KERNELIZATION AND ENUMERATION:
NEW APPROACHES TO SOLVING HARD PROBLEMS

A Dissertation

by

JIE MENG

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

May 2010

Major Subject: Computer Science

KERNELIZATION AND ENUMERATION:

NEW APPROACHES TO SOLVING HARD PROBLEMS

A Dissertation

by

JIE MENG

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:

Chair of Committee,	Jianer Chen
Committee Members,	Donald K. Friesen
	Andreas Klappenecker
	Catherine Yan
Head of Department,	Valerie E. Taylor

May 2010

Major Subject: Computer Science

ABSTRACT

Kernelization and Enumeration:
New Approaches to Solving Hard Problems. (May 2010)
Jie Meng, B.S.; M.S., Fudan University
Chair of Advisory Committee: Dr. Jianer Chen

NP-Hardness is a well-known theory to identify the hardness of computational problems. It is believed that NP-Hard problems are unlikely to admit polynomial-time algorithms. However since many NP-Hard problems are of practical significance, different approaches are proposed to solve them: Approximation algorithms, randomized algorithms and heuristic algorithms. None of the approaches meet the practical needs. Recently parameterized computation and complexity has attracted a lot of attention and been a fruitful branch of the study of efficient algorithms. By taking advantage of the moderate value of parameters in many practical instances, we can design efficient algorithms for the NP-Hard problems in practice.

In this dissertation, we discuss a new approach to design efficient parameterized algorithms, kernelization. The motivation is that instances of small size are easier to solve. Roughly speaking, kernelization is a preprocess on the input instances and is able to significantly reduce their sizes.

We present a $2k$ kernel for the CLUSTER EDITING problem, which improves the previous best kernel of size $4k$; We also present a linear kernel of size $7k + 2d$ for the d-CLUSTER EDITING problem, which is the first linear kernel for the problem. The kernelization algorithm is simple and easy to implement.

We propose a quadratic kernel for the PSEUDO-ACHROMATIC NUMBER problem. This implies that the problem is tractable in term of parameterized complexity. We also study the general problem, the VERTEX GROUPING problem and prove it is intractable in term of parameterized complexity.

In practice, many problems seek a set of good solutions instead of a good solution. Motivated by this, we present the framework to study enumerability in term of parameterized complexity. We study three popular techniques for the design of parameterized algorithms, and show that combining with effective enumeration techniques, they could be transferred to design efficient enumeration algorithms.

To My Family

ACKNOWLEDGMENTS

First and foremost, I want to give my greatest gratitude to my advisor Dr. Chen, who has invoked my interest in the study of parameterized complexity. Dr. Chen not only inspired me in research, also taught me about my career plan and my life. He is more than an excellent advisor and close friend to me. Dr. Chen guided me through my Ph.D. study with his wonderful academic achievement and great generosity, he was always willing to help and rectified the mistakes I had made. I would not achieve anything now or in the future without his advices.

I am immensely grateful to Dr. Donald Friesen, Dr. Andreas Klappenecker and Dr. Catherine Yan. They are always willing to help whenever I turned to them. Dr. Friesen always offered his advice any time I need it. I took a great course from Dr. Klappenecker and accomplished a paper with him and his student, I am impressed with his seriousness and excellency in research and teaching. Dr. Yan taught me some of most elegant and interesting mathematical techniques and always offered her kindness to me. I am fortunate to have them as my committee members.

I would like to express my deep appreciation all my colleagues and friends, especially to Dr. Iyad Kanj, Dr. Xia Ge and Dr. Fenghui Zhang. Working with them is always fruitful, their serious attitude and creative ideas in research set excellent examples. Also I thank Bright Micah, Songjian Lu, Yang Liu, Qilong Feng, Jiahao Fan and Yixin Cao who all are in the great group led by Dr. Chen.

I would like to take this opportunity to thank the faculty, stuff and students in the Department of Computer Science and Engineering, their support made my study at Texas A&M University a great experience.

Last but definitely not least, I want to thank my family. My parents are always supportive and at my back, they always offer me their advice and help when I am far away from them. I also thank my wife Yue Lu, for her support and love for me.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The NP-Hardness theory [46] provides a foundation for the study of hardness of computational problems, in the next section we introduce the NP-Hardness theory and popular approaches to solve them.

A. NP-Hard problems

Many computational problems are polynomial-time solvable, i.e. instances of the problem can be solved by algorithms in time bounded by a polynomial in the size of the instances. The SHORTEST PATH problem is a good example. For a graph G, a *path* is a sequence of vertices in G and any consecutive pair of vertices are adjacent, the *length* of the path is the number of the vertices. The SHORTEST PATH problem asks for the shortest path between two vertices u and v in a given graph G. We know that the SHORTEST PATH problem can be solved by Dijkstra' algorithm in almost linear time, [30], so the problem is polynomial-time solvable. On the other hand, no polynomial-time algorithm is known for the LONGEST PATH problem, which asks for the longest simple path in the input graph. The definitions of the two problems are similar, but they have different complexity. In order to identify the hardness of the computational problems, NP-Hardness theory is introduced. [46].

Many computational problems are *optimization* problems. [73] Given an instance of the optimization problem, it seeks the "best" solution to that instance. For example, the SHORTEST PATH problem asks for the shortest path between two vertices u and v in a given graph G. However, a *decision* problem simply asks for a "Yes" or "No" answer to a given instance. For example, is there a path of length at most k connecting u and v in G. A

The journal model is Theoretical Computer Science.

decision problem is "easier" in the sense that if the shortest path between u and v is present, by checking the length of the path, one can easily figure out if there is a path of length at most k. The decision problems are closely related to the optimization problems, we can put bound on the solution to construct the decision problems, as in the example above. Although the NP-Hardness theory studies the decision problems, often it contributes to the optimization problems as well.

