tau(k, V_0, V_1, V_2), \end{aligned}$$

where we have used the fact $\#c(V_2 \cup \{w\}) \leq \#c(V_2) - 1$ because in this case, we assume that the vertex w has two neighbors in two different trees in $G[V_2]$, therefore, adding w to V_2 merges at least two connected components in $G[V_2]$ and reduces the number of connected components by at least 1.

Therefore, by the inductive hypothesis, $T(k-1, V_0, V_1 - \{w\}, V_2) \leq 2^{\tau_1}$, and $T(k, V_0, V_1 - \{w\}, V_2 \cup \{w\}) \leq 2^{\tau_2}$. Combining these with Inequality (6.4), we get

$$\begin{aligned} T(k, V_0, V_1, V_2) &\leq T(k-1, V_0, V_1 - \{w\}, V_2) + T(k, V_0, V_1 - \{w\}, V_2 \cup \{w\}) \\ &\leq 2^{\tau_1} + 2^{\tau_2} \\ &\leq 2^{\tau(k, V_0, V_1, V_2)-1} + 2^{\tau(k, V_0, V_1, V_2)-1} \\ &= 2^{\tau(k, V_0, V_1, V_2)} \end{aligned}$$

In conclusion, the induction goes through for step 4.2 of the algorithm.

Now we consider step 5.1, which is the least trivial case, and makes the major difference from the unweighted cases. Let $V_0' = V_0 \cup \{w_1, \ldots, w_t\}$, $V_1' = V_1 - \{w, w_1, \ldots, w_t\}$, and $V_2' = V_2 \cup \{w\}$. The execution of step 5.1 gives the following inequality:

$$T(k, V_0, V_1, V_2) \leq T(k-1, V_0, V_1 - \{w\}, V_2) + T(k, V_0', V_1', V_2').$$

As we have shown above, by the inductive hypothesis, we have

$$T(k-1, V_0, V_1 - \{w\}, V_2) \leq 2^{\tau(k, V_0, V_1, V_2)-1}. \tag{6.5}$$

To estimate the value $T(k, V_0', V_1', V_2')$, first note that $|V_0'| = |V_0| + t$. Moreover, at this point, we must have either that the vertex w has a neighbor in V_2 or that the vertex w has more than one child in the tree T in $G[V_1]$.

If w has a neighbor in V_2, then adding w to V_2 will "attach" the vertex w to a connected component in $G[V_2]$. In consequence, the number of connected components in $G[V_2]$ will be equal to that in $G[V_2 \cup \{w\}]$ (recall that w has only one neighbor in

V_2). In this case (note that $t \geq 1$), we have

$$\begin{aligned}\tau(k, V_0', V_1', V_2') &= k - (|V_0'| - \#c(V_2') + 1) \\ &= k - ((|V_0| + t) - \#c(V_2) + 1) \\ &\leq \tau(k, V_0, V_1, V_2) - 1\end{aligned}$$

Now let us assume that the vertex w has no neighbor in V_2 but has more than one child in the tree T in $G[V_1]$ (i.e., $t \geq 2$). Then $|V_0'| = |V_0| + t \geq |V_0| + 2$. In this case, adding the vertex w to the set V_2 increases the number of connected components in $G[V_2]$ by 1 (since w has no neighbor in V_2, the vertex w will become a single-vertex connected component in the induced subgraph $G[V_2 \cup \{w\}]$). That is, $\#c(V_2 \cup \{w\}) = \#c(V_2) + 1$. Therefore,

$$\begin{aligned}\tau(k, V_0', V_1', V_2') &= k - (|V_0'| - \#c(V_2') + 1) \\ &= k - ((|V_0| + t) - \#c(V_2 \cup \{w\}) + 1) \\ &\leq k - ((|V_0| + 2) - (\#c(V_2) + 1) + 1) \\ &\leq \tau(k, V_0, V_1, V_2) - 1\end{aligned}$$

In conclusion, in all cases in step 5.1, we have $\tau(k, V_0', V_1', V_2') \leq \tau(k, V_0, V_1, V_2) - 1$. Therefore, now we can apply the induction and get

$$T(k, V_0', V_1', V_2') \leq 2^{\tau(k, V_0', V_1', V_2')} \leq 2^{\tau(k, V_0, V_1, V_2) - 1}$$

Combining this with the inequalities (6.4) and (6.5), we conclude that

$$T(k, V_0, V_1, V_2) \leq 2^{\tau(k, V_0, V_1, V_2)}$$

holds for the case of step 5.1.

We should also remark that it can be verified that for all non-branching recursive calls in the algorithm, i.e., steps 3, 4.1, and 5.2, the instance deficiency is never increased. In particular, if the first recursive call in step 5.2 is executed, then since the vertex w_1 has a unique neighbor in V_2, $\#c(V_2) = \#c(V_2 \cup \{w_1\})$. Thus,

$$\begin{aligned}\tau(k, V_0, V_1 - \{w_1\}, V_2 \cup \{w_1\}) &= k - (|V_0| - \#c(V_2 \cup \{w_1\}) + 1) \\ &= k - (|V_0| - \#c(V_2) + 1) \\ &= \tau(k, V_0, V_1, V_2).\end{aligned}$$

If the second recursive call in step 5.2 is executed, then $\#c(V_2 \cup \{w\}) = \#c(V_2) + 1$ because w has no neighbor in V_2 and w will become a single-vertex connected component in the induced subgraph $G[V_2 \cup \{w\}]$. Therefore,

$$\begin{aligned}& \tau(k, V_0 \cup \{w_1\}, V_1 - \{w, w_1\}, V_2 \cup \{w\}) \\ = \;& k - (|V_0 \cup \{w_1\}| - \#c(V_2 \cup \{w\}) + 1) \\ = \;& k - ((|V_0| + 1) - (\#c(V_2) + 1) + 1) \\ = \;& \tau(k, V_0, V_1, V_2).\end{aligned}$$

Summarizing all the above discussions, we complete the inductive proof that the number of leaves in the search tree for the algorithm **W-Feedback**(G, V_0, V_1, V_2, k) is at most $2^{\tau(k, V_0, V_1, V_2)}$.

In the same way as in the proof for the unweighted case, we observe that along each root-leaf path in the search tree, the total number of executions of steps 1, 2, 3, 4, 4.1, 4.2, 5, 5.1, and 5.2 of the algorithm is $O(n)$ because each of these steps either stops immediately, or reduces the size of the set V_1 by at least 1. Step 1 is only preformed in leaf nodes of the tree, and thus only adds $O(n^2)$ time to the total. By similar arguments as the one used for the unweighted case, all steps except

step 1 can be preformed in $O(n)$ time. Therefore, the running time of the algorithm **W-Feedback**(G, V_0, V_1, V_2, k) is $O(2^{\tau(k,V_0,V_1,V_2)}n^2)$. □

With Lemma B.5, we can now proceed in the same way as for the unweighted case to solve the original PW-FEEDBACK VERTEX SET problem. Consider the following weighted version of the FEEDBACK VERTEX SET REDUCTION problem.

> W-FEEDBACK VERTEX SET REDUCTION: given a weighted graph G and an FVS F of size $k+1$ for G, either construct an FVS F' of minimum weight that satisfies $|F'| \leq k$, or report that no such an FVS exists.

Note that in the definition of W-FEEDBACK VERTEX SET REDUCTION, we do not require that the given FVS F of size $k+1$ have the minimum weight.

Lemma B.6 *The* W-FEEDBACK VERTEX SET REDUCTION *problem on an n-vertex graph is solvable in time $O(5^k n^2)$.*

PROOF. The proof proceeds similarly to the proof of Lemma B.2. For the given FVS F of size $k+1$ in the graph $G = (V, E)$, every FVS F' of size at most k for G (including the one with the minimum weight) is a union of a subset F_1 of at most $k-j$ vertices in $V - F$ and a subset F_2 of j vertices in F, for some integer j, $0 \leq j \leq k$, where $(V - F, F - F_2)$ is a forest bipartition of the graph $G_0 = G - F_2$. Therefore, we can enumerate all subsets F_2 of j vertices in F, for each j, $0 \leq j \leq k$, such that $(V - F, F - F_2)$ is a forest bipartition of the graph $G_0 = G - F_2$, and construct the minimum-weight FVS F_0 of G_0 satisfying $|F_0| \leq k-j$. Note that the forest bipartition $(V - F, F - F_2)$ of G_0 is in fact a special IF-partition (V_0, V_1, V_2) of G_0, where $V_0 = \emptyset$, $V_1 = V - F$, and $V_2 = F - F_2$. Therefore, by Lemma B.5, a minimum-weight FVS

F_0 of G_0 satisfying $|F_0| \leq k - j$ can be constructed in time

$$O(2^{\tau(k-j,V_0,V_1,V_2)}n^2) = O(2^{(k-j)-(0-\#c(F-F_2)+1)}n^2) = O(4^{k-j}n^2),$$

where we have used the fact $\#c(F-F_2) \leq |F-F_2| = k+1-j$. Now the proof proceeds exactly the same way as in Lemma B.2, and concludes that the W-FEEDBACK VERTEX SET REDUCTION problem can be solved in time $O(5^k n^2)$. □

Using Theorem B.3 and Lemma B.6, we obtain the main result of this section.

Theorem B.7 *The* PW-FEEDBACK VERTEX SET *problem on an n-vertex graph is solvable in time $O(5^k kn^2)$.*

PROOF. Let (G, k) be a given instance of the PW-FEEDBACK VERTEX SET problem. As we explained in the proof of Theorem B.3, we can first construct, in time $O(5^k kn^2)$, an FVS F of size $k + 1$ for the graph G (the weight of F is not necessarily the minimum). Then we simply apply Lemma B.6. □

Acknowledgment:

We thank Hans L. Bodlaender for the suggestion of combining iterative compression and an approximation algorithm, to further reduce the polynomial factor in the complexity of the algorithms.

C. Chapter Conclusion

In this chapter, we studied the PD-FEEDBACK VERTEX SET, P-FEEDBACK VERTEX SET, and PW-FEEDBACK VERTEX SET problems. We gave the fist algorithm of running time in form of $f(k)n^{O(1)}$ for the PD-FEEDBACK VERTEX SET problem, where $f(k)$ is independent of n. Thus we proved that the PD-FEEDBACK VERTEX SET problem is fixed parameter tractable and solved an outstand open problem, i.e. if

PD-FEEDBACK VERTEX SET is fixed parameter tractable. For the P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET problems, we improved the running time to $O^*(5^k)$ for both of them. The running time of our (deterministic) algorithm for the PW-FEEDBACK VERTEX SET problem is even better than that of the previous best (randomized) algorithm, which has a running time of $O^*(6^k)$ for the PW-FEEDBACK VERTEX SET problem.

Like using branch and bound method to solve the P-NODE MULTIWAY CUT problem, when we used branch and bound to solve the PD-FEEDBACK VERTEX SET, P-FEEDBACK VERTEX SET, and PW-FEEDBACK VERTEX SET problems, we had more than one direction to simplify instances and more than one condition to stop branching. For example, for the PD-FEEDBACK VERTEX SET problem, we first transformed it into the PD-SKEW SEPARATOR problem that is related to two parameters k and m. Then we used branch and bound to solve the PD-SKEW SEPARATOR problem. In each branch, we obtained two sub-instances such that for one sub-instance, k was reduced by 1 and m was reduced by at most 1 (m might keep the same value), where for another sub-instance, k kept the same value and m was increased by 1. The condition to stop branching was either $k = 0$ or $k < m$. Similarly, when solving the P-FEEDBACK VERTEX SET problem, we first transformed it into the P-F-BIPARTITION FEEDBACK VERTEX SET problem that is related to two parameters k and l. Then we used branch and bound to solve the P-F-BIPARTITION FEEDBACK VERTEX SET problem. In each branch, we obtained two sub-instances such that for one sub-instance, k was reduced by 1 and l kept the same value, where for another sub-instance, k kept the same value and l was reduced by 1. The condition to stop branching was either $k = 0$ or $l = 1$. In solving the PW-FEEDBACK VERTEX SET problem, the instance was even branched in three directions, i.e. values for three parameters were changed. Accordingly, we also have more conditions to stop branching.

Branch and conquer method is a very general way to design parameterized algorithms. Using this method more flexibly will lead us to solve more parameterized NP-hard problems.

CHAPTER VII

APPLICATIONS IN BIOINFORMATICS*

Bioinformatics is the application of information technology to the field of molecular biology. Usually, when we study bioinformatics problems, first we formulate biology problems into computational problems. Then we design algorithms for computational problems. Finally, we implement algorithms and test data.

As many computational problems from bioinformatics are NP-hard, it is very hard to solve those problems efficiently. For most NP-hard problems, as current algorithms are so ineffective, even we have the best supercomputer, we cannot find optimal solutions for those NP-hard problems in million years. Hence, traditionally, people try to avoid using NP-hard problems in applications. If they cannot avoid NP-hard problems, they usually use heuristic algorithms to solve them. Heuristic algorithms are fast, but performance is not guaranteed, i.e. we are not guaranteed to find optimal solutions.

We found that many bioinformatics problems are naturally related to parameters. they can be formulated as parameterized NP-hard problems. Furthermore, in applications, those parameters only take very small values. As we introduced before, if parameters only take small values, taking advantage of these small values to develop parameterized algorithms is a good way to find optimal solution for those parameterized NP-hard problems.

In this chapter, we will discuss our new results about using parameterized algorithms to solve the signaling pathway problem and the motif finding problem in

---

*Reprinted with permission from "Finding Pathway Structures in Protein Interaction Networks" by Songjian Lu, Fenghui Zhang, Jianer Chen, Sing-Hoi Sze, 2007. *Algorithmica*, 48, 363-374, Copyright 2007 by Springer.

bioinformatics.