A decision problem is *polynomial-time solvable* if there is an algorithm which can compute the answer to the instances in polynomial time. And a decision problem P is *polynomial-time verifiable* if there is an algorithm A such that:

- Given an instance s and a proof for s, A could *verify* whether s is a Yes-instance of P;
- The runtime of A is bounded by a polynomial in the size of s;

For example, the k-PATH problem asks if there is a path of length at least k in the given graph G. The longest path in G can be used as a proof, the algorithm simply verifies if it is a path in G and has at least k vertices, then outputs the answer. Roughly speaking, problems in P are polynomial-time solvable, and problems in NP are polynomial-time verifiable. The original definition of P and NP are different, problems in P are deterministic Turing machine polynomial time solvable, and problems in NP are non-deterministic Turing machine polynomial time solvable. In this dissertation, we adopt the more "algorithmic" definitions. Reader who are interested in Turing machine are referred to the book. [78]

A *polynomial-time reduction algorithm* is a transformation algorithm A, given an instance p of a decision problem P, in polynomial time of the size of p, A could produce an instance q of Q such that p is a Yes-instance of P if and only if q is a Yes-instance of Q. We say that P is polynomial-time reducible to Q. Under many occasions, this reduction is also called "Cook Reduction". The importance of the reduction is as follows: if there is an algorithm A' which can solve Q, one could also solve P by the following procedure.

1. Transform instances of P to instances of Q by the algorithm A;

2. solve instances of Q by A'.

Thus if Q is polynomial-time solvable, so is P, whereas if P is "hard", Q must be hard too. We say that P is no harder than Q.

Now we are ready to present the NP-Hardness theory, recall that a problem is in NP if it is polynomial-time verifiable and a problem is in P if it is polynomial-time solvable. Obviously P is a subset of NP, since problems in P can be solved in polynomial time without any proof. However, whether P is a proper subset of NP is the most famous open problem in theoretical computer science, worthy of million dollars bonus. The*NP-Hard problems* are problems to which all problems in NP are polynomial-time reducible, i.e. problems in NP are no harder than any NP-Hard problem. Furthermore, a problem is NP-Complete if it is NP-Hard and is in NP. NP-Complete problems are considered as the hardest problems in NP, and are believed not in P.

The SATISFIABILITY problem was first proved to be NP-Complete, [30]. After that many problems are proved NP-Hard by reducing known NP-Hard problems to these problems. Garey and Johnson's book [46] provides a comprehensive list of the well-known NP-Complete problems. If any NP-Complete problem is polynomial-time solvable, P = NP. The hypothesis that P $\neq$ NP is a foundation of the hardness theory: NP-Hard problems are believed not polynomial-time solvable. Unfortunately many NP-hard problems are important in practice, different approaches are proposed to solve them. For example, approximation algorithms, randomized algorithms and heuristic algorithms. In the following we briefly introduce these approaches.

1. Approximation algorithm

For the NP-Hard problems, we believe that it is hard to develop polynomial-time algorithms to compute the optimal solutions to them. However, it is still possible to compute *near-optimal* solutions to them efficiently. A polynomial-time algorithm A is an *approximation algorithm* if A could produce near-optimal solutions to some NP-Hard problem. We say that A has an *approximation ratio* $r(n)$ if given an instance of size n, the cost C of the optimal solution is within a factor of the cost C' of the approximate solution returned by A, more precisely,

$$\max\{\frac{C}{C'}, \frac{C'}{C}\} \leq r(n).$$

We illustrate the approach by providing an approximation algorithm for the VERTEX COVER problem. An instance of the VERTEX COVER problem is a graph G, and it asks for a subset of vertices so that its removal also removes all edges in G. The algorithm is simple,

1. Arbitrarily pick an edge $[u, v]$, include u and v in the solution and remove them from G;

2. Repeat the process above until there is no edge in G.

We can prove that the simple algorithm can produce solution at most twice as big as the optimal solution: for any edge $[u, v]$, either u or v must be included in the optimal solution, otherwise the edge $[u, v]$ will not be covered. By including both u and v, we obtained a solution of size at most twice of the optimal solution. For more details about approximation algorithms, readers are referred to the book. [85]

2. Randomized algorithm

In addition to the regular input, some algorithms employ extra random strings to compute the optimal solutions. An algorithm is called a *randomized algorithm* if it takes an extra

random string as input. In general two kinds of randomized algorithms are developed, *Monte Carlo* algorithms and *Las Vegas* algorithm.

A randomized algorithm is called *Monte Carlo* algorithm if it runs a fixed number of steps for all inputs and produces a correct answer with bounded probabilities. A *Las Vegas* algorithm guarantees the correct answers, but the runtime is a random variable whose expectation can be bounded. For more details, reader are referred to the book. [72]

Heuristic algorithms are often provided to compute acceptable solutions to practical instances, it is possible that the heuristic algorithms have bad performance in theory. In many applications, heuristic algorithms present good solutions, but the solutions are not always good. A simple example is an algorithm for the VERTEX COVER problem: Repeatedly include the vertex adjacent to most vertices into the solution until all edges are covered. It is possible that the algorithm produces a good solution, but not necessarily the optimal.

B. Parameterized computation

None of the approaches above meet the practical needs: in practice sometimes it is necessary to efficiently compute exact solutions. The theory of parameterized computation and complexity is recently developed to deal with the NP-Hard problems, we try to overcome the hardness of the NP-Hard problems and design practically efficient algorithms. For many problems in industry and applications, an observation is that in practice they contain moderate values of parameters. By taking the advantage of the small parameters, we could design efficient algorithms for these problems. The readers are referred to the book [35] for more details of parameterized computation and complexity.

In this section, we will introduce the parameterized complexity and computation. The following is the definition of the parameterized problem.

Definition A *parameterized problem* Q is a decision problem (i.e. a language) that is

a subset of $\Sigma * \times N$, where Σ is a fixed alphabet and N is the set of all natural numbers. Thus, each element of Q is of the form (x, k), where the second component, i.e. the natural number k, is the parameter.

We point out that the form of instances of parameterized problems is quite natural. Taking the VERTEX COVER problem as an example, an instance of the decision version of the VERTEX COVER decision problem contains a graph G as input, and ask if there is a set of k vertices in G whose removal also removes all edges in G. The classical definition matches the form of the parameterized definition.

Similar to P versus NP in the classical complexity theory, we show that with moderate value of parameters, some problems can be solve efficiently, for example, the VERTEX COVER problem can be solved in time $O^*(1.2738^k)$ [24]. [1] On the other hand, we believe that some problems can not be solved in time $|x|^{o(k)}$, for example, the CLIQUE problem [25]. To further classify the hardness of the NP-Hard problems, we introduce the fixed-parameter tractability and intractability.

1. Fixed-parameter tractability and intractability

Definition A parameterized problem is *fixed-parameter tractable* (FPT) if instances (x, k) of the problem can be solved by a parameterized algorithm in time $f(k)|x|^{O(1)}$, where f is a recursive function, k is the parameter and x is the input. Denote by FPT the class of all fixed-parameter tractable problems.