A. Pathway Structure and Protein Interaction Networks

1. Introduction

By representing a biological network as a graph in which vertices represent genes or proteins and edges represent interactions between them, many algorithms have been proposed to study substructures in these graphs in order to understand the organization of interacting components within these networks [34, 64, 73, 77, 79, 96, 99, 110, 111, 114, 118]. The increased availability of these networks that describe interactions at a genome scale presents serious computational challenges since they can be very large, with thousands of vertices and tens of thousands of edges. In protein interaction networks, one important biological problem is to find chains of proteins that belong to a functional pathway, which corresponds to finding simple paths in the network with closely interacting proteins and is already a very difficult *NP*-hard problem [56]. In reality, biological pathways are not linear since there may be multiple interacting paths within a pathway. In order to understand these complex interactions, a more accurate model is to find a collection of closely related chains of proteins that form a pathway structure, which can be achieved by identifying a subgraph of the given network that includes the chains.

To address this problem, Steffen et al. [114] employed an exhaustive search procedure to find short simple paths in a given network that contain functional related proteins and combine top scoring paths into a path structure. Kelley et al. [73] developed a probabilistic algorithm that allows the identification of longer optimal simple paths in $k!n^{O(1)}$ time, where n is the number of vertices and k is the path length. Scott et al. [110] proposed an improved probabilistic algorithm for finding simple paths that

runs in $5.44^k n^{O(1)}$ time by using the color coding technique [3] and allowed the identification of more complicated substructures such as trees and series-parallel graphs. A few other approaches allow the identification of arbitrary biological substructures, but these algorithms either can only identify small substructures optimally or have to use heuristics to find large substructures. Koyutürk et al. [77] used a branch-and-bound procedure to identify small frequent subgraphs that occur in many networks. Hu et al. [64] employed a few graph-theoretic reductions to find dense subgraphs that occur in many networks. Sharan et al. [111] defined the notion of a network alignment graph and used a greedy heuristic to identify conserved subnetworks that occur in multiple species. Koyutürk et al. [79] defined the notion of local network alignment and used a greedy heuristic to identify conserved substructures in two graphs.

We are interested in developing algorithms with at least a probabilistically guaranteed performance that allows systematic investigation of pathway structures. A popular previous technique is to employ a two stage approach that first identifies high scoring simple paths and then combines these paths into a graph (that is a subgraph of the original network) to represent the path structure [73, 110, 114]. One drawback of this approach is that potential interactions between related paths are not taken into account. We investigate a different formulation that models a pathway structure directly as a leveled structure of proteins in which proteins from adjacent levels are connected together to represent a collection of closely related chains of proteins. We develop a divide-and-conquer algorithm to find an optimal path structure with high probability under a scoring scheme that models characteristics of biological pathways. Since the scoring scheme may not perfectly model biological reality, we also consider a variant of the algorithm that provides probabilistic guarantees for the top suboptimal path structures with a slight increase in time and space. We show that our algorithm can identify biological pathway structures by applying it to protein

interaction networks in the DIP database [120].

2. Problem formulation

Let k be the number of vertices in the target path structure and t be the maximum number of vertices at each level of the structure. We use the following notion of a (k, t)-path to represent a path structure.

Definition Given an undirected graph G, two disjoint sets of vertices S_1 and S_2 are said to be *connected* if each vertex in S_1 is adjacent to at least one vertex in S_2 and each vertex in S_2 is adjacent to at least one vertex in S_1. A (k, t)-*path* is a subgraph of G with k vertices that are partitioned into l disjoint subsets $S_1, S_2, \ldots, S_l$ such that (1) $|S_i| \leq t$ for $1 \leq i \leq l$, and (2) S_i is connected to S_{i+1} for $1 \leq i < l$. We denote the (k, t)-path as $[S_1, S_2, \ldots, S_l]$, and call S_1 and S_l the *ends* of the (k, t)-path (see Figure 17).

A (k, t)-path represents a path structure of size k with l levels in which each level is of size at most t, with the property that each vertex is included in a simple path of length l within the path structure. With this formulation, multiple interacting paths within a pathway can be modeled and proteins within the same level in a path structure are likely to play similar roles within the pathway. By imposing appropriate vertex or edge weights in G, it becomes possible to investigate these interacting paths in biological networks directly. Although the formulation only considers edges between adjacent levels in the structure, no restrictions are imposed on edges that can exist between non-adjacent levels. Furthermore, complicated path structures can be represented even when t is small (e.g., $t \leq 2$ or 3). To study paths between two vertices in G, one can impose a source and a sink in the path structure (i.e., both

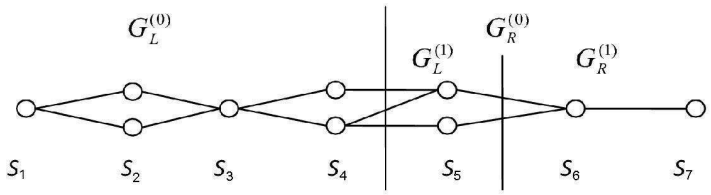


Fig. 17. Illustration of the definition of a (k,t)-path and the probabilistic divide-and–conquer algorithm. The parameters are $k = 10$, $t = 2$ and $l = 7$. S_1 and S_7 are the ends of the (k,t)-path. Only edges between adjacent levels are shown (there may be many other edges in G that connect vertices from non-adjacent levels). One iteration of step 3 of the algorithm in Figure 18 is shown in which G is randomly partitioned into two parts $G_L^{(0)}$ and $G_R^{(0)}$ that also subdivides the (k,t)-path into two smaller path structures of roughly equal sizes. The recursion within $G_L^{(0)}$ returns a set of path structures, in which $[S_1, S_2, S_3, S_4]$ is among one of them. $G_R^{(0)}$ is further partitioned into $G_L^{(1)}$ and $G_R^{(1)}$. The path structure $[S_1, S_2, S_3, S_4]$ is then concatenated to $[S_5]$ before $[S_6, S_7]$ is finally added

the first and the last levels have only one vertex). Since vertices in different levels are distinct, a (k,t)-path then represents multiple simple paths of length l between the source and the sink. Note that in this formulation, the number of levels l is not a parameter and the best path length l (with respect to a given (k,t) pair) will be found automatically during the computation. Since it is sometimes necessary to find paths of a specific length l that form a path structure, an alternative strategy is to consider l as a parameter. We will show that a variant of our algorithm can be used to find multiple (k,t)-paths for each value of l simultaneously without a significant increase in time and space.

Our formulation is different from the series-parallel graph formulation in Scott et al. [110] which defines a different type of non-linear structure. Our representation of the path structure is also different from a path-decomposition of bounded

pathwidth [10] which allows vertex groupings that may overlap and has additional constraints on edges within a vertex group. Since very extensive overlaps can occur within a vertex group in a path-decomposition, much more complicated structures than multiple simple paths are allowed, which may not be biologically useful since meaningless paths with very high scores that contain many repeated vertices can be included.

Even for the case $t = 1$, the problem is difficult since it corresponds to the *NP*-hard problem of finding simple paths of length k [56]. Despite significant efforts, there is no practical deterministic approach that guarantees that the optimal path is found unless the path length is very short [114]. The best previous deterministic algorithm uses the color coding technique [3], which, when coupled with the dynamic programming technique in Scott et al. [110], gives an algorithm that runs in $d^k n^{O(1)}$ time, where d is an impractically large constant. The best previous probabilistic algorithm combines the probabilistic version of the color coding technique in Alon et al. [3] with the dynamic programming technique in Scott et al. [110] to give an approach that guarantees to find the optimal path with high probability and runs in $(2e)^k n^{O(1)} = 5.44^k n^{O(1)}$ time and $O(nk2^k + m)$ space, which are both exponential in k. Kelley et al. [73] gave the best previous probabilistic algorithm that uses polynomial space by imposing acyclic edge orientations and runs in $k! n^{O(1)}$ time and $O(m + n)$ space. For the special case $t = 1$, our probabilistic algorithm that runs in $4^k n^{O(1)}$ time and $O(nk \log k + m)$ space offers a significant improvement over these probabilistic approaches, with a much lower constant in the exponential part of the time complexity while requiring only polynomial space.

3. Probabilistic algorithm

For simplicity, we first consider finding an arbitrary (k,t)-path in G. Suppose that such a (k,t)-path exists and t is small. The idea is to use a divide-and-conquer approach to randomly partition G into two subgraphs. Consider this partition a success if it subdivides a (hidden) (k,t)-path into two smaller path structures of roughly equal sizes, which results in two subproblems of finding (k',t)-paths with k' roughly equal to $k/2$ that can connect together to form a (k,t)-path. Continue to recursively partition into smaller subgraphs until the base case $k \leq t$ is reached, in which the (k,t)-paths can be enumerated exhaustively as follows.

Lemma A.1 *Let G be an undirected graph with n vertices and let $k \leq t$. There are at most $(2n)^k$ (k,t)-paths in G and these (k,t)-paths can be constructed in $O(k(2n)^k) = O(t(2n)^t)$ time.*

Proof Let $S = \{v_1, \ldots, v_k\}$ be any set of k vertices in G. We first consider how many (k,t)-paths can be obtained from these vertices in S. Let π be a permutation of S. We insert "separators" between the vertices in π to partition π into non-empty segments (we require that at most one separator be inserted between two vertices in π). We call this a *partitioned permutation* of S. Any (k,t)-path formed from S can be represented by a partitioned permutation of S, in which segments correspond to levels in the (k,t)-path. Although each (k,t)-path formed from S may be represented by more than one partitioned permutation of S, each partitioned permutation of S can represent at most one valid (k,t)-path. Thus the number of partitioned permutations of S gives an upper bound on the number of (k,t)-paths that can be obtained from S. Fix a permutation π and let $T(k)$ be the total number of partitioned permutations that can be formed from π. Since for each i, $0 < i \leq k$, there are $T(k-i)$ partitioned permutations in which the first segment consists of the first i vertices in π, we have

the following recurrence

$$T(k) = T(k-1) + T(k-2) + \cdots + T(0),$$

which gives $T(k) \leq 2^k$ (since $T(0) = 1$). Since there are $k!$ permutations of the vertices in S, there are at most $2^k k!$ partitioned permutations of the vertices of S. Thus at most $2^k k!$ (k,t)-paths can be obtained from S. Since there are $\binom{n}{k} \leq n^k/k!$ sets of k vertices in G, we conclude that the total number of (k,t)-paths in G is bounded by $(n^k/k!)(2^k k!) = (2n)^k$. These (k,t)-paths can be easily constructed in $O(k(2n)^k) = O(t(2n)^t)$ time by the exhaustive enumeration method described above. □

To allow the merging of two disjoint smaller path structures p and p' to form a larger path structure, one end of p must be connected to an end of p'. We introduce the following notations. Let S and S' be two sets of vertices of size at most t in G. An (S,k,t)-path is a (k,t)-path in which one end is S. Similarly, an (S,S',k,t)-path is a (k,t)-path in which one end is S and the other end is S'. Our algorithm recursively constructs a set P_0 of (k',t)-paths for a subgraph G_0, where for each set S of size at most t in G_0, at most one (S,k',t)-path is stored in P_0. Suppose that we have constructed such a set P_0 of (k',t)-paths for a subgraph G_0 and let G be a graph that shares no vertices with G_0. The algorithm find-paths(G,P_0,k,t) returns a set P of $(k+k',t)$-paths in the induced subgraph of G formed by the vertices in G_0 and G, with at most one $(S,k+k',t)$-path in P for each vertex set S of size at most t in G, and each $(S,k+k',t)$-path is a concatenation of a (k',t)-path in P_0 and an (S,k,t)-path in G. The initial call to the algorithm is find-paths$(G,\emptyset,k,t)$, which returns a set of (k,t)-paths in G. Figure 18 illustrates the details of the algorithm, where we assume that no path structure in P_0 contains a vertex in G (see also Figure 17). Step 2 of

the algorithm represents the base case in which exhaustive enumeration is possible and step 3 represents the recursive partition step.

Algorithm find-paths(G, P_0, k, t)
{Assume that no path structure in P_0 contains a vertex in G}

```
1.   P = ∅;
     if k ≤ t then
2.     for each (S, S', k, t)-path p in G do
         if P_0 = ∅ then
2.1.       add p to P if no (S', k, t)-path is in P;
         else
2.2.       for each (S'', k', t)-path p' in P_0 do
             if S'' is connected to S then
2.3.           concatenate p and p' into an (S', k + k', t)-path p'';
2.4.           add p'' to P if no (S', k + k', t)-path is in P;
     else
3.     loop 2.51 · 2^k times do
3.1.     randomly partition the vertices in G into two parts V_L and V_R;
3.2.     let G_L and G_R be the subgraphs induced by V_L and V_R respectively;
3.3.     for i = k − ⌊(k + t)/2⌋ to ⌊(k + t)/2⌋ do
3.4.       P_L = find-paths(G_L, P_0, i, t);
           if P_L ≠ ∅ then
3.5.         P_R = find-paths(G_R, P_L, k − i, t);
3.6.         for each (S, k + k', t)-path p in P_R do
3.7.           add p to P if no (S, k + k', t)-path is in P;
4.   return P;
```

Fig. 18. Illustration of the find-paths algorithm

Theorem A.2 *Let G be an undirected graph with n vertices and m edges and let S be a set of vertices of size at most t in G. If G contains an (S, k, t)-path, then with probability larger than $1-1/e > 0.632$, the set P returned by find-paths$(G, \emptyset, k, t)$ contains an (S, k, t)-path. The algorithm find-paths$(G, \emptyset, k, t)$ runs in $O(4^k n^{2t} k^{t+\log(t+1)+2.92} t^2)$ time and $O(n^t k \log k + m)$ space.*

Proof We prove the following claims by induction on k.

1. If $P_0 = \emptyset$ and G has an (S, k, t)-path, then with probability larger than $1 - 1/e$, the set P returned by find-paths(G, P_0, k, t) contains an (S, k, t)-path.