In this dissertation, we focus on the design of fixed-parameter tractable algorithms for

[1] We note that the O*() notation may omit insignificant polynomial factors for simplicity

the NP-hard problems. With the moderate values of parameters, the runtime of fixed-parameter tractable algorithms is actually polynomials in the size of the input, which are considered as effective algorithms in general. Many problems have fixed-parameter tractable algorithms, for example, the VERTEX COVER problem we mentioned above, and the FEEDBACK VERTEX SET problem, which admits an algorithm of runtime $O^*(5^k)$ [26].

However, there are some NP-Hard problems which we believe do not admit fixed-parameter tractable algorithms, in the following we define the intractability of the parameterized problems, which, similar to NP-Hardness in classical complexity theory, identifies the hard problems in parameterized complexity. Before we present the definition, we introduce a group of satisfiability problems.

A *circuit* C of n variables is a directed acyclic graph, in which each node of in-degree 0 is an input gate and is labeled by either a positive literal x_i or a negative literal $\bar{x}_i$, where $1 \leq i \leq n$. All other nodes in C are called gates and labeled by a Boolean operator either AND or OR. A designated gate of out-degree 0 in C is the output gate. The circuit C computes a Boolean function in a natural way. The size of the circuit C is the number of nodes in C, and the depth of C is the length of a longest path from an input gate to the output gate in C. The circuit C is a Π_t-circuit if its output is an AND gate and its depth is bounded by t. An assignment τ to the input variables of the circuit C satisfies C if τ makes the output gate of C have value 1. The weight of an assignment τ is the number of variables assigned value 1 by τ.

The parameterized problem WEIGHTED SATISFIABILITY on Π_t-circuits, abbreviated $WCS[t]$, consists of the pairs (C, k), where C is a Π_t-circuit and k is the parameter, and C admits a satisfying assignment of weight k. The parameterized problem WEIGHTED CNF FORMULA SATISFIABILITY, abbreviated WCNF-SAT, consists of the pairs (F, k), where F is a CNF Boolean formula and k is the parameter such that the formula F has a satisfying assignment of weight k. Finally, the WEIGHTED CNF 3-SAT problem, abbreviated WCNF-

3SAT, is the WCNF-SAT problem whose instances satisfy a further condition that every clause in the CNF Boolean formula F contains at most three literals.

Extensive studies on the problem WCNF-3SAT and the problems $WCS[t]$, for all $t > 1$, show that they are unlikely in FPT. To identify more hard parameterized problems, we introduce a new type of reduction for parameterized problems.

Definition A parameterized problem Q is FPT-reducible to a parameterized problem Q' if there is an algorithm that on a given instance (x, k) of Q produces an instance (x', k') of Q' in time $O(f(k)|x|^{O(1)})$, where $k' \leq g(k)$, and f and g are recursive functions, such that (x, k) is a yes-instance of Q if and only if (x', k') is a yes-instance of Q'.

The FPT reduction is transitive, and preserves the fixed-parameter tractability, i.e. if the problem Q' admits a FPT algorithm, and Q is FPT-reducible to Q', we can construct a FPT algorithm for Q by reducing the instances of Q to instances of Q' and solving them.

Definition The class $W[1]$ consists of all parameterized problems that are FPT-reducible to the problem WCNF-3SAT. For each integer $t > 1$, the class $W[t]$ consists of all parameterized problems that are FPT-reducible to the problem $WCS[t]$.

This gives us the fixed-parameter intractability hierarchy, the W-hierarchy $\{W[t]|t \geq 1\}$:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[t] \subseteq \ldots$$

From the definition above, we obtain a natural complete problem for each of the level in the W-Hierarchy, i.e. WCNF-3SAT is $W[1]$-complete, and $WCS[t]$ is $W[t]$-complete for $t > 1$. Other problems are proved fixed-parameter intractable [35], for example, the CLQIUE problem is shown in $W[1]$, and the DOMINATING SET problem is shown in $W[2]$. Even these

problems can be solved in $O(f(k)n^k)$ by trivial enumeration algorithms, it has shown that they are unlikely to be solved in time $f(k)n^{o(k)}$ [25].

Our main working hypothesis is that $FPT \neq W[i],\ \forall i \geq 1$.

2. Kernelization

Kernelization is a new technique to design fixed-parameter tractable algorithms, and has been an important contribution of parameterized computation. The technique is motivated by the fact that small size instances are easier to solve, in general kernelization is regarded as "preprocessing" or "data-reducing". The formal definition is the following:

Definition A *kernelization* algorithm for a parameterized problem P is an algorithm A, given an instance (x, k) of P as input, the output of A is another instance (x', k') of P, such that

- (x, k) is a Yes-instance of P if and only if (x', k') is a Yes-instance;
- The runtime of A is bounded by a polynomial over $|x| + k$;
- $|x'| \leq f(k)$, $k' \leq k$, where f is a recursive function;

For an instance (x, k) of P, A produces a kernel x' of x with the size bounded by $f(k)$, we say that P admits a kernel of size f. Obviously if we bound the instance size by a function $f(k)$, we could solve it in time $O(g(f(k)) + p(|x| + k))$, where g is the runtime of any algorithm for problem P, and $p()$ is the runtime of the kernelization algorithm, so P is in FPT. Moreover the following theorem shows that all problems in FPT have kernels.

Theorem B.1 *[34] Let Q be a parameterized problem, Q is fixed-parameter tractable if and only if Q has a kernel.*

We can illustrate the kernelization algorithms by providing a kernel for the LINE COVER problem. The input of the LINE COVER problem is a set of n points in a given plane, it asks for k lines which cover all the points. The problem is fundamental and we can show that a simple algorithm can reduce the size of the instances to k^2.

We enumerate all lines connecting pairs of points in the input set: if the line can cover more than k points, we include it in all optimal solutions, and remove points it covers. If the input set has more than k^2 points left, the instance is a No-instance; Otherwise we obtain a small instance with no more than k^2 points. The proof is as follows: if a line covers at least $k+1$ points, it must be included in all optimal solutions. Otherwise we need at least $k+1$ lines to cover the $k+1$ points that it covers; After the preprocessing, no line can cover more than k points. If there is a solution consisting of k lines, they can cover at most k^2 points.

We have shown that simple ideas can reduce the instance size significantly. Moreover kernelization closely relates to the development of approximation algorithms. For example, the VERTEX COVER problem has a $2k$ kernel. Simply including all vertices in the kernel in the solution, we obtain an approximation algorithm with ratio 2 for the VERTEX COVER problem. Thus any improvement on the kernel will lead to a better approximation algorithm. On the other hand, it is well-known that problems in the class $MIN\ F^+\ \Pi_1$ and $MAX\ NP$ have polynomial-time approximation algorithms with constant factors, Kratsch [63] shows that problems in the two classes admit polynomial kernels for their natural decision version, which improves the result of Cai and Chen [19] that problems in the two classes are in FPT. And this result is strengthened by Bodlaender et al. [15]. The following theorem appears in [13].

Theorem B.2 *Let g be a fixed integer. Let P be a CMSO-expressible property of graphs and vertex sets. Consider a problem Q, whose input consists of a graph $G = (V, E)$ of Euler-genus at most g, a set of vertices $Y \subseteq V$, and an integer k. Suppose Q is compact or the complement of G is compact.*

1. *If Q is of the form: $\exists S \subseteq Y : |S| \leq k \bigwedge P(G, S)$, then Q has a kernel of size $O(k^2)$.*
2. *If Q is of the form: $\exists S \subseteq Y : |S| = k \bigwedge P(G, S)$, then Q has a kernel of size $O(k^3)$.*
3. *If Q is of the form: $\exists S \subseteq V : |S \cap Y| \geq k \bigwedge P(G, S)$, then Q has a kernel of size $O(k^2)$.*

Lower bound results on kernelization are also proposed, Chen et al. [23] show linear lower bounds on certain parameterized problems, for example, the PLANAR VERTEX COVER problem does not have a kernel size smaller than $4k/3$. Recently the idea of *or-composition* algorithms is proposed to prove that certain parameterized problems do not have kernel of size bounded by a polynomial in k, in another word, the problem does not have a polynomial kernel. We show the approach as follows.

Conjecture (Or-distillation conjecture [14]). Let P be an NP-Complete problem, there is no algorithm A, which takes m instances of P as input, and output an instance of P, such that:

- The runtime of A is bounded by a polynomial in $m + n$, where m is the number of instances in the input, and n is the maximum size of the instances;
- The output of A is a Yes-instance of P if and only if one of the instances in the input is a Yes-instance.

Fortnow et al. [43] show that if the Or-distillation conjecture fails, $NP \subseteq coNP/poly$.

Definition An *or-composition* algorithm A for a parameterized problem P works as follows: A takes r instance $((x_1, k), (x_2, k), \ldots, (x_r, k))$ of P as input, and output an instance (x', k') of P, such that:

- the runtime of A is bounded by a polynomial in $\sum_{1\leq i\leq r}|x_i|+k$;
- k' is bounded by a polynomial in k;
- $(x',k')\in P$ if and only if $\exists i$, $1\leq i\leq r$ and $(x_i,k)\in P$.

Or-composition algorithm is the key to prove the lower bound on polynomial kernel size. We show the details in the following theorem.

Theorem B.3 *[14] P is a parameterized problem, its classical version is NP-Complete and P has an or-composition algorithm; If P admits a kernel of size bounded by a polynomial in k, the or-distillation conjecture fails*

We can illustrate the approach by proving that the k-PATH problem does not admit a polynomial kernel. A *path* in an input graph is a sequence of vertices where any pair of consecutive vertices are adjacent, a path is of length k if it contains k vertices. The k-PATH problem is defined as follows: given an input graph G and an integer k, it asks if there is a path of length k in G. We can easily construct an or-composition algorithm for the problem, given r input instances $((G_1,k),(G_2,k),\ldots,(G_r,k))$, the algorithm simply output the union of the r graphs $(G_1\bigcup G_2\bigcup\ldots\bigcup G_r,k)$. The runtime of the algorithm is $\sum_{1\leq i\leq r}|G_i|+k$, and the graph $G_1\bigcup G_2\bigcup\ldots\bigcup G_r$ has a path of length k if and only if there is an instance graph in the input which has a path of length k.

C. This dissertation

In this dissertation, we discuss the new approach to design fixed-parameter tractable algorithms, kernelization; We also propose the framework to study the enumerability of pa-

rameterized problems and introduce the concept of the *fixed-parameter enumerable* (FPE) problems.

1. Kernelization

We present improved kernels for several problems, including the CLUSTER EDITING problem, the d-CLUSTER EDITING problem and the PSEUDO-ACHROMATIC NUMBER problem.

In Chapter II, we present a $2k$ kernel for the CLUSTER EDITING problem, which improves the previous known $4k$ kernel size [53]. The CLUSTER EDITING problem is NP-Hard [81], has been studied by groups in biological research [28, 32, 80]. The CLUSTER EDITING problem takes a graph as input, and asks if one can insert/delete at most k edges in the graph to transfer the graph to a union of disjoint cliques. Given the critical clique graph, we define the *editing degree* of vertices. Roughly speaking, the editing degree of a vertex v is the number of inserted/deleted edges adjacent to v which are applied to make the cluster a disjoint clique, the cluster consists of vertices including v, one of its adjacent critical clique and neighbors of that critical clique. We develop several reduction rules to reduce the size of the input graph so that the resulting graph contains no more than $2k$ vertices.

In Chapter III, we present a linear kernel for the d-CLUSTER EDITING problem , which is a variant of the CLUSTER EDITING problem, and in addition requires that the resulting graph consists of exactly d disjoint cliques. The d-CLUSTER EDITING problem is harder than the original one in the sense that we can solve the CLUSTER EDITING problem by an algorithm for the d-CLUSTER EDITING problem with the value of d varying from 1 up to $|x|$, the size of the input.

We introduce the *class-partition* of an input graph, and show that the optimal solution to the d-CLUSTER EDITING problem can be computed from a class-partition by simply splitting or combining some classes. Based on the observation above, we present a linear kernel of size $7k + 2d$ for the problem. Furthermore, we present a branch-and-search algorithm,

and combining it with spliting/combining algorithms we develop the first fixed-parameter tractable algorithm for the d-CLUSTER EDITING problem.

In Chapter IV, we proposed a quadratic kernel for the PSEUDO-ACHROMATIC NUMBER problem, which is a variance of the famous GRAPH COLORING (CHROMATIC NUMBER) problem. The problem is that if we can color the vertices of a given graph with different colors so that in any pair of groups of vertices colored by the same color, there is at least one edge connecting two vertices, one in each group. The maximum number of colors we can use to color the graph is the *pseudo-achromatic number* of the graph.