2. If P_0 is a non-empty set of (k', t)-paths, and G has an (S, k, t)-path such that its other end is connected to a (k', t)-path in P_0, then with probability larger than $1 - 1/e$, the set P returned by find-paths(G, P_0, k, t) contains an $(S, k + k', t)$-path that is a concatenation of an (S, k, t)-path in G and a (k', t)-path in P_0.

The claims are obviously true when $k \leq t$ since all (k, t)-paths in G are exhaustively enumerated. We let $k > t$ and first consider the case when $P_0 = \emptyset$. Suppose that $[S_1, S_2, \ldots, S_{k_1}, S_{k_1+1}, \ldots, S_l]$ is an (S, k, t)-path in G, where each S_i is a level in the (S, k, t)-path of size at most t, $S_l = S$, $\sum_{i=1}^{k_1} |S_i| = d_1 \leq (k + t)/2$ and $\sum_{i=k_1+1}^{l} |S_i| = d_2 \leq (k+t)/2$. Such a choice of k_1 is always possible since $|S_i| \leq t$ for all i. With probability $1/2^k$, step 3.1 puts the vertices in $S_1, S_2, \ldots, S_{k_1}$ into V_L and the vertices in $S_{k_1+1}, \ldots, S_l$ into V_R. In this case, G_L contains the (S_{k_1}, d_1, t)-path $[S_1, \ldots, S_{k_1}]$, and G_R contains the $(S, k - d_1, t)$-path $[S_{k_1+1}, \ldots, S_l]$. By the inductive hypothesis, with probability larger than $1 - 1/e$, P_L from step 3.4 contains an (S_{k_1}, d_1, t)-path. When this is the case, the $(S, k-d_1, t)$-path $[S_{k_1+1}, \ldots, S_l]$ in G_R has its other end S_{k_1+1} connected to the (S_{k_1}, d_1, t)-path in P_L. By the inductive hypothesis, with probability larger than $1 - 1/e$, P_R from step 3.5 contains an (S, k, t)-path. Thus the probability ρ that an (S, k, t)-path is added to P in step 3 is larger than

$$\frac{(1 - 1/e)^2}{2^k} > \frac{0.632^2}{2^k} > \frac{1}{2.51 \cdot 2^k}.$$

When $P_0 \neq \emptyset$, we follow the same argument as before except that we require that the (S, k, t)-path in G has its other end connected to a (k', t)-path in P_0, P_L contains an $(S_{k_1}, d_1 + k', t)$-path that is a concatenation of an (S_{k_1}, d_1, t)-path in G_L and a (k', t)-path in P_0, and P_R contains an $(S, k + k', t)$-path that is a concatenation

of an $(S, k-d_1, t)$-path in G_R and a (d_1+k', t)-path in P_L.

Since step 3 loops $2.51 \cdot 2^k$ times, the overall probability that the set P returned by find-paths(G, P_0, k, t) contains an (S, k, t)-path when P_0 is empty or an $(S, k+k', t)$-path when P_0 is not empty is

$$1-(1-\rho)^{2.51\cdot 2^k} > 1-\left(1-\frac{1}{2.51\cdot 2^k}\right)^{2.51\cdot 2^k} > 1-\frac{1}{e}.$$

This completes the first part of the proof.

We now analyze the complexity of the algorithm. Let $T(k)$ be the time complexity of find-paths(G, P_0, k, t). When $k \le t$, by Lemma A.1, there are at most $(2n)^t$ (k, t)-paths in G, and these (k, t)-paths can be constructed in $O(t(2n)^t)$ time. Since P_0 contains at most one (S, k', t)-path for each set S of size at most t, the number of path structures in P_0 is at most $\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t} \le n^t$. For each (S'', k', t)-path in P_0 and each (S, S', k, t)-path in G, since both S'' and S are of size at most t, it takes $O(t^2)$ time to check if S'' and S are connected. Thus we have $T(k) = O(t(2n)^t + (2n)^t n^t t^2) = O(n^{2t} 2^t t^2)$ when $k \le t$.

When $k > t$, since there are at most n^t path structures in P_R, steps 3.6-3.7 take $O(kn^t)$ time. We have the following recurrence

$$T(k) = 2.51\cdot 2^k \sum_{i=k-\lfloor (k+t)/2 \rfloor}^{\lfloor (k+t)/2 \rfloor} \big(T(i)+T(k-i)+ckn^t\big) \le 2.51(t+1)2^k\big(2T(\lfloor (k+t)/2 \rfloor)+ckn^t\big),$$

where c is a constant. We can assume that $ckn^t \le T(\lfloor (k+t)/2 \rfloor)$. Thus

$$T(k) \le 7.53(t+1)2^k T(\lfloor (k+t)/2 \rfloor) \le 7.53(t+1)2^k T(k/2+t/2).$$

To solve this, note that a general form derived from the recurrence is

$$T(k) \le (7.53(t+1))^i 2^{g(k,t,i)} T(k/2^i + t/2^i + t/2^{i-1} + \cdots + t/2),$$

where

$$g(k,t,i) = k+(k/2+t/2)+(k/2^2+t/2^2+t/2)+\cdots+(k/2^{i-1}+t/2^{i-1}+t/2^{i-2}+\cdots+t/2).$$

Let $i = \log k$ and note that $T(k) = O(n^{2t}2^t t^2)$ when $k \leq t$, we have

$$T(k) = O((7.53(t+1))^{\log k} 2^{2k+t(\log k-1)} n^{2t}2^t t^2) = O(4^k n^{2t} k^{t+\log(t+1)+2.92} t^2).$$

We now analyze the space complexity of the algorithm. Since each recursive call of find-paths uses $O(n^t k)$ space (mainly for the sets P_L, P_R, and P) and the depth of recursion is $\log k$, find-paths$(G, \emptyset, k, t)$ uses $O(n^t k \log k + m)$ space. □

The algorithm is thus practical when k is moderately large and t is small, with 4^k being the dominating term in the time complexity. Note that as t becomes larger relative to k, the running time increases to compensate for the more unbalanced subdivisions of the target path structure into two smaller path structures of different sizes in step 3.3. To achieve an arbitrarily small error bound ϵ, we can run the algorithm r times so that r satisfies $1/e^r < \epsilon$ (e.g., for $\epsilon = 0.001$, we have $r = 7$). To allow (k,t)-paths for each value of l to be found independently, instead of storing at most one (S,k,t)-path in steps 2.4 and 3.7, at most one (S,k,t)-path is stored for each value of l. This increases the time and space complexity by at most a factor of k.

4. Optimization for $\boldsymbol{t \leq 2}$

Since protein interaction graphs are often sparse, we replace steps 2 and 2.2–2.4 ($k \leq t$ and $P_0 \neq \emptyset$) of our algorithm by the following procedure to lower the running time when $t = 1$: concatenate each $(\{v\}, k', t)$-path in P_0 (if it exists) and each edge (v,w) in G to obtain a $(\{w\}, k'+1, t)$-path, while storing at most one $(\{w\}, k'+1, t)$-path

for each w. When $t = 2$, a slightly more complicated procedure is used: for each pair of edges (v_1, w_1) and (v_2, w_2) in G such that v_1 and v_2 appear among the path structures in P_0 and w_1 and w_2 are in G (v_1 and v_2, w_1 and w_2 are not necessarily distinct), concatenate the $(\{v_1, v_2\}, k', t)$-path in P_0 (if it exists) with $\{w_1, w_2\}$ to obtain a $(\{w_1\}, k' + 1, t)$-path if $w_1 = w_2$ or a $(\{w_1, w_2\}, k' + 2, t)$-path if $w_1 \neq w_2$, while storing at most one path structure for each $\{w_1, w_2\}$. This procedure gives $(S, k' + 1, t)$-paths for all S of size 1 and $(S, k' + 2, t)$-paths for all S of size 2, with at most one path structure for each S. Since $k \leq t = 2$, the other $(S, k' + 2, t)$-paths for S of size 1 can be obtained from the $(k' + 1, t)$-paths by starting from each vertex v in G and search among all its adjacent edges (v, w) to find a $(\{w\}, k' + 1, t)$-path to concatenate into a $(\{v\}, k' + 2, t)$-path. The above procedures for $t \leq 2$ take $O(m^t)$ time, which are much better than the original $O(n^{2t})$ time for step 2 when G is sparse and also lead to the same improvement for the entire algorithm. Note that since we have $m = O(n^2)$ in the worst case, the worst case time complexity of the algorithm stays the same, but it is much faster in practice for interaction graphs. Also note that this technique is applicable only when $t \leq 2$, since it may take more than t edges in G to fully specify a connection between two sets of vertices of size t when $t \geq 3$.

5. Scoring path structures

In reality, for given k and t, there may be many (k, t)-paths in a biological network. We need to assign a score to each (k, t)-path so that biologically relevant structures are likely to get better scores. Since we are interested in identifying paths between two proteins in G, we assume that an additional source and sink are given in addition to G. Since the proteins that are related to the source and the sink in biological function are more likely to belong to the same pathway, we assign a weight to each vertex in G to reflect its functional relatedness to the source and the sink. For yeast

protein interaction networks, we estimate the functional relatedness of proteins by using a compendium set of expression profiles corresponding to 300 mutations and chemical treatments from Hughes et al. [65]. We define the functional similarity $s(p_1, p_2)$ between two proteins p_1 and p_2 to be the Pearson correlation coefficient ρ of the log(expression ratio) across experiments, which is given by

$$\rho = \left(\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})\right) \Bigg/ \sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2},$$

where n is the number of experiments, x_i and y_i are the log(expression ratio) of proteins p_1 and p_2 respectively in the ith experiment, and $\overline{x}$ and $\overline{y}$ are the means of x_i and y_i respectively. The weight of a vertex v in G is defined to be $s(v, \text{source}) + s(v, \text{sink})$, with a higher value representing better functional relatedness to the source and the sink. We then define the score of a (k, t)-path to be the sum of the weight of its k vertices. This allows different (k, t)-paths to be compared directly for a fixed (k, t) pair.

To accommodate the source and the sink, we make the following changes to our algorithm. In step 2.1 ($k \leq t$ and $P_0 = \emptyset$), only return (S, k, t)-paths with the other end connected to the source. After the algorithm is completed, only consider the best (S, k, t)-paths with S connected to the sink. Instead of storing an arbitrary (S, k, t)-path, we remember one currently best (S, k, t)-path with the highest score in steps 2.1, 2.4 and 3.7. Since only vertex weights are used, merging two disjoint optimal path structures always gives a larger optimal path structure. Thus the optimal substructure property is satisfied for our divide-and-conquer technique and the probabilistic guarantee in Theorem A.2 is valid for optimal (S, k, t)-paths.

One problem of the current algorithm is that only one (S, k, t)-path is stored for each fixed S within each iteration, and thus not many distinct path structures may

be obtained from one run. Since many relevant biological path structures may have suboptimal scores, it would be desirable to return as many distinct path structures as possible in one run. Without significantly increasing the time and space complexity, we modify our algorithm to store top x suboptimal (S, k, t)-paths for each S in steps 2.4 and 3.7. A similar argument shows that a probabilistically guaranteed performance can be obtained for each of the top x suboptimal path structures and the time complexity increases by at most a factor of x^2. To verify that these path structures are sufficiently different from random path structures, we assume that the weights of each vertex are normally distributed and check that the score of a (k, t)-path is at least a few standard deviations away from the mean score of random (k, t)-paths in which k vertices are chosen independently.

6. Application to protein interaction networks

We test our algorithm on the core protein interaction network of yeast [35] from the DIP database [120], which has 3170 vertices and 6600 edges. Figures 19 and 20 show top scoring path structures on a few pairs of source and sink that correspond to the endpoints of four mitogen-activated protein kinase (MAPK) cascades in Gustin et al. [61], including the pheromone response pathway, the filamentation/invasion pathway, the cell integrity pathway and the high osmolarity pathway, in which detailed biological models are available. The main characteristics of these pathways are that each of them contains a MAPKK kinase, a MAPK kinase and a MAP kinase in series [61]. To further validate our results, we compare our results to the computational results in Steffen et al. [114] and Scott et al. [110]. With $t = 2$, it takes minutes to obtain the results for $k = 10$ from one run of the algorithm, hours for $k = 11$, and many hours to a day for $k = 12$. It takes only slightly more time to guarantee a high probability of finding more than one top suboptimal path structure when compared

to finding only the optimal path structure.

In general, a larger value of k results in more interacting proteins found within a path structure and also provides more information. Note that in order to investigate multiple paths, all the results in Steffen et al. [114] and some of the results in Scott et al. [110] are aggregates of many linear paths. Since our path structure allows multiple paths in its definition, we can investigate each path structure separately. Due to the relatively small values of k we used, each of our path structure generally contains a smaller number of proteins than their aggregate results. However, our algorithm makes it easier to identify related proteins at the same level of the path structure that may indicate that they play similar roles than looking at aggregates of simple paths.