We develop a kernel for this problem, this implies that the PSEUDO-ACHROMATIC NUMBER problem is fixed-parameter tractable. The techniques used to develop the kernel are elegant and of independent interest. We also study the generalization of the PSEUDO-ACHROMATIC NUMBER problem, the VERTEX GROUPING problem. Although the special subproblem is in FPT, we show that the general problem is fixed-parameter intractable.

2. Enumeration

In practice, many problems seek a set of good solutions instead of a good solution. Motivated by this, we present a framework to study the enumerability of parameterized problems. Unlike algorithms for the counting problems, the outputs of the enumeration algorithms are a set of good solutions instead of the number of total solutions. Especially the counting version of the k-PATH problem is fixed-parameter intractable, however we can still efficiently enumerate solutions of the k-PATH problem.

We define the class of fixed-parameter enumerable (FPE) problems. Solutions to the parameterized problems in FPE can be efficiently enumerated. FPE is a proper subset of FPT, to explore the connection of them we study three popular techniques for the design of fixed-parameter tractable algorithms – branch-and-search, color coding and tree decomposition. We show that cooperating with effective enumeration techniques, they could be

transferred to design efficient enumeration algorithms.

CHAPTER II

AN IMPROVED KERNEL FOR THE CLUSTER EDITING PROBLEM

The CLUSTER EDITING problem for a given graph G and a given parameter k asks if one can apply at most k edge insertion/deletion operations on G so that the resulting graph is a union of disjoint cliques. The problem has attracted much attention recently because of its applications in bioinformatics. In this section dissertation, we present a polynomial time kernelization algorithm for the problem that produces a kernel of size bounded by $2k$, improving the previously best kernel of size $4k$ for the problem.

A. Introduction

The CLUSTER EDITING problem is formulated as follows: given a graph G and a parameter k, is it possible to apply at most k edge insertion/deletion operations so that the resulting graph becomes a union of disjoint cliques?

The CLUSTER EDITING problem arises from many application areas [61]. In particular, it has been recently studied by a number of research groups in biological research [28, 32, 80]. An example of this line of research is the analysis of gene expression data, in which a critical step is to identify the groups of genes that manifest similar expression patterns. The corresponding problem in algorithmic research is the GENE CLUSTERING problem [80]. An instance of the GENE CLUSTERING problem consists of a set of genes, and a measure of similarity of genes. A threshold can be used to differentiate the similarity of the genes. The goal is to partition the genes into *clusters* that achieve both *homogeneity* (genes in the same cluster are highly similar) and *separation* (genes from different clusters have low similarity) criteria.

Therefore, when an instance of the GENE CLUSTERING problem is given, and a measure

threshold is provided, we can represent the instance as a graph G, whose vertices correspond to the genes and whose edges correspond the high similarity between the genes. Ideally, if the similarity measure is perfect and the measure threshold is precise, the graph G should be a union of disjoint cliques. Unfortunately, the nature of biological research provides biological data by which the graph G can only be "close" to a union of disjoint cliques. This motivates the algorithmic research of the CLUSTER EDITING problem, which tries to "correct" a small number k of similarity pairs (i.e., apply a small number k of edge insertion/deletion operations on the graph G) so that the resulting graph becomes a union of disjoint cliques.

There have been extensive algorithmic research on the CLUSTER EDITING problem. The optimization version of the problem was first studied by Ben-dor, Shamir and Yakhini [18]. Shamir, Sharan, and Tsur [81] proved that the problem is NP-hard. Approximation algorithms for the problem have been studied. The currently best polynomial time approximation algorithm for the problem has an approximation ratio 2.5 [88]. It is also known that the problem is APX-complete, thus it is unlikely that the problem has a polynomial time approximation scheme [20].

Given the fact that the parameter k is small in the applications of bioinformatics, research on parameterized algorithms and complexity for the CLUSTER EDITING problem has become active recently [38, 40, 47, 48, 53, 81]. The first parameterized algorithm of time $O(2.27^k + n^3)$ for the CLUSTER EDITING problem was developed in [48], which was improved to $O(1.92^k + n^3)$ [47], current the best parameterized algorithm takes time $O(1.82^k + n^3)$ [11]. A research direction closely related to the parameterized algorithms is the study of the kernelization of the problem. We say that the problem CLUSTER EDITING *has a kernel of size* $g(k)$ if there is a polynomial-time algorithm that reduces an instance (G, k) of the problem to an equivalent instance (G', k') where the graph G' has at most $g(k)$ vertices. Gramm *et al* [48] showed that the CLUSTER EDITING problem has a kernel of size $2k^2 + k$.

Fellows [38] announced an improved kernel of size $24k$ for the problem, and conjectured that a kernel of size bounded by $6k$ for the problem should exist. The conjecture was confirmed later in [40]. The kernel size for the CLUSTER EDITING problem was further improved to $4k$ by Guo [53] based on the idea of *critical cliques* [70, 33].

In this section, we develop a new polynomial-time kernelization algorithm that provides a kernel of size $2k$ for the CLUSTER EDITING problem, which improves the previous best result.

B. Reduction rules

We start with necessary definitions. A *clique* K in a graph G is a subgraph of G that is a complete graph. A *disjoint clique* is a clique K in which no vertex is adjacent to any vertex not in K. For a vertex v, denote by $N(v)$ the set of vertices that are adjacent to v. For a subset S of vertices, denote by $G[S]$ the subgraph of G that is induced by S, by $N(S)$ the set of vertices that are not in S but adjacent to some vertex in S, i.e., $N(S) = \bigcup_{v \in S} N(v) - S$, and by $N_2(S)$ the neighbors of $N(S)$ that are not in $S \cup N(S)$, i.e., $N_2(S) = N(N(S)) - (S \cup N(S))$.

Definition A *critical clique* K in a graph G is a clique such that for all vertices u and v in K, $N(v) - K = N(u) - K$, and K is maximal under this property.

It has been proved [70] that every vertex in a graph G belongs to a unique critical clique. Therefore, the vertices of the graph G are uniquely partitioned into groups such that each group induces a critical clique. The *critical clique graph* G_c of the graph G is defined as follows. Vertices of G_c correspond to critical cliques in G, and two vertices in G_c are adjacent if the union of the corresponding critical cliques in G induces a larger clique in G.

It is known [58] that for a given graph G, the critical clique graph G_c of G can be constructed in linear time. For a critical clique K, in case there is no confusion, we also denote by K the vertex set of the critical clique.

A *solution* to a graph G for the CLUSTER EDITING problem is a sequence of edge insertion/deletion operations that converts G into a collection of disjoint cliques. The solution to the graph G can be represented by a partition $\mathcal{P} = \{C_1, C_2, \ldots, C_h\}$ of the vertex set of G, where each vertex subset C_i (called a *cluster* of $\mathcal{P}$) becomes a disjoint clique after the edge insertion/deletion operations of the solution. An *optimal solution* to G is a solution that uses the minimum number of edge insertion/deletion operations.

Proposition B.1 ([53]) *Let K be a critical clique in a graph G. Then in any optimal solution $\mathcal{P}$ to G, the critical clique K is entirely contained in a single cluster of $\mathcal{P}$.*

According to Proposition B.1, no edge whose both ends are in the same critical clique needs to be considered when we are looking for an optimal solution to a given graph.

Let K be a critical clique in a graph G and suppose that we want to make $K \cup N(K)$ a disjoint clique. Then we need to add edges between vertices in $N(K)$ if the edges are missing, and delete edges that have one end in $N(K)$ and the other end not in $K \cup N(K)$. Motivated by this, we introduce the following definition.

Definition Let K be a critical clique in a graph G and let $v \in N(K)$, the *editing degree* $p_K(v)$ of v with respect to K is defined to be the number of vertex pairs $\{v, w_1\}$, where $w_1 \in N(K) - \{v\}$ and $[v, w_1]$ is not an edge, plus the number of edges $[v, w_2]$, where $w_2 \notin K \cup N(K)$.

Let S be a vertex subset in a graph G, by *making S a disjoint clique*, we mean to perform the following edge operations to make S a disjoint clique: adding edges between

pairs of vertices in S that are not adjacent, and deleting edges that are between a vertex in S and a vertex not in S.

Now we are ready to describe our reduction rules. Let (G, k) be an instance of the CLUSTER EDITING problem, and let K be a critical clique in G.

Reduction Rules

Rule 1 if $|K| > k$, then make $K \cup N(K)$ a disjoint clique, remove $K \cup N(K)$ from G, and decrease k by p, where p is the number of edge operations that make $K \cup N(K)$ a disjoint clique;

Rule 2 if $|K| \geq |N(K)|$ and $|K|+|N(K)| > \sum_{v \in N(K)} p_K(v)$, then make $K \cup N(K)$ a disjoint clique, remove $K \cup N(K)$ from G, and decrease k by p, where p is the number of edge operations that make $K \cup N(K)$ a disjoint clique;

Rule 3 if $|K| < |N(K)|$ and $|K| + |N(K)| > \sum_{v \in N(K)} p_K(v)$, and if there is a vertex $u \in N_2(K)$ with $|N(u) \cap N(K)| > (|K| + |N(K)|)/2$, then insert necessary edges among vertices in $N(K)$ to make $K \cup N(K)$ a clique, remove edges between $N(K)$ and $N_2(K) - u$, and decrease k accordingly.

In the remaining of this section, we verify that the above rules are all "safe", i.e., the edge operations applied by each of rules are entirely contained in an optimal solution to the graph G for the CLUSTER EDITING problem.

Lemma B.2 *Rule 1 is safe.*

PROOF. Suppose that an optimal solution $\mathcal{P}$ to the graph G uses no more than k edge operations to make G a collection of disjoint cliques. By Proposition B.1, the critical clique K must be entirely contained in a single cluster C in the optimal solution $\mathcal{P}$. If any vertex

v_1 in $N(K)$ is not in C, then the solution $\mathcal{P}$ would have to delete at least the $|K| > k$ edges between v_1 and K, contradicting the assumed number of edge operations by $\mathcal{P}$. On the other hand, if any vertex v_2 not in $N(K)$ is in C, then the solution $\mathcal{P}$ would have to insert at least the $|K| > k$ edges between v_2 and K, again contradicting the assumed number of edge operations by $\mathcal{P}$. Therefore, the cluster C in $\mathcal{P}$ must consist of exactly the vertices in $K \cup N(K)$, and all edges operations applied by Rule 1 are contained in the optimal solution $\mathcal{P}$. □

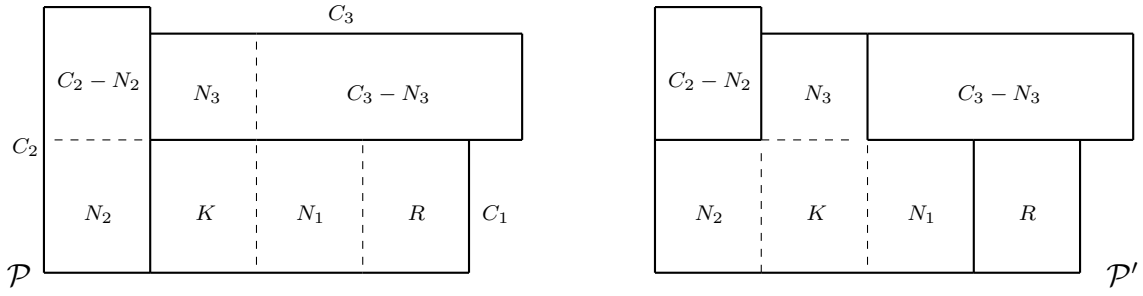

Fig. 1. The critical clique K and the solutions $\mathcal{P}$ and $\mathcal{P}'$

Now we consider Rules 2 and 3. For this, let K be a critical clique in the graph G, and let $\mathcal{P} = \{C_1, C_2, \ldots, C_h\}$ be an optimal solution to the graph G, where C_i, $1 \leq i \leq h$, are the clusters in $\mathcal{P}$. By Proposition B.1 and without loss of generality, we can assume $K \subseteq C_1$. Let $N_i = C_i \cap N(K)$ for $1 \leq i \leq h$. Note that some N_i can be empty. Let $R = C_1 - K - N_1$ (see the left figure in Figure 1 for an illustration, where $h = 3$).

We also define another solution $\mathcal{P}'$ to the graph G based on the above notations: $\mathcal{P}' = \{K \cup N(K), R, C_2 - N_2, \ldots, C_h - N_h\}$ (see the right figure in Figure 1 for an illustration), and will compare the number of edge operations of the solutions $\mathcal{P}$ and $\mathcal{P}'$.