Figure 19 shows the top path structures obtained between Ste3 and Ste12 that correspond to the pheromone response pathway and between Ras2 and Ste12 that correspond to the filamentation/invasion pathway. On the test between Ste3 and Ste12 for the pheromone response pathway, most proteins in the main chain in Gustin et al. [61], including Ste3, Ste18, Ste4, Ste5, Ste7, Fus3 and Ste12, were found. A few other proteins that are present in the biological model in Gustin et al. [61], including Far1, Bem1, Cdc42 and Gpa1, were also found. The proteins Akr1 and Kss1 are also present in the results in Steffen et al. [114] and Scott et al. [110], while the protein Sst2 is also present in the results in Steffen et al. [114]. Akr1 and Sst2 are both related to the pathway, while Kss1 is related to the almost identical filamentation/invasion pathway that replaces Fus3 by Kss1 [61]. On the test between Ras2 and Ste12 for the filamentation/invasion pathway, our algorithm identified many of the same proteins as above due to the strong resemblance of the two pathways. The proteins Cdc25 and Hsp82 are also present in the results in Steffen et al. [114] and Scott et al. [110], while the proteins Cyr1 and Srv2 are also present in the results in Steffen et al. [114]. Some

Between Ste3 and Ste12: (pheromone response)

$k = 10$: Ste3–Akr1–Ste18–Ste4–Ste5–Fus3–Gpa1–Sst2–Kss1–Ste12

Ste3–Akr1–(Ste18,Ste5)–(Ste4,Fus3)–Gpa1–Sst2–Kss1–Ste12

$k = 11$: Ste3–Akr1–Ste4–Far1–Bem1–Ste5–Fus3–Gpa1–Sst2–Kss1–Ste12

Ste3–Akr1–Ste18–Ste4–Ste5–Ste7–Fus3–Gpa1–Sst2–Kss1–Ste12

$k = 12$: Ste3–Akr1–Ste18–Ste4–Far1–Bem1–Ste5–Fus3–Gpa1–Sst2–Kss1–Ste12

Ste3–Akr1–Ste4–Far1–Cdc42–Bem1–Ste5–Fus3–Gpa1–Sst2–Kss1–Ste12

Between Ras2 and Ste12: (filamentation/invasion)

$k = 10$: Ras2–Ras1–Cdc25–Ssa2–Gpa1–(Ste4,Fus3)–Ste5–Kss1–Ste12

Ras2–Cdc25–Ssa2–Gpa1–Ste4–Ste5–Fus3–Ste7–Kss1–Ste12

$k = 11$: Ras2–Cdc25–Ssa2–Gpa1–Ste4–Ste5–Fus3–Mpt5–Sst2–Kss1–Ste12

Ras2–Cdc25–Hsp82–Skp1–Cln1–Far1–Ste4–Gpa1–Sst2–Kss1–Ste12

$k = 12$: Ras2–Cyr1–Dbf2–Sec27–Prp11–Ssk2–Far1–Ste4–Gpa1–Sst2–Kss1–Ste12

Ras2–Cyr1–Srv2–Abp1–Cla4–Cdc42–Far1–Ste4–Gpa1–Sst2–Kss1–Ste12

Fig. 19. Top two path structures from the find-paths algorithm on the core protein interaction network of yeast between two pairs of source and sink with k from 10 to 12 and $t = 2$. Two path structures with the same end are stored within the algorithm and the algorithm is run seven times to guarantee a 99.9% probability of finding each of the top two path structures. Proteins within the same level in a path structure are enclosed in parentheses while adjacent levels are connected by a horizontal line

Between Rho1 and Rlm1: (cell integrity)	Between Sln1 and Hog1: (high osmolarity)
Rho1–Rga1–Cdc28–Swi6–Slt2–Rlm1	Sln1–Rvs167–Abp1–Tup1–Sko1–Hog1
Rho1–Rga1–Cdc28–Spa2–Slt2–Rlm1	Sln1–Ypd1–Ssk1–Ssk2–Pbs2–Hog1
Rho1–Rga1–Cdc28–Kin2–Slt2–Rlm1	Sln1–Rvs167–Rxt1–Cyc8–Sko1–Hog1
Rho1–Bni1–Myo3–Bck1–Slt2–Rlm1	Sln1–Rvs167–Idh1–Slt2–Ptp2–Hog1
Rho1–Bni1–Bck1–Mkk2–Slt2–Rlm1	Sln1–Rvs167–Lys12–Slt2–Ptp2–Hog1
Rho1–Pkc1–Mkk1–Bck1–Slt2–Rlm1	Sln1–Rvs167–Idh1–Slt2–Ptp3–Hog1

Fig. 20. Top six path structures from the find-paths algorithm on the core protein interaction network of yeast between two pairs of source and sink with $k = 6$ and $t = 2$. Six path structures with the same end are stored within the algorithm and the algorithm is run seven times to guarantee a 99.9% probability of finding each of the top six path structures

number of other proteins were also included that may not be related to the pathway.

Figure 20 shows the top path structures obtained between Rho1 and Rlm1 that correspond to the cell integrity pathway and between Sln1 and Hog1 that correspond to the high osmolarity pathway. Since these pathways are short [61], we set $k = 6$ (we were not able to obtain good results for larger k). On the test between Rho1 and Rlm1 for the cell integrity pathway, the sixth suboptimal path structure that includes the proteins Rho1, Pkc1, Mkk1, Bck1, Slt2 and Rlm1 had the best agreement with the biological model in Gustin et al. [61], which contains all the components in the main chain. The fifth suboptimal path structure that includes the proteins Rho1, Bni1, Bck1, Mkk2, Slt2 and Rlm1 also agreed extremely well, which includes the protein Bni1 that is related to the pathway. Steffen et al. [114] used a different path length seven to find the pathway, while Scott et al. [110] missed the MAPKK kinase Bck1. On the test between Sln1 and Hog1 for the high osmolarity pathway, the second

suboptimal path structure that includes the proteins Sln1, Ypd1, Ssk1, Ssk2, Pbs2 and Hog1 had the best agreement with the biological model in Gustin et al. [61], which contains all the components in the main chain. Steffen et al. [114] was unable to find the pathway, while Scott et al. [110] only found the pathway among lower ranked suboptimal paths.

7. Discussion

We have developed a new formulation that allows the direct modeling of biological pathways by a non-linear path structure and gave an algorithm that guarantees that the top suboptimal path structures are found with high probability. Our approach can be easily adapted to analyze directed graphs and it can be applied to other types of biological networks such as metabolic networks. It is also possible to apply the algorithm to analyze conserved path structures in multiple graphs, by following a similar idea as in Kelley et al. [73], Sharan et al. [111] and Koyutürk et al. [79] to first construct a combined graph from the given graphs to represent desirable vertex correspondences, and then identify high scoring path structures in the combined graph to obtain alignments of the conserved structures.

One limitation of our current model is that all paths within a path structure must be of the same length l. Although other edges can exist outside the path structure that allow some of the paths to have different lengths and different path structures can have different path lengths, it would be desirable to extend the model to allow multiple paths of different lengths to be included in a path structure directly. Another difficulty is that since the algorithm still has exponential time complexity, it can only be used when k and t are not large. Although our algorithm is asymptotically faster than the previous approaches, for these relatively small values of k and t, we did not observe significant improvements in actual running time when compared to the

previous approaches and most of the path structures found were almost linear.

Since the false positive rate of edges in protein interaction networks can be quite high, it may be desirable to also take edge reliability into account [35, 39, 110, 114]. One way to do this is to incorporate edge weights in addition to vertex weights. Although no modifications to the algorithm are necessary since the optimal substructure property is still satisfied, one has to make sure that it is still meaningful to directly compare the scores of two path structures that have different number of edges when both vertex and edge weights are used.

We finished the discussion of the signaling pathway problem. Let us go to the next section and discuss the motif finding problem.

B. Motif Finding and DNA Sequences

1. Introduction

The motif finding problem is among the most well-studied problems in computational biology, with important applications in the computational identification of regulatory sites given a set of genes believed to be co-regulated. Most early motif finding approaches used conventional statistical optimization techniques to identify the most over-represented patterns in a sample of sequences (Stormo and Hartzell [116]; Lukashin et al. [88]; Lawrence et al. [81]; Bailey and Elkan [8]). Pevzner and Sze ([98]) proposed a combinatorial graph-theoretic formulation and suggested algorithms WINNOWER and SP-STAR demonstrating that, at least for artificial samples with uniform background distribution, these previous approaches have not reached the limit of prediction yet: they are not able to find implanted patterns in the cases when these patterns should be found. More recently, a steady stream of combinatorial approaches were proposed which investigate the theoretical prediction limit of motif

finding algorithms and further improve both prediction performance and computational speed (Marsan and Sagot [89]; Buhler and Tompa [17]; Eskin and Pevzner [47]; Keich and Pevzner [72]; Price et al. [100]). Instead of looking for the motif instances directly as in early sample-driven approaches, most of these later approaches assume the existence of a single string which is close to each motif instance and try to locate this string directly by a pattern-driven technique. These approaches fall roughly into two categories: either a fast heuristic is used which does not guarantee that the best solution is found, or an exact algorithm is employed which is often quite slow in practice.

In this section, we propose an integrated approach utilizing both sample-driven and pattern-driven techniques which allows us to develop a fast exact approach for moderate size problems. We employ the clique formulation from Pevzner and Sze ([98]) to represent each candidate motif as a clique containing a set of motif instances as in the sample-driven approach, while further requiring the existence of a string that is close to every motif instance in the clique as in the pattern-driven approach. This reduces the motif finding problem to finding large cliques which satisfy the close string constraint. Computational experiments show that the new technique is able to solve some of the most difficult motif finding problems which were unsolvable before using the clique formulation alone.

One possible approach to solve the problem is to employ a branch-and-bound technique to repeatedly expand a growing clique (Bomze et al. [15]), while pruning those branches which will not possibly lead to a clique that satisfies the constraint. However, a recent result (Chen et al. [23]) showed that unless unlikely consequences occur in parameterized complexity theory, the clique finding problem (and many other NP-hard problems) cannot be solved in $n^{o(k)}$ time, where n is the size of the graph (number of vertices) and k is the size of the maximum clique. Thus this kind of

clique-based approach is not likely to be efficient enough for large samples if we do not take advantage of special structures inherent in the given graphs.

The key observation is that we can restrict our attention to k-partite graphs, which are graphs consisting of k parts with no edges between vertices within the same part. Although the clique finding problem in k-partite graphs is still an intrinsically difficult computational problem, we will show that a divide-and-conquer approach can be used to circumvent the difficulties in the cases when every sequence contains a motif instance, by subdividing the original graph into smaller subgraphs, using a branch-and-bound algorithm to solve each subproblem independently, and combining the results. To cope with imperfect biological samples, we will demonstrate how to generalize this approach to handle the cases when almost all sequences contain a motif instance. When many sequences do not contain a motif instance, we will show an improved branch-and-bound approach to handle realistically sized samples. When there can be more than one motif instance in a sequence, we will show how to reduce the more general problem of having at most p instances per sequence, for a small p, to the original problem so that we still have an efficient algorithm.

One very important requirement of the proposed algorithm is to be able to quickly decide if a close string exists when given a clique containing a set of strings of the same length. Unfortunately, this problem has been proven to be NP-hard (Lanctot et al. [80]). Although branch-and-bound techniques for this problem have been proposed (Gramm et al. [58]), they are too slow for our purpose if used directly. One way to solve this problem is not to check for the existence of a close string every time when we expand a clique, but check only when a clique potentially larger than before is found. To reduce the number of intermediate cliques, weaker necessary conditions are used instead during every clique expansion to prune impossible branches.

A further concern is that while this model is adequate in many cases, it is not

a sufficient model when some position within the motif has more than one dominant letter, since we are forced to choose only one letter in each position of the close string. We will discuss various possibilities to address this problem. Recently, Price et al. ([100]) and Eskin ([46]) tried to address this problem by using a profile instead of a single string to represent a motif.

2. Problem formulation

We use the combinatorial (l, d)-motif model proposed in Pevzner and Sze ([98]): start from a set of random sequences $\{s_1, \ldots, s_k\}$, each of a fixed length, implant an (l, d)-motif by fixing a pattern P of length l and putting a randomly mutated pattern with exactly d substitutions from P at a randomly chosen position in each sequence. Without knowing P and where the implantations are, the motif finding problem is to recover the locations of the implanted patterns. Notice that making exactly d substitutions gives a more difficult model than another alternative of making at most d substitutions since distances between the motif instances will be greater.

A graph-theoretic approach was proposed in Pevzner and Sze ([98]): for each position j in sequence s_i, construct a vertex s_{ij} representing the substring of length l starting at position j in s_i. Connect two vertices s_{ij} and s_{pq} by an edge if $i \neq p$ and the distance (number of substitutions) between them does not exceed $2d$. In this formulation, an (l, d)-motif is modeled by a clique of size k. In light of substantial computational difficulty, Pevzner and Sze ([98]) developed the WINNOWER algorithm to find large clique-like structures instead of cliques. Liang ([82]) made a few refinements to WINNOWER to make it more sensitive.

The typical size of a motif finding problem consists of 10 to 50 sequences with sequence lengths ranging from a few hundred to about 2000, and thus there can be from a few thousand to tens of thousands of vertices in the constructed graph. Despite

the large graph size, an important observation is that the size of the maximum clique is small. Since edges are constructed only between vertices in different sequences in the (l, d)-motif model, the resulting graph is k-partite, with all the vertices of a sequence residing in the same part. This restricts the size of the maximum clique to be at most k. In fact, in the (l, d)-motif model, each sequence contains exactly one motif instance and the maximum clique size is k. We will take advantage of these two facts to develop a divide-and-conquer approach for the problem. For simplicity of analysis, we assume that the length of each sequence is the same, and thus, the number of vertices in each part of the graph is the same. We formulate the unconstrained clique finding problem as follows.

> CLIQUE$_{\boldsymbol{k}\mathbf{P}}$: Given a $\boldsymbol{k}$-partite graph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices in each part, find a maximum clique in $\boldsymbol{G}$.

Unfortunately, the problem is still hard to solve.

Proposition B.1 *CLIQUE$_{k\mathrm{P}}$ is NP-hard.*

We introduce the close string constraint as follows.

> CLIQUE$_{\boldsymbol{k}\mathbf{P}}(\boldsymbol{l}, \boldsymbol{d})$: Given a $\boldsymbol{k}$-partite graph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices in each part, where each vertex represents a string of length $\boldsymbol{l}$ and an edge is connected between two vertices if they are in different parts and their distance (number of substitutions) is at most $\mathbf{2}\boldsymbol{d}$, find a maximum clique in $\boldsymbol{G}$ such that there exists a string $\boldsymbol{s}$ of length $\boldsymbol{l}$ with distance at most $\boldsymbol{d}$ to every string in the clique.