Besides the edge operations that are common to $\mathcal{P}$ and $\mathcal{P}'$, the solution $\mathcal{P}$ does the following edge operations:

$\mathcal{P}$-operation

(1) inserting missing edges between K and R;

(2) deleting all edges between N_i and K for $i \geq 2$;

(3) inserting missing edges between N_1 and R.

(4) inserting missing edges between N_i and $C_i - N_i$ for $i \geq 2$; and

(5) deleting all edges between N_i and N_j, for $i \neq j$, $1 \leq i, j \leq h$,

while the solution $\mathcal{P}'$ does the following edge operations:

$\mathcal{P}'$-operation

(1') deleting all edges between N_1 and R;

(2') inserting missing edges between N_i and N_j, for $i \neq j$, $1 \leq i, j \leq h$; and

(3') deleting all edges between N_i and $C_i - N_i$, for $i \geq 2$.

Lemma B.3 *Let G be a graph and let K is a critical clique in G with $|K| \geq |N(K)|$, and for all $v \in N(K)$, $p_K(v) \leq |K|$. Then there is an optimal solution to G that has $K \cup N(K)$ as a cluster.*

PROOF. By the definitions, there is no edge between K and R in the graph G. Thus, the $\mathcal{P}$-operation set (1) contains exactly $|K| \cdot |R|$ edge insertion operations. Also by definition, each vertex in $N(K)$ is adjacent to every vertex in K in the graph G. Thus, the $\mathcal{P}$-operation set (2) contains exactly $|K|(|N_2| + \cdots + |N_h|)$ edge deletion operations. In conclusion, the solution $\mathcal{P}$ contains at least $|K|(|R| + |N_2| + \cdots + |N_h|)$ edge operations that are not in the solution $\mathcal{P}'$.

Now the $\mathcal{P}'$-operation set (1') contains at most $|N_1| \cdot |R| \leq |K| \cdot |R|$ edge deletion operations (here we have used the lemma assumption $|N(K)| \leq |K|$). The number of edge

operations in the $\mathcal{P}'$-operation sets (2') and (3') is bounded by

$$\sum_{v \in N_2 \cup \cdots \cup N_h} p_K(v) \leq |K|(|N_2| + \cdots + |N_h|),$$

here we have used the lemma assumption $p_K(v) \leq |K|$ for all $v \in N(K)$. Also note that $\sum_{v \in N_2 \cup \cdots \cup N_h} p_K(v)$ includes all operations in set (2') that insert missing edges between N_1 and N_j for $j \geq 2$. In conclusion, the solution $\mathcal{P}'$ contains at most $|K|(|R| + |N_2| + \cdots + |N_h|)$ edge operations that are not in the solution $\mathcal{P}$.

By the above comparison, we conclude that the number of edge operations in the solution $\mathcal{P}'$ is not larger than that in the solution $\mathcal{P}$. Since $\mathcal{P}$ is an optimal solution and $\mathcal{P}'$ contains $K \cup N(K)$ as a cluster, the lemma is proved. □

Note that for each vertex v in $N(K)$, where K is a critical clique, we can always assume that $p_K(v) \geq 1$. In fact, if $p_K(v) = 0$, then $N(K)$ would consist of a single critical clique and $K \cup N(K)$ would make a disjoint clique in the graph G. Thus, in this case, we can directly reduce the problem instance (G, k) to the smaller instance $(G - (K \cup N(K)), k)$.

Corollary B.4 *Rule 2 is safe.*

Proof. By the conditions of Rule 2, $|K| \geq |N(K)|$ and $\sum_{v \in N(K)} p_K(v) \leq |K| + |N(K)| - 1$. For each vertex v in $N(K)$, we have

$$\begin{aligned} p_K(v) &= \sum_{u \in N(K)} p_K(u) - \sum_{u \in N(K),\ u \neq v} p_K(u) \leq \sum_{u \in N(K)} p_K(u) - (|N(K)| - 1) \\ &\leq (|K| + |N(K)| - 1) - (|N(K)| - 1) = |K|, \end{aligned}$$

here we have used the fact $p_K(v) \geq 1$ for all v. Thus, under the conditions of Rule 2, all conditions of Lemma B.3 are satisfied so by the lemma, there is an optimal solution that has $K \cup N(K)$ as a cluster. Therefore, the edge operations of Rule 2 are all contained in an

optimal solution to the graph G. In consequence, Rule 2 is safe. □

Now only Rule 3 remains.

Lemma B.5 *Let K be a critical clique with $|K| < |N(K)|$ and $\sum_{v\in N(K)} p_K(v) < |K| + |N(K)|$. There is an optimal partition $\mathcal{P}$ such that $K \cup N(K)$ is entirely contained in a single cluster in $\mathcal{P}$.*

Proof. Again let $\mathcal{P} = \{C_1, C_2, \ldots, C_h\}$ is an optimal solution to the graph G, and let $\mathcal{P}' = \{K \cup N(K), R, C_2 - N_2, \ldots, C_h - N_h\}$, as described in Figure 1. If $\mathcal{P}'$ is an optimal solution to G, then the lemma is proved. Thus, we suppose that $\mathcal{P}'$ is not an optimal solution to G.

By the lemma assumption $\sum_{v\in N(K)} p_K(v) < |K| + |N(K)|$, we have

$$\begin{aligned}\sum_{v\in N(K)-N_1} p_K(v) &= \sum_{v\in N(K)} p_K(v) - \sum_{v\in N_1} p_K(v) \\ &\leq (|K| + |N(K)| - 1) - |N_1| = |K| + (|N(K)| - |N_1|) - 1. \quad (2.1)\end{aligned}$$

Note that the total number of missing edges between N_i and N_j for $i \neq j$ is upper bounded by $\sum_{v\in N(K)-N_1} p_K(v)$. Therefore, the total number of existing edges between $N(K) - N_1$ and K and between N_i and N_j for $i \neq j$ is at least

$$\begin{aligned}&(|K| + |N_1|)(|N(K) - N_1|) - \sum_{v\in N(K)-N_1} p_K(v) \\ \geq\ &(|K| + |N_1|)(|N(K) - N_1|) - (|K| + (|N(K) - N_1|) - 1).\end{aligned}$$