Computational experiments show that with this constrained formulation, we are able to solve very difficult motif finding problems, such as samples containing 20

sequences each of length 600 with (16,5)- or (18,6)-motifs, which were unsolvable before due to an exceedingly large number of unconstrained maximum cliques ($> 10^6$) that do not represent the implanted motif (these results were obtained by running our clique finding program without the close string constraint). In fact, if we think of a solution as the close string s instead of the corresponding clique, this constrained formulation has exactly the same solutions as a pure pattern-driven approach. The major advantage of including cliques in the formulation is that it allows the use of a fast divide-and-conquer approach when the size of the maximum clique is very close to k.

With the above formulation, there is no guarantee that the letter at each position of a close string s is among the most frequent letters at that position in the motif instances, and thus s may not characterize the motif very well. When it is desirable to address this requirement, define the consensus pattern of a set of strings to consist of the most frequent letter at each position (in case of ties, include all the tied letters in the consensus pattern at that position), and define a consensus string to be any string obtained by picking one letter from each position of the consensus pattern. We revise the model as follows.

> CLIQUE$'_{\boldsymbol{k}\mathbf{P}}(\boldsymbol{l}, \boldsymbol{d})$: The problem is CLIQUE$_{\boldsymbol{k}\mathbf{P}}(\boldsymbol{l}, \boldsymbol{d})$, with the additional requirement that $\boldsymbol{s}$ is among one of the consensus strings.

3. Finding cliques of size $\boldsymbol{k}$ in $\boldsymbol{k}$-partite graph

Instead of addressing the full problem directly, we first consider the simpler problem of finding cliques of size k only. This corresponds to the case when every sequence contains a motif instance, which can be applied to more confident samples involving confirmed co-regulated genes. Later we will generalize the approach to find maximum

cliques not necessarily of size k. We first consider the basic problem of finding k-cliques without imposing the close string constraint.

> $\boldsymbol{k}$-CLIQUE$_{\boldsymbol{kP}}$: Given a $\boldsymbol{k}$-partite graph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices in each part, find a $\boldsymbol{k}$-clique in $\boldsymbol{G}$.

We will use a divide-and-conquer approach to handle the problem. The idea is to subdivide the given k-partite graph into several k_0-partite subgraphs with $k_0 < k$ and solve each smaller subproblem independently using a branch-and-bound approach, as long as the number of cliques of size k_0 in each subproblem is not too high. This approach reduces the waste due to repeated computations within the same subgraphs in conventional branch-and-bound techniques when the search is applied to the entire graph. These cliques can then be used as vertices in a second graph in which cliques from the same subproblem reside in the same part and two vertices are connected by an edge if the corresponding cliques together form a larger clique. There is an one-to-one correspondence between cliques in this graph and cliques in the original graph, and a recursive procedure can be used to find them.

Fig. 21 shows the divide-and-conquer algorithm utilizing a recursive procedure. One detail is that k_0 may not divide k evenly. In this case, the last subgraph will have overlapping parts with other subgraphs, requiring a slight modification to the algorithm.

One difficulty with the above approach is that it is hard to determine what k_0 to use. One could use a random k-partite graph model and estimate appropriate values of k_0 based on average case analysis, but significant correlations between adjacent vertices and severe non-randomness around the implanted patterns make such an estimate very unreliable. This problem can be corrected by an automatic approach which subdivides the original graph dynamically.

Algorithm find-clique($G = (\{V^{(1)}, \ldots, V^{(k)}\}, E)$)

for $i \leftarrow 1$ to k/k_0 **do**
  $C^{(i)} \leftarrow$ set of all k_0-cliques in the induced subgraph $G[V^{((i-1)k_0+1)}, \ldots, V^{(ik_0)}]$;
if $(k_0 < k)$ **then**
  construct $G' = (\{C^{(1)}, \ldots, C^{(k/k_0)}\}, E')$, where for $c_p \in C^{(p)}$ and
  $c_q \in C^{(q)}$ with $p \neq q$, $(c_p, c_q) \in E'$ if and only if $c_p \cup c_q$ is a clique in G;
  $C^{(1)} \leftarrow$ set of all cliques from find-clique(G') transformed to
  cliques in G;
return $C^{(1)}$;

Fig. 21. Algorithm find-clique(G) for finding k-cliques in a k-partite graph

Given a threshold t denoting the maximum number of cliques allowed in a subgraph, the idea is to find the smallest induced subgraph $G[V^{(1)}, \ldots, V^{(i)}]$ such that the number of cliques of size i is at most t by trying each i incrementally. To utilize intermediate results as much as possible, we follow a combined breadth-first depth-first approach based on the branch-and-bound technique. Consider each target clique size i in turn (starting from 2), and keep an ordered list L of cliques representing all partial results computed so far, initialized to a single entry containing an empty clique with no vertices. Initialize a pointer P pointing to this entry. Repeat the following: suppose that the clique c pointed by P is of size j with one vertex from each of $V^{(1)}, \ldots, V^{(j)}$. If $j < i$, replace c by a list of all cliques of size $j+1$ that form a clique with c by adding one vertex from $V^{(j+1)}$; otherwise $j = i$ and we advance P to the next entry in L. If the number of cliques in L before P is more than t, the current subgraph has too many cliques: increment i by 1, reset P to point to the first entry in L and continue to look into the next part $V^{(i)}$. If there is no more entry in L, we are done finding all cliques in the subgraph $G[V^{(1)}, \ldots, V^{(i)}]$, repeat this procedure starting with the next part $V^{(i+1)}$ to determine the next subgraph.

By choosing $t = O(n)$, we can make sure that the second clique finding problem is not more complicated than the original. Since the procedure does not waste any partial results, its computational performance should be similar to the previous approach. In terms of memory requirement, with appropriate data structures we only need space for at most t cliques (entries before P) in addition to the normal space requirement for depth-first search (entries after P). To further decrease the number of cliques that need to be considered, when expanding a clique of size j to a clique of size $j + 1$, vertices that do not have a neighbor in all the remaining parts are discarded.

We now incorporate the close string constraint in the model. As discussed before, instead of imposing it every time when a clique is expanded, we check only when the clique size becomes i. During every clique expansion, the following necessary conditions are used to prune impossible branches (proofs omitted).

Proposition B.2 *Given three strings s_1, s_2 and s_3 each of length l, assume that the maximum pairwise distance (number of substitutions) is between s_1 and s_2 with $d(s_1, s_2) = d'$, where $d < d' \leq 2d$. Without loss of generality, assume that all these differences are in the last d' positions of s_1 and s_2, and cut each string s_i into two parts s'_i and s''_i with $|s''_i| = d'$ (in other words, we have $s'_1 = s'_2$ and $d(s''_1, s''_2) = d'$). Let $x = d(s'_1, s'_3) = d(s'_2, s'_3)$, $y = d(s''_1, s''_3)$, and $z = d(s''_2, s''_3)$. Then a string s of length l with distance at most d to each of the three strings exists if and only if $x + y + z \leq 3d$.*

Proposition B.3 *Given a set S of strings all of length l, if there exists a string s of length l with distance (number of substitutions) at most d to every string in S, then (i) for every pair of strings s_1 and s_2 in S with distance $d(s_1, s_2) = 2d$, the jth letter of s must either be the jth letter of s_1 or the jth letter of s_2; (ii) for every pair of strings s_1 and s_2 in S with distance $d(s_1, s_2) = 2d - 1$, when the jth letter of s_1 and*

the jth letter of s_2 are the same, the jth letter of s must be this letter.

In the first condition, if the maximum pairwise distance between three strings is at most d, a close string s always exists. Otherwise, it is in the form of an inequality which can be easily checked given any sub-clique of size 3. To use the second condition, whenever a clique c is expanded to include a vertex v, v is checked against each vertex in c and also against the old constraints on c. It is very useful since the vast majority of edges in the graph have a distance of either $2d$ (typically over 50%) or $2d-1$ (typically over 20%). If any inconsistencies are found, the branch is pruned. Computational experiments show that these conditions help to reduce the subdivision size k_0 in many cases and make the problem much easier to solve (as opposed to not using them). When a new clique of size i is found, we use the branch-and-bound algorithm from Gramm et al. ([58]) to search for a close string s, modifying it to avoid trying letters disallowed from the above constraints to further reduce the search space.

To address the revised model where the close string s is required to be among one of the consensus strings, when a clique reaches size i, we compute its consensus pattern (which may consist of more than one letter at some positions in case of frequency ties) and use it to further restrict the letters that can appear in each position of s. When i is large enough, most positions of the consensus pattern consist of only one letter and thus this requirement eliminates almost all the need to search for s. However, one must be careful not to use the consensus patterns to prune the search branch since it is still possible that the clique will expand into a solution.

Pevzner and Sze ([98]) used samples containing 20 random sequences each of length 600 as test samples (which are typical sizes of realistic biological samples) and found that finding (15,4)-motifs is very challenging. Buhler and Tompa ([17]) found that the (15,4)-motif problem is not the hardest, but (14,4)-, (16,5)- and (18,6)-

motif problems are considerably more challenging. They showed, by a probabilistic analysis of the (l, d)-motif model, that these problems are already at the prediction limit of motif finding algorithms, which include any sample-driven or pattern-driven approaches. Table III shows the performance of the divide-and-conquer approach for two classes of limiting motif finding problems. The correct motif was found in all cases.

Table III. Typical values of k_0 and typical computation times of the divide-and-conquer approach after the graph construction phase. In each case, the values are consistently obtained over a few runs with t set to 600 on random samples with 20 sequences each of length 600 containing an (l, d)-motif

(l, d)	(11,2)	(13,3)	(15,4)	(17,5)	(19,6)	(10,2)	(12,3)	(14,4)	(16,5)	(18,6)
k_0	2	3	4	6	8	3	5	8	10	13
speed	secs	secs	secs	mins	mins	secs	secs	mins	mins	hrs

These results showed that in almost all cases we were able to subdivide the original problem containing 20 sequences into at least two subproblems. Note that for easier samples (data not shown), it is almost always the case that $k_0 = 2$ and thus the algorithm is very fast. We found that even for the most difficult motif finding problems, the graphs become so sparse in subsequent recursion levels that it is almost always the case that $k_0 = 2$ in these deeper levels and thus the running time becomes negligible when compared to the outermost level. For most samples in Table III, the total number of 20-cliques with a close string was more than one, sometimes even in the tens or hundreds, suggesting that these samples become so difficult that there are random patterns which can serve as motif variances. However, for all these cliques the close string is unique and is always the correct one, and thus at least for these simulated samples, it is sufficient to find only one solution. Price et al. ([100], Table 1)

compared the performance of several algorithms on the (15,4)-motif problem. From this table, we found that on this problem while the exact divide-and-conquer approach is not as fast as the fastest heuristics with 99.7% success rate (in a few seconds), it is much faster than other exact algorithms with 100% success rate (in minutes).

4. Finding maximum cliques in $\boldsymbol{k}$-partite graph

A similar divide-and-conquer technique can be used to find maximum cliques when their size k' is less than k, by subdividing the original k-partite graph into r subgraphs G_i, $1 \leq i \leq r$, such that G_i is k_i-partite and $\sum_{i=1}^{r} k_i = k$, and finding cliques within the subgraphs independently. In each subgraph G_i, we need to look for cliques of size $k'_i \leq k_i$ such that $\sum_{i=1}^{r} k'_i = k'$ over all possible combinations of $k'_1, \ldots, k'_r$. Individual clique finding in subgraphs can be accomplished by branch-and-bound approaches. To find maximum cliques of unknown size, one can start the search with $k' = k$, and iteratively reducing k' until maximum cliques are found.

One drawback with the above approach is that it is difficult to determine the subdivision sizes k_i so that there are not too many cliques within each subgraph. Alternatively, a straightforward direct reduction is possible so that the original divide-and-conquer approach can be used. For each choice of k' parts out of k parts to form an induced k'-partite subgraph, run the original k'-partite clique algorithm (with automatic determination of subdivision sizes) on each subgraph to find k'-cliques. The problem is that there are a total of $\binom{k}{k'}$ possible choices and this number grows very quickly as $k - k'$ increases.

To avoid this combinatorial explosion, we employ a reduction at the graph level to construct a single k'-partite graph $G' = (\{W^{(1)}, \ldots, W^{(k')}\}, E')$ to represent all k'-cliques in the original k-partite graph $G = (\{V^{(1)}, \ldots, V^{(k)}\}, E)$, where $W^{(i)}$ $(= W^{(i)}(k, k'))$ is a set of new vertices representing the vertices in $V^{(i)} \cup \ldots \cup V^{(i+k-k')}$,

and for $i < j$, a vertex v in $W^{(i)}$ is connected to a vertex w in $W^{(j)}$ if $v \in V^{(i+p)}$ and $w \in V^{(j+q)}$ are connected in G with $p \leq q$. It is evident that there is an one-to-one correspondence between k'-cliques in G' and k'-cliques in G, and thus only one application of the divide-and-conquer algorithm is needed to find them. One problem with this reduction is that the size of G' grows very quickly as $k - k'$ increases: if G contains nk vertices, G' contains $n(k - k' + 1)k$ vertices. Although G' is less dense, given threshold t the number of parts in the subgraphs after subdivision and the running time increase substantially as $k - k'$ increases.

The above two reductions stand at two extremes: the first reduction employs a large number of small k'-partite graphs while the second reduction uses a single k'-partite graph G' with many vertices. One approach that lies between these two extremes is to decompose G' into an intermediate number of k'-partite graphs with an intermediate number of vertices. Recall that in the first attempt we subdivide G into r subgraphs such that the ith subgraph G_i is k_i-partite and $\sum_{i=1}^{r} k_i = k$. Let s_i $(= \sum_{j=1}^{i-1} k_j + 1)$ be the index of the first part of G_i within G. By applying the previous reduction to each k_i-partite subgraph G_i, $W_i = \{W^{(s_i)}(k_i, k_i'), \ldots, W^{(s_i+k_i'-1)}(k_i, k_i')\}$ includes all vertices for constructing a single k_i'-partite graph to represent all k_i'-cliques in G_i. For each fixed combination of $k_1', \ldots, k_r'$ such that $\sum_{i=1}^{r} k_i' = k'$, the induced k'-partite subgraph $G'[\cup_{i=1}^{r} W_i]$ represents all k'-cliques to which the dynamic subdivision algorithm can be applied.

In effect, we have decomposed G' into smaller induced subgraphs, one for each combination of $k_1', \ldots, k_r'$, which together specify exactly the same set of k'-cliques as G' and can be searched separately. For each value of r and for each choice of values of k_i, we get one decomposition and the number of decomposed subgraphs increases as r increases, which is a tradeoff against the smaller sizes of these subgraphs. We found that $r = 2$ and $k_1 = k_2$ work well, with a linear increase in the number of

decomposed subgraphs as $k-k'$ increases. When almost all sequences contain a motif instance, the above technique reduces the motif finding problem to a manageable number of subproblems to which the divide-and-conquer approach is applied. We have successfully applied it (with reasonable computation times) to find motifs in very difficult motif finding problems such as samples containing (14,4)- or (16,5)-motifs where 20 out of 21, 22 or 23 sequences each of length 600 contain a motif instance.

To handle the more general case when there can be at most p motif instances per sequence, for a small p, we reduce this problem to the original problem as follows: instead of constructing only one part for each sequence, construct p parts each representing the same sequence, resulting in a kp-partite graph. As before, connect vertices from different parts by an edge if the distance between them does not exceed $2d$, except that vertices representing overlapping positions within the same sequence are not connected.

5. Branch-and-bound algorithm

The above divide-and-conquer approach is very fast when every sequence contains a motif instance (i.e., $k' = k$), but its running time requirement increases rapidly as $k-k'$ increases. To handle cases when many sequences do not contain a motif instance, we have to resort back to traditional branch-and-bound approaches. Standard branch-and-bound techniques employ the following recursive algorithm find-clique(G) to find a maximum clique in $G = (V, E)$: let C_v be the set of cliques returned from applying find-clique on the graph induced by all neighbors of vertex v, output the largest set $\{v\} \cup C_v$ over all $v \in V$ (Bomze et al. [15]). Although these approaches were shown to be feasible only on unrestricted graphs with at most a few hundred vertices, we will show that with appropriate optimizations these approaches can still be applied

to motif finding.

If the goal is to find at most u maximum cliques, once a maximal clique of size m is found, there is no need to look for maximal cliques of size less than m. Also, once more than u maximal cliques of size m are found, there is no need to look for maximal cliques of size m. These observations help us to prune the search tree branches when it is no longer possible to obtain a large enough clique. With the above optimization, it is advantageous to consider the vertices in non-increasing degree order (denoted by $\succ$) so that it is more likely to get large cliques during the earlier branches. To make each maximum clique appear uniquely while simultaneously trying to further reduce the running time, when a vertex v is considered in the search tree, only the vertices w satisfying $v \succ w$ and $(v, w) \in E$ are included in the subgraph at the next recursion level to build larger cliques. During every clique expansion, we use the necessary conditions for the existence of a close string to prune impossible branches. Only when a new potential maximal clique is found, we look for a close string.

Computational experiments show that except in the cases when almost every sequence contains a motif instance (when the divide-and-conquer approach can be many times faster), this enhanced branch-and-bound algorithm is faster than the divide-and-conquer approach. By also allowing edges between vertices in the same sequence, the branch-and-bound approach can be further generalized to handle the cases when there can be at most p motif instances per sequence, for a small p, or even to the case when there can be at most p motif instances in total, without restrictions on the number of motif instances in any particular sequence.

6. Biological samples

Since Price et al. ([100]) reported considerable difficulties in obtaining experimentally confirmed samples with biological motifs that are very difficult to find, our goal here

is simply to show that our model is appropriate and our new algorithms are effective by testing on a few published data sets. For these data sets, most standard sample-driven or pattern-driven approaches are also able to find the motifs. In each case, we assume that l is known while d is unknown and the general strategy is to start with $d = 0$ and try successively larger d. Since the motif finding problem is usually much easier when d is small, this approach does not add much to the computation time.

The sample in Stormo and Hartzell ([116]) contains 18 sequences each of length 105 with experimentally confirmed *E.coli* CRP binding sites. Using $l = 16$, the most accurate solution returned correct sites in 16 out of 18 sequences when $d = 6$. There are three samples in Sze et al. ([117]) (also from *E.coli*), all containing sequences of length 200. The first (ARG) sample contains 9 upstream sequences from genes regulated by the arginine repressor ArgR, with a two-part site in each sequence, where each part is of length 18. The patterns in the two parts are very similar and they are separated by 3 positions in 8 out of 9 sequences. When the patterns were treated as one long motif of length 39, all the 8 two-part sites were found when $d = 14$. With such a large d, this is the only case when the consensus string requirement is required in addition to the close string constraint to limit the search space so as to get reasonable computation time (in the other cases, this is optional). When the patterns were treated as independent motifs of length 18 and we looked for at most two motif instances per sequence, most of the sites were found when $d = 6$. The second (PUR) sample contains 19 upstream sequences from genes regulated by the purine repressor PurR with sites of length 16, and most sequences contain one site. Almost all of these sites were found when $d = 5$. Many of the above cases reduce to either the (16,5)- or (18,6)-motif problems and the divide-and-conquer approach was able to finish the computation within seconds or minutes. The third (CRP) sample is a much more difficult variant of the sample in Stormo and Hartzell ([116]) containing 33 sequences

with variable number of weak sites in each sequence and many sequences contain no sites. Unfortunately, the divide-and-conquer approach was not applicable, and with $l = 16$, the branch-and-bound approach was not able to terminate in a reasonable amount of time when $d = 5$. Partial results revealed that more than 10 correct sites and an approximately correct close string were found when $d = 4$.

7. Discussion

By employing a constrained clique formulation to model motifs, we provide two exact algorithms: a very fast divide-and-conquer approach to handle the cases when almost all sequences contain a motif instance and an optimized branch-and-bound algorithm for the other cases when many sequences do not contain a motif instance. There are a few additional complications that need to be addressed before the new algorithms can be applied in all situations: most biological samples have a non-uniform nucleotide composition, and both l and d are unknown. A good strategy is to use weighted distance values according to the background distribution and try different values of d over a range of values of l, while employing heuristics to avoid wasting time on unrealistically large d for a given l. To compare motifs of different lengths and different number of instances, for each motif a score can be computed based on similarity between predicted motif instances.

To further improve the formulation to better model biological reality, one possibility is to allow degenerate letters in the close string. Although this allows motifs to have more than one dominant letter in some positions, the search space becomes much larger. In fact, one can go a step further to use a profile instead of a string to model a motif, which takes into consideration the relative frequency of all letters within a position. However, it is unclear whether a graph-theoretic approach will be able to help to limit the search space.

C. Chapter Conclusion

In this chapter, we used parameterized algorithms to solve the signaling pathway problem and the motif finding problem from bioinformatics. The signaling pathway problem is a very important problem in molecular biology. Understanding the mechanism of how signals are transmitted in living body can help us to study many fundamental problems in biology. In this chapter, we used the (k, t)-path to formulate the signaling pathway problem into a graph problem. As we designed an algorithm of running time $O(4^k n^{2t} k^{t+\log(t+1)+2.92} t^2)$ for the (k, t)-path problem and in applications, k is usually from 6 to 15 and t is from 1 to 3, the signaling pathway problem was solved very effectively. Furthermore, the (k, t)-path is a generalization of the k-path which was used to formulate the signaling pathway problem before by other researchers. Hence in addition to pathways that have been founded by other models, we also found more pathways that cannot be found by the previous k-path model.

The motif finding problem is among the most well-studied problem in bioinformatics, with important application in the computational identification of regulatory sites given a set of genes believed to be co-regulated. The motif finding problem can be formulated by (l, d) model (proposed by Pevzner and Sze [98]) into a graph problem problem, where l is less than 20 and the d is less than 7. In the chapter, we combined parameter algorithm techniques that take advantage of these small l and k with pattern driven and sample driven techniques to solve the motif finding problem. From the result, our method is very effective and found patterns that could not be found by previous methods for the problem.

Many computational problems from bioinformatics are related to parameters that take only small values. We believe that using parameterized algorithms is a good way to find optimal solutions for NP-hard problems from bioinformatics. With further

improvement of algorithms, more and more NP-hard problems from bioinformatics will be solved efficiently.

CHAPTER VIII

CONCLUSIONS

A. Summary

In this dissertation, we studied parameterized NP-hard problems and their applications in bioinformatics. We are especially interested in parameterized NP-hard problems that are fixed parameter tractable, i.e. that can be solved by algorithms of running time in form of $f(k)n^{O(1)}$, where $f(k)$ depends only on k. Hence if k is not very large, those algorithms are very effective.

In Chapter II, we introduced basic principles of techniques that are suitable for designing parameterized algorithms. These techniques include divide and conquer, color coding and dynamic programming, iterative compression, iterative expansion, and kernelization. All these techniques have been used to solve parameterized NP-hard problems in later chapters.

In Chapter III, we used the branch and bound to study the p-node multiway cut problem, where we improved the time complexity of the best previous algorithm for the problem from $O^*(4^{k^3})$ to $O^*(4^k)$. The p-node multiway cut problem has important applications in networks, such as finding a separator to segregate a set of terminals. This separator can be used to monitor communication among terminals. In addition to significant progress of the algorithm, the method developed in this chapter is very interesting. This new method helped us to solve a long standing open problem – the pd-feedback vertex set problem on directed graphs. Another high point in this chapter is the extended idea about the branch and bound method. In solving the problem, we extended the traditional branch and bound method such that the instance was branched in more than one direction to make the instance simpler, i.e.

values of more than one parameter were changed. This idea enhances the power of using branch and bound method to solve parameterized NP-hard problems.

In Chapter IV, by our newly developed divide and conquer technique for parameterized NP-hard problems, we first designed a randomized parameterized algorithm of running time $O^*(4^k)$ for the PW-PATH problem. Comparing with the previous best algorithm for the problem, our new algorithm not only significantly improves the time complexity, but also greatly improves the space complexity, where the previous best algorithm has a time complexity of $O^*(5.5^k)$ and a space complexity of $O^*(2^k)$ while our new algorithm has a time complexity of $O^*(4^k)$ and a space complexity of $O(nk \log k)$. The divide and conquer is a general technique to deal with parameterized NP-hard problems. In this chapter, we also used this technique to develop improved randomized parameterized algorithms for the PW-r-D MATCHING and PW-r-SET PACKING problems. Then we introduced how to use random set partition technique to solve the P-SET SPLITTING problem. Especially, we gave the first parameterized algorithm of running time in form of $f(k)n^{O(1)}$ for the PW-SET SPLITTING problem. Finally, we introduced (n, k)-universal sets that were first developed by Noa et. al. [93] and used (n, k)-universal sets to derandomize our new randomized algorithms for the PW-PATH, PW-r-D MATCHING, PW-r-SET PACKING and PW-SET SPLITTING problems.

In Chapter V, first, we introduced a color coding scheme of size $O^*(6.4^k)$. A color coding scheme for a set X is a collection of k-coloring functions such that every subset of k elements in X is colored properly by at least one k-coloring function in the scheme. The color coding scheme can be used to derandomized algorithms based on random coloring technique. With our improved scheme, many previous algorithms for the P-PATH, P-MATCHING and P-PACKING problems are greatly improved. Then we combined this new color coding scheme with other two new techniques to improve the

algorithm for the P-3-D MATCHING problem. One technique is the iterative expansion, which in the searching for a solution of size k, begins from a solution of size 1 and iteratively finds a solution of size $i+1$ on the base of a solution of size i until a solution of size k is found. Another technique is that instead of coloring symbols from all three columns of the problem, we only color symbols from two columns. By this technique, we designed an algorithm for the P-3-D MATCHING problem that is better than the algorithm for the more general P-3-SET PACKING problem (note: other algorithms usually solve both problems in the same time complexity).

In Chapter VI, we studied parameterized FEEDBACK VERTEX SET problems on directed (unweighted), undirected unweighted and undirected weighted graphs (PD-FEEDBACK VERTEX SET, P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET). Especially, the algorithm for the PD-FEEDBACK VERTEX SET problem is the first algorithm that has the running time in form of $f(k)n^{O(1)}$. Hence solved a long-standing open problem, i.e. if PD-FEEDBACK VERTEX SET problem is FPT. In the process of solving the PD-FEEDBACK VERTEX SET problem, we first used $O^*(g(k))$ time, where $g(k)$ depends only on k, to transform the PD-FEEDBACK VERTEX SET problem into a new problem called the PD-SKEW SEPARATOR problem. Then we solved the PD-SKEW SEPARATOR problem by techniques that are similar to techniques in Chapter III for the P-NODE MULTIWAY CUT problem. Both of our algorithms for P-FEEDBACK VERTEX SET and PW-FEEDBACK VERTEX SET problems have the running time $O^*(5^k)$, where the running time for the P-FEEDBACK VERTEX SET is very close to the running time of $O^*(4^k)$ for the previous best (randomized) algorithm and the running time for the PW-FEEDBACK VERTEX SET problem is better than the running time of $O^*(6^k)$ for the previous best (randomized) algorithm. One important technique we have used to solve these three problems is the iterative compression which was proposed by Reed et. al. [107]. Other important techniques are extended

branch and bound techniques. Unlike traditional branch and bound techniques that have only one direction (reduce the value of the only parameter) to do branching and one condition to stop branching (when $k = 0$), we have more than one direction to do branching and also more than one condition to stop branching. These extended techniques let us have more flexibility to solve parameterized NP-hard problems.

In Chapter VII, we used parameterized algorithm techniques to solve the signaling pathway problem and the motif finding problem in bioinformatics. The signaling pathway problem is a very important problem in biology, such as to understand how signals are transmitted in living body can help us to study cancers. In this chapter, first we used a better model, the (k, t)-path, to formulate the pathway problem into a parameterized NP-hard problem on graphs. Then we designed a very effective algorithm to find the (k, t)-path on graphs. Our model and algorithm enable us to find pathways more quickly and also to find pathways that cannot be found by other models. The motif finding problem is one of most well-studied problems in bioinformatics. This problem is to find the regulatory sites in DNA sequences. These sites can help us to understand where specific genes are located in DNA sequences and how these genes work. In our study, we first used (l, d) model to formulate the motif finding problem into a graph problem. We then combined our parameterized algorithm techniques with sample-driven and pattern-driven techniques to find more sites that are hard to be found by previous methods.

In the process of solving above problems, we proposed new general techniques that can be used to solve other parameterized NP-hard problems. These techniques include divide and conquer, iterative expansion, probabilistic method to deduce a deterministic kernalization algorithm etc. We have used them to obtain some exciting results. We believe that those new techniques will be used to solve more NP-hard problems.

B. Future Work

Studying parameterized algorithms is a very exciting research direction. In recent years, much improvement has been made in this direction, including better algorithms, methods and techniques. Currently, by parameterized algorithm techniques, even using a normal PC, we can find optimal solutions practically for some NP-hard problems from applications. This is a big progress for solving NP-hard problems.

Traditionally, people try to avoid using NP-hard problems to formulate application problems. As it is hard to find effective algorithms to deal with NP-hard problems, most people think that finding optimal solutions for NP-hard problems is impractical. If people have to solve NP-hard problems, they usually use heuristic methods which can not guarantee to find optimal solutions. However, parameterized algorithms provide new ways to find optimal solutions for NP-hard problems from applications.

We believe that in next 10 years, studying parameterized algorithms is a very important research direction to find optimal solutions for NP-hard problems from applications. Currently, some parameterized algorithms are so effective that they can solve parameterized NP-hard problems quickly in a normal PC even when parameters' values are between 60 to 70. Hence, if we further improve those parameterized algorithms, more NP-hard problems from applications will be solved practically. Progress in computer hardwares or using super computers will further guarantee the possibility of using parameterized techniques to solve NP-hard problems from applications. All above are important motivation that we pay much attention to parameterized algorithms and their applications in our future research. Examples of problems for our further study follows.

PD-FEEDBACK VERTEX SET **problem**: The PD-FEEDBACK VERTEX SET prob-

lem comes from the dead-lock problem in database systems and operating systems. We gave the first algorithm of running time in form of $f(k)n^{O(1)}$, where $f(k) = k!4^k$. The running time of this algorithm is still large. Hence, our next task is trying to improve the time complexity of the algorithm to $c^k n^{O(1)}$ for a constant c, such as $4^k n^{O(1)}$ which is more practical. Another interesting direction is to design an algorithm for the weighted PD-FEEDBACK VERTEX SET problem, i.e. to find an FVS F of size bounded by k with minimum weight, where the weight of F is the weight sum of all vertices in F. In applications, such as the dead lock problem in operating systems, the weight of a vertex can be the priority of the process or the CPU time that the process has used. A minimum weight solution means the minimum expense for removing processes to solve the dead lock. Therefore, in applications, the weighted PD-FEEDBACK VERTEX SET problem is even more important. By the way, the problem of whether the PW-FEEDBACK VERTEX SET problem is FPT is still open.

Motif finding problem: The motif finding problem, i.e. to find special patterns in a set of given DNA sequences, is among the most important problems in computational biology, with applications to the identification of transcription factor binding sites given a set of upstream sequences of potentially co-expressed genes. We developed a method that combines parameterized algorithm techniqurs with sample driven and pattern driven techniques that can find (l, d)-motif quickly and determinately. But our new algorithm cannot find motifs that do not appear in all sequences. This is a place in which we continue working. We are trying to use our new color coding technique for this problem, such as suppose a motif appears in k sequences of n given sequences, we color all n sequences with k colors and make sure that all k sequences with the motif are colored properly, then concatenate sequences with the same color together. After that, the motif will appear in every sequence in the new

instance.

Genome alignment problems: The problem is: given several (eg. 3) DNA sequences of length n from different species, each sequence has several genomes that we can see each as a substring of length l, where l is small. The alignment is to find how these genomes from different species are related. To solve the problem, first, noting that each sequence has $n - l + 1$ substrings of length l, we find a score for every group of substrings from different sequences, then we try to find k groups with maximum score such that two substrings from the same sequence intersect. If we see each substring as a symbol, then the problem is similar to the MATCHING problem, but each symbol is associated to $2l - 2$ symbols, and the final solution is a matching that does not allow any two symbols to be associated.

There are also many other application problems that are NP-hard and associated with parameters from Bioinformatics, Database Systems, Operating Systems, Networks and other areas. Designing parameterized algorithms of running time in form of $f(k)n^{O(1)}$ is a promising direction to solve these problems practically. With more techniques for designing parameterized algorithms and progress of hardware in computers, designing parameterized algorithms will become a more important way to solve NP-hard problems from applications.

REFERENCES

[1] N. Alon, L. Babai, A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, J. Algorithms 7 (4) (1986) 567–583.

[2] N. Alon, O. Goldreich, J. Hastad, R. Peralta, Simple constructions of almost k-wise independent random variables, J. Rand. Struct. and Algorithms 3 (3) (1992) 289–304.

[3] N. Alon, R. Yuster, U. Zwick, Color-coding, J. ACM 42 (4) (1995) 844-856.

[4] G. Andersson, L. Engebretsen, Better approximation algorithms and tighter analysis for set splitting and not-all-equal Sat, Inf. Process. Lett. 65 (5) (1998) 305-311.

[5] M. Ashburner, C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, M. Harris, D. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. Matese, J. Richardson, M. Ringwald, G. Rubin, G. Sherlock, Gene ontology: tool for the unification of biology, Nature Genetics 25 (2000) 25–29.

[6] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer, Heidelberg, 1999.

[7] V. Bafna, P. Berman, T. Fujito, A 2-approximation algorithm for the undirected feedback vertex set problem, SIAM J. Discrete Math. 12 (3) (1999) 289–297.

[8] T. Bailey, C. Elkan, Fitting a mixture model by expectation maximization to discover motifs in biopolymers, in: Proc. 2nd Int. Conf. on Intelligent Systems for Mol. Biol., 1994, pp. 28–36.

[9] A. Becker, R. Bar-Yehuda, D. Geiger, Randomized algorithms for the loop cutset problem, J. Artif. Intell. Res. 12 (2000) 219–234.

[10] H. Bodlaender, A tourist guide through treewidth, Acta Cybernetica 11 (1-2) (1993) 1–21.

[11] H. Bodlaender, On disjoint cycles, Internat. J. Found. of Comput. Sci. 5 (1) (1994) 59–68.

[12] H. Bodlaender, On linear time minor tests with depth-first search, J. Algorithms 14 (1) (1993) 1–23.

[13] H. Bodlaender, A cubic kernel for feedback vertex set, in: STAC, in: Lecture Notes in Comput. Sci., vol. 4393, Springer, 2007, pp. 320–331.

[14] H. Bodlaender, E. Penninkx, A linear kernel for planar feedback vertex set, in: IWPEC, in: Lecture Notes in Comput. Sci., vol. 5018, Springer, 2008, pp. 160–171.

[15] I. Bomze, M. Budinich, P. Pardalos, M. Pelillo, The maximum clique problem, in: Handbook of Combinatorial Optimization, Supplement vol. A, Kluwer Acad. Publ., Dordrecht, 1999, pp. 1–74.

[16] Y. Boykov, O. Veksler, R. Zabih, Markov random fields with efficient approximations, in: Proc. 1998 Conf. on Comput. Vision and Pattern Recog., 1998, pp. 648–655.

[17] J. Buhler, M. Tompa, Finding motifs using random projections, J. Comp. Biol. 9 (2002) 225–242.

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

[19] J. Carter, M. Wegman, Universal classes of hash functions, J. Comput. System Sci. 18 (2) (1979) 143–154.

[20] G. Chartrand, L. Lesniak, Graphs & Digraphs, second ed., Wadsworth & Brooks, Monterey, 1986.

[21] J. Chen, F. Fomin, Y. Liu, S. Lu, Y. Villanger, Improved algorithms for the feedback vertex set problems, in: WADS, in: Lecture Notes in Comput. Sci. vol. 4619, Springer, 2007, pp. 422-433.

[22] J. Chen, D. Friesen, W. Jia, I. Kanj, Using nondeterminism to design efficient deterministic algorithms, Algorithmica 40 (2) (2004) 83–97.

[23] J. Chen, X. Huang, I. Kanj, G. Xia, Linear FPT reductions and computational lower bounds, in: Proc. 36th ACM Symp. on Theory of Computing, 2004, pp. 212–221.

[24] J. Chen, I. Kanj, Improved exact algorithms for Max-Sat, Discrete Applied Math. 142 (1-3) (2004) 17–27.

[25] J. Chen, I. Kanj, W. Jia, Vertex cover: further observations and further improvements, J. Algorithms 41 (2) (2001) 280–301.

[26] J. Chen, Y. Liu, S. Lu, An improved parameterized algorithm for the minimum node multiway cut problem, in: WADS, in: Lecture Notes in Comput. Sci., vol. 4619, Springer, 2007, pp. 495–506.

[27] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, I. Razgon, Directed feedback vertex set problem is FPT, in: Proc. 40th ACM Symp. on Theory of Computing, 2008, pp. 177–186.

[28] J. Chen, S. Lu, Improved algorithm for weighted and unweighted set splitting problems, in: COCOON, in: Lecture Notes in Comput. Sci., vol. 4598, Springer, 2007, pp. 573–547.

[29] J. Chen, S. Lu, S. Sze, F. Zhang, Improved algorithms for path, matching, and packing problems, in: Proc. 18th ACM-SIAM Symp. on Discrete Algorithms, 2007, pp. 298–307.

[30] J. Cong, W. Labio, N. Shivakumar, Multi-way VLSI circuit partitioning based on dual net representation, in: Proc. 1994 IEEE/ACM Internat. Conf. on Computer-Aided Design, 1994, pp. 56–62.

[31] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, Cambridge, 2001.

[32] W. Cunningham, The optimal multiterminal cut problem, DIMACS Series in Discrete Math. and Theoretical Comput. Sci., 5 (1991) 105–120.

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

[34] T. Dandekar, S. Schuster, B. Snel, M. Huynen, P. Bork, Pathway alignment: application to the comparative analysis of glycolytic enzymes, Biochemical J. 343 (1) (1999) 115–124.

[35] C. Deane, L. Salwinski, I. Xenarios, D. Eisenberg, Protein interactions: two methods for assessment of the reliability of high throughput observations, Molecular & Cellular Proteomics 1 (5) (2002) 349–356.

[36] F. Dehne, M. Fellows, M. Langston, F. Rosamond, K. Stevens, An $O(2^{O(k)}n^3)$ fpt algorithm for the undirected feedback vertex set problem, Theory of Comput.

Systems 41 (2007) 479–492.

[37] F. Dehne, M. Fellows, F. Rosamond, An FPT algorithm for set splitting, in: WG, in: Lecture Notes in Comput. Sci., vol. 2880, Springer, 2003, pp. 180–191.

[38] F. Dehne, M. Fellows, F. Rosamond, P. Shaw, Greedy localization, iterative compression, modeled crown reductions: New FPT techniques, and improved algorithm for set splitting, and a novel $2k$ kernelization of vertex cover, in IWPEC, in: Lecture Notes in Comput. Sci., vol. 3162, Springer, 2004, pp. 271–280.

[39] M. Deng, S. Mehta, F. Sun, T. Chen, Inferring domain-domain interactions from protein-protein interactions, Genome Research 12 (10) (2002) 1540-1548.

[40] M. Dom, J. Guo, F. uffner, R. Niedermeier, A. Tru, Fixed-parameter tractability results for feedback set problems in tournaments, in: CIAC, in: Lecture Notes in Comput. Sci., vol. 3998, Springer, 2006, pp. 320–331.

[41] R. Downey, M. Fellows, Fixed parameter intractability, in: Proc. 7th Structural Complexity Conf., 1992, pp. 36–49.

[42] R. Downey, M. Fellows, Fixed Parameter Tractability and Completeness, Complexity Theory: Current Research, Cambridge University Press, Cambridge, 1992.

[43] R. Downey, M. Fellows, Fixed-parameter tractability and completeness I: basic results, SIAM J. Computing 24 (4) (1995) 873–921.

[44] R. Downey, M. Fellows, Parameterized Complexity, Monograph in Computer Science, Springer, New York, 1999.

[45] R. Downey, M. Fellows, Fixed-parameter tractability and completeness II: on completeness for $W[1]$, Theoretic Comput. Sci. 141 (1) (1995) 109–131.

[46] E. Eskin, From profiles to patterns and back again: a branch and bound algorithm for finding near optimal motif profiles, in: Proc. 8th Int. Conf. on Comp. Mol. Biol., 2004, pp. 115–124.

[47] E. Eskin, P. Pevzner, Finding composite regulatory patterns in DNA sequences, Bioinformatics 18 (2002) 354–363.

[48] G. Even, J. Naor, B. Schieber, M. Sudan, Approximating minimum feedback sets and multicuts in directed graphs, Algorithmica 20 (1) (1998) 151–174.

[49] M. Fellows, P. Heggernes, F. Rosamond, C. Sloper, J. Telle, Exact algorithms for finding k disjoint triangles in an arbitrary graph, in: WG, in: Lecture Notes in Comput. Sci., vol. 3353, Springer, 2004, pp. 235–244.

[50] M. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. Rosamond, U. Stege, D. Thilikos, S. Whitesides, Faster fixed-parameter tractable algorithms for matching and packing problems, in: ESA, in: Lecture Notes in Comput. Sci., vol. 3221, Springer, 2004, pp. 311–322.

[51] P. Festa, P. Pardalos, M. Resende, Feedback set problems, in: Handbook of Combinatorial Optimization, Supplement vol. A, Kluwer Acad. Publ., Dordrecht, 1999, pp. 209–258.

[52] L. Ford, D. Fulkerson, Flows in Networks, Princeton University Press, Princeton, New Jersey, 1962.

[53] F. Fomin, S. Gaspers, A. Pyatkin, Finding a minimum feedback vertex set in time $O(1.7548^n)$, in: IWPEC, in: Lecture Notes in Comput. Sci., vol. 4169, Springer, 2006, pp. 184–191.

[54] M. Fredman, J. Komlos, E. Szemeredi, Storing a sparse table with $O(1)$ worst case access time, J. ACM 31 (3) (1984) 538-544.

[55] G. Gardarin, S. Spaccapietra, Integrity of databases: a general lockout algorithm with deadlock avoidance, Modeling in Data Base Management System, G. Nijsssen, ed., (1976) 395–411.

[56] M. Garey, D. Johnson, Computers, Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[57] R. Graham, D. Knuth, O. Patashnik, Concrete Mathematics, second ed., Addison-Wesley, Reading, Massachusetts, 1994.

[58] J. Gramm, R. Niedermeier, P. Rossmanith, Exact solutions for closest string and related problems, in: ISAAC, in: Lecture Notes in Comput. Sci., vol. 2223, Springer, 2001, pp. 441–453.

[59] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, S. Wernicke, Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization, J. Comput. System Sci. 72 (8) (2006) 1386–1396.

[60] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, S. Wernicke, Improved fixed-parameter algorithms for two feeback set problems, in: WADS, in: Lecture Notes in Comput. Sci., vol. 3608, Springer, 2005, pp. 158–168.

[61] M. Gustin, J. Albertyn, M. Alexander, K. Davenport, MAP kinase pathways in the yeast *Saccharomyces cerevisiae*, Microbiology and Molecular Biology Reviews 62 (4) (1998) 1264–1300.

[62] G. Gutin, A. Yeo, Some parameterized problems on digraphs, The Computer J. 51 (3) (2007) 363-371.

[63] G. Hardy, E. Wright, An Introduction to the Theory of Numbers, fifth ed., Oxford University Press, Oxford, 1978.

[64] H. Hu, X. Yan, Y. Huang, J. Han, X. Zhou, Mining coherent dense subgraphs across massive biological networks for functional discovery, Bioinformatics 21 (suppl. 1) (2005) 213–221.

[65] T. Hughes, M. Marton, A. Jones, C. Roberts, R. Stoughton, C. Armour, H. Bennett, E. Coffey, H. Dai, Y. He, M. Kidd, A. King, M. Meyer, D. Slade, P. Lum, S. Stepaniants, D. Shoemaker, D. Gachotte, K. Chakraburtty, J. Simon, M. Bard, S. Friend, Functional discovery via a compendium of expression profiles, Cell 102 (2000) 109–126.

[66] W. Jia, C. Zhang, J. Chen, An efficient parameterized algorithm for m-set packing, J. of Algorithms 50 (2004) 106–117,.

[67] I. Kanj, M. Pelsmajer, M. Schaefer, Parameterized algorithms for feedback vertex set, in: IWPEC, in: Lecture Notes in Comput. Sci., vol. 3162, Springer, 2004, pp. 235–247.

[68] V. Kann, J. Lagergren, A. Panconesi, Approximability of maximum splitting of k-sets and some other APX-complete problems, Inf. Process. Lett. 58 (3) (1996) 105–110.

[69] D. Karger, P. Klein, C. Stein, M. Thorup, N. Young, Rounding algorithms for a geometric embedding of minimum multiway cut, in: Proc. 31th ACM Symp. on Theory of Computing, 1999, pp. 668–678.

[70] D. Karger, M. Levine, Finding maximum flows in undirected graphs seems easier than bipartite matching, in: Proc. 30th ACM Symp. on Theory of Computing,

1998, pp. 69–78.

[71] R. Karp, Reducibility among combinatorial problems, in: Complexity of computer computations, Plenum Press, New York, 1972, pp. 85–103.

[72] U. Keich, P. Pevzner, Finding motifs in the twilight zone, in: Proc. 6th Ann. Int. Conf. on Comp. Mol. Biol., 2002, pp. 195–204.

[73] B. Kelley, R. Sharan, R. Karp, T. Sittler, D. Root, B. Stockwell, T. Ideker, Conserved pathways within bacteria and yeast as revealed by global protein network alignment, in: Proc. National Academy of Sci. USA, vol. 100, 2003, pp. 11394–11399.

[74] J. Kneis, D. Mölle, S. Richter, P. Rossmanith, Divide-and-color, in: WG, in: Lecture Notes in Comput. Sci., vol. 4271, Springer, 2006, pp. 58–67.

[75] I. Koutis, A faster parameterized algorithm for set packing, Inf. Process. Lett. 94 (1) (2005) 7–9.

[76] I. Koutis, Faster algebraic algorithms for path and packing problems, in: ICALP, in: Lecture Notes in Comput. Sci., vol. 5125, Springer, 2008, pp. 575–586.

[77] M. Koyutürk, A. Grama, W. Szpankowski, An efficient algorithm for detecting frequent subgraphs in biological networks, Bioinformatics 20 (Suppl. 1) (2004) 200–207.

[78] M. Koyutürk, A. Grama, W. Szpankowski, Pairwise local alignment of protein interaction networks guided by models of evolution, in: Proc. 9th Ann. Int. Conf. on Comp. Mol. Biol., 2005, pp. 48–65.

[79] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, A. Grama, Pairwise alignment of protein interaction networks, J. Computational Biology 13 (2006) 182–199.

[80] J. Lanctot, M. Li, B. Ma, S. Wang, L. Zhang, Distinguishing string selection problems, Information and Computation 185 (2003) 41–55.

[81] C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, J. Wootton, Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment, Science 262 (5131) (1993) 208–214.

[82] S. Liang, cWINNOWER algorithm for finding fuzzy DNA motifs, in: Proc. 2nd IEEE Computer Society Bioinformatics Conf., 2003, pp. 260–265.

[83] T. Leighton, S. Rao, An approximation max-flow min-cut theorem for uniform multi-commodity flow problems with applications to approximation algorithms, J. ACM 46 (6) (1999) 787–832, 1999.

[84] C. Leiserson, J. Saxe, Retiming synchronous circuitry, Algorithmica 6 (1) (1991) 5–35.

[85] O. Lichtenstein, A. Pnueli, Checking that finite state concurrent programs satisfy their linear specification, in: Proc. 12th ACM Symp. on Principles of Programming Languages, 1985, pp. 97–107.

[86] Y. Liu, S. Lu, J. Chen, S. Sze, Greedy localization and color-coding: improved matching and packing algorithms, in: IWPEC, in: Lecture Notes in Comput. Sci. vol. 4169, Springer, 2006, pp. 84–95.

[87] D. Lokshtanov, C. Sloper, Fixed parameter set splitting, linear kernel and improved running time, in: Algorithms and Complexity in Durham 2005, vol. 4,

2005, pp. 105–113.

[88] A. Lukashin, J. Engelbrecht, S. Brunak, Multiple alignment using simulated annealing: branch point definition in human mRNA splicing, Nucleic Acids Research 20 (1992) 2511–2516.

[89] L. Marsan, M. Sagot, Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification, J. Computational Biology 7 (2000) 345–362.

[90] D. Marx, Parameterized graph separation problems, Theoretical Comput. Sci. 351 (3) (2006) 394–406.

[91] L. Mathieson, E. Prieto, P. Shaw, Packing edge disjoint triangles: a parameterized view, in: IWPEC, in: Lecture Notes in Comput. Sci., vol. 3162, Springer, 2004, pp. 127–137.

[92] B. Monien, How to find long paths efficiently, Annals of Discrete Math. 25 (1985) 239–254.

[93] M. Naor, L. Schulman, A. Srinivasan, Splitters and near-optimal derandomization, in: Porc. 36th IEEE Symp. on Foundations of Comput. Sci., 1995, pp. 182–190.

[94] J. Naor, L. Zosin, A 2-approximation algorithm for the directed multiway cut problem, SIAM J. Compututing 31 (2) (2001) 477–482.

[95] A. Nilli, Perfect hashing and probability, Probability and Computing 3 (1994) 407–409.

[96] H. Ogata, W. Fujibuchi, S. Goto, M. Kanehisa, A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters, Nucleic Acids Research 28 (20) (2000) 4021–4028.

[97] C. Papadimitriou, M. Yannakakis, On limited nondeterminism and the complexity of the V-C dimension, J. Comput. System Sci. 53 (2) (1996) 161–170.

[98] P. Pevzner, S. Sze, Combinatorial approaches to finding subtle signals in DNA sequences, in: Proc. 8th Internat. Conf. on Intelligent Systems for Molecular Biology, 2000, pp. 269–278.

[99] R. Pinter, O. Rokhlenko, E. Yeger-Lotem, M. Ziv-Ukelson, Alignment of metabolic pathways, Bioinformatics 21 (2005) 3401–3408.

[100] A. Price, S. Ramabhadran, P. Pevzner, Finding subtle motifs by branching from sample strings, Bioinformatics Suppl 2 (2003) 149–155.

[101] E. Prieto, C. Sloper, Looking at the stars, Theoretical Comput. Sci. 351 (3) (2006) 437–445.

[102] V. Raman, S. Saurabh, Parameterized complexity of directed feedback set problems in tournaments, in: WADS, in: Lecture Notes in Comput. Sci., vol. 2748, Springer, 2003, pp. 484–492.

[103] V. Raman, S. Saurabh, Parameterized algorithms for feedback set problems and their duals in tournaments, Theoretical Comput. Sci. 351 (3) (2006) 446–458.

[104] V. Raman, S. Saurabh, C. Subramanian, Faster fixed parameter tractable algorithms for finding feedback vertex sets, ACM Trans. Algorithms 2 (3) (2006) 403–415.

[105] V. Raman, S. Saurabh, C. Subramanian, Faster fixed parameter tractable algorithms for undirected feedback vertex set, in: ISAAC, in: Lecture Notes in Comput. Sci., vol. 2518, Springer, 2002, pp. 241–248.

[106] I. Razgon, Exact computation of maximum induced forest, in: SWAT, in: Lecture Notes in Comput. Sci., vol. 4059, Springer, 2006, pp. 160–171.

[107] B. Reed, K. Smith, A. Vetta, Finding odd cycle transversals, Operation Research Letters 32 (4) (2004) 299–301.

[108] J. Schmidt, A. Siegel, The spatial complexity of oblivious k-probe hash functions, SIAM J. Computing 19 (5) (1990) 775–786.

[109] A. Schrijver, Combinatorial Optimization, Springer, Berlin, 2003.

[110] J. Scott, T. Ideker, R. Karp, R. Sharan, Efficient algorithms for detecting signaling pathways in protein interaction networks, J. Computational Biology 13 (2) (2006) 133–144.

[111] R. Sharan, S. Suthram, R. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. Karp, T. Ideker, Conserved patterns of protein interaction in multiple species, in: Proc. National Academy of Sci., USA, vol. 102, 2005, pp. 1974–1979.

[112] A. Silberschatz, P. Galvin, Operating System Concepts, foutth ed., Addison-Wesley, Reading, Massachusetts, 1994.

[113] V. Shoup, A Computational Introduction to Number Theory and Algebra, second ed., Cambridge Univesity Press, New York, 2008.

[114] M. Steffen, A. Petti, J. Aach, P. D'haeseleer, G. Church, Automated modelling of signal transduction networks, BMC Bioinformatics 3:34 (2002) doi:10.1186-2105.

[115] H. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Trans. Software Engineering 3 (1) (1977) 85–93.

[116] G. Stormo, G. Hartzell, Identifying protein-binding sites from unaligned DNA fragments, in: Proc. National Academy of Sci., USA vol. 86, 1989, pp. 1183–1187.

[117] S. Sze, M. Gelfand, P. Pevzner, Finding weak motifs in DNA sequences, in: Pacific Symp. on Biocomputing, 2002, pp. 235–246.

[118] Y. Tohsato, H. Matsuda, A. Hashimoto, A multiple alignment algorithm for metabolic pathway analysis using enzyme hierarchy, in: Proc. 8th Internat. Conf. on Intelligent Systems Molecular Biology, 2000, pp. 376-383.

[119] J. Wang, Q. Feng, An $O^*(3.52^{3k})$ parameterized algorithm for 3-set packing, in: TAMC, in: Lecture Notes in Comput. Sci., vol. bf 4978, 2008, pp. 82–93.

[120] I. Xenarios, D. Rice, L. Salwinski, M. Baron, E. Marcotte, D. Eisenberg, DIP: the Database of Interacting Proteins, Nucleic Acids Research 28 (1) (2000) 289-291.

[121] H. Zhang, C. Ling, An improved learning algorithm for augmented naive Bayes, in: PAKDD, in: Lecture Notes in Comput. Sci., vol. 2035, 2001, pp. 581–586.

[122] U. Zwick, Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint, in: Proc. 9th ACM-SIAM Symp. on Discrete Algorithms, 1998, pp. 201–220.

[123] U. Zwick, Outward rotations: A tool for rounding solutions of semidefinite programming relaxation, with applications to max cut and other problem, in: Proc. 31st ACM Symp. on Theory of Computing, 1999, pp. 679–687.

VITA

Songjian Lu received his bachelor's degree in Mathematics, his master's degrees in Mathematics Biology and in Computer Science, and his doctor of philosophy degree in Computer Science from Guangxi University, Xi'an Jiaotong University, the University of Houston, and Texas A&M University, respectively. His research interests are in parameterized algorithm designing and bioinformatics.

Dr. Lu may be reached at the Department of Computer Science and Engineering, Texas A&M University, TAMU 3112, College Station, TX 77843-3112. His email is sjlu@cs.tamu.edu.

The typist for this dissertation was Songjian Lu.