This gives a lower bound on the number of edges deleted by the $\mathcal{P}$-operation sets (2) and (5). With the $|K| \cdot |R|$ edge insertion operations in the $\mathcal{P}$-operation set (1), we conclude that the solution $\mathcal{P}$ contains at least $(|K| + |N_1|)(|N(K) - N_1|) - (|K| + (|N(K) - N_1|) - 1) + |K| \cdot |R|$ edge operations that are not contained in the solution $\mathcal{P}'$. On the other hand, the total

number of edge operations in the $\mathcal{P}'$-operation sets (1')-(3') that are not in $\mathcal{P}$ is upper bounded by $\sum_{v\in N(K)} p_K(v)$. Since $\mathcal{P}'$ is not an optimal solution to G, we must have

$$\begin{aligned} & (|K|+|N_1|)(|N(K)-N_1|)-(|K|+(|N(K)-N_1|)-1)+|K|\cdot|R| \\ < & \sum_{v\in N(K)} p_K(v) \le |K|+|N(K)|-1, \end{aligned}$$

which gives

$$|K|(|N(K)-N_1|+|R|-2)+|N(K)|(|N(K)-N_1|-1) < |N(K)-N_1|^2+|N(K)-N_1|-2 \quad (2.2)$$

We first consider the case $|R|>0$. Then we must have $|N_1|>0$: if $|N_1|=0$, the $\mathcal{P}$-operation set (1) would have been unnecessary and the solution $\mathcal{P}$ would have not been an optimal solution to G. Therefore, in this case, we have

$$\begin{aligned} & |K|(|N(K)-N_1|-1)+(|N(K)-N_1|+1)(|N(K)-N_1|-1) \\ \le & |K|(|N(K)-N_1|-1)+(|N(K)-N_1|+|N_1|)(|N(K)-N_1|-1) \\ \le & |K|(|N(K)-N_1|+|R|-2)+|N(K)|(|N(K)-N_1|-1) \\ < & |N(K)-N_1|^2+|N(K)-N_1|-2. \end{aligned}$$

The last inequality has used the inequality (2.2). However, This cannot hold true unless $|N(k)-N_1|=0$, i.e., $N(k)=N_1$. Thus, in this case, we have $K\cup N(K)\subseteq C_1$ and the lemma is proved.

This leaves us with the remaining case $|R|=0$. As we have analyzed above, the solution $\mathcal{P}$ contains at least $(|K|+|N_1|)(|N(K)-N_1|)-(|K|+(|N(K)-N_1|)-1)$ edge operations that are not contained in the solution $\mathcal{P}'$ (note that $|R|=0$). On the other hand, the total number of operations in the $\mathcal{P}'$-operation sets (2') and (3') is bounded by $\sum_{v\in N(K)-N_1} p_K(v) \le |K|+(|N(K)-N_1|)-1$, where we have used the inequality (2.1). Thus (noting that the $\mathcal{P}'$-operation set (1') is empty because $|R|=0$), the total number of edge

operations in $\mathcal{P}'$ that are not in $\mathcal{P}$ is bounded by $|K| + (|N(K)| - |N_1|) - 1$. Since $\mathcal{P}'$ is not optimal, we must have

$$(|K| + |N_1|)(|N(K) - N_1|) - (|K| + (|N(K) - N_1|) - 1) < |K| + (|N(K)| - |N_1|) - 1. \quad (2.3)$$

We show that the inequality (2.3) can never hold true.

If $|N_1| = 0$, then all the $|K| \cdot |N(K)|$ edges between K and $N(K)$ should be deleted in the solution $\mathcal{P}$. By the lemma assumption $|N(K)| > |K| \geq 1$, we have

$$|K| \cdot |N(K)| \geq |K| + |N(K)| + 1 \geq \sum_{v \in N(K)} p_K(v).$$

Since $\sum_{v \in N(K)} p_K(v)$ upper bounds the total number of edge operations that are in $\mathcal{P}'$ but not in $\mathcal{P}$, the above inequality would imply that $\mathcal{P}'$ is an optimal solution, contradicting our assumption.

If $|N(K) - N_1| = 1$, then the inequality (2.3) would give $|N_1| < |K|$, which implies $|N(K)| = |N_1| + |N(K) - N_1| = |N_1| + 1 \leq |K|$, contradicting the lemma assumption $|N(K)| > |K|$.

Finally, if $|N(K) - N_1| \geq 2$ and $|N_1| > 0$, then from the inequality (2.3), we would have $(|N(K) - N_1| - 2)|K| < |N(K) - N_1| - 2$, which is again impossible.

This verifies that either $\mathcal{P}'$ is an optimal solution that has $K \cup N(K)$ as a cluster, , or the optimal solution $\mathcal{P}$ has a cluster that contains $K \cup N(K)$. The lemma now follows directly. □

In fact, we can derive a result that is stronger and more precise than Lemma C.3.

Lemma B.6 *Let K be a critical clique with $|K| < |N(K)|$ and $\sum_{v \in N(K)} p_K(v) < |K| + |N(K)|$. There is an optimal partition $\mathcal{P}$ that either has $K \cup N(K)$ as a cluster, or has a cluster that contains $K \cup M(K)$ plus a single vertex u, such that $|N(u) \cap N(K)| > (|K| + |N(K)|)/2$.*

PROOF. As we have proved in Lemma C.3, either the solution $\mathcal{P}'$ in Figure 1 that has $K \cup N(K)$ as a cluster is an optimal solution, or the optimal solution $\mathcal{P}$ in Figure 1 has a cluster C_1 that contains $K \cup N(K)$ plus a vertex subset R.

There are at most $\sum_{v \in N(K)} p_K(v) \leq |K| + |N(K)| - 1$ edges between R and $N(K)$ in G, since $\sum_{v \in N(K)} p_K(v)$ is an upper bound on the number of edges between $N(K)$ and $N_2(K)$. To construct a disjoint clique induced by C_1, at least $|R| \cdot (|K| + |N(K)|) - (|K| + |N(K)| - 1)$ edges are inserted. Since $\mathcal{P}$ is optimal, to construct two disjoint cliques induced by R and $C_1 - R$ is at least as expensive as that of constructing the disjoint clique induced by C_1. Therefore,

$$|K| + |N(K)| - 1 \geq |R| \cdot (|K| + |N(K)|) - (|K| + |N(K)| - 1),$$

which gives $|R| \cdot (|K| + |N(K)|) \leq 2(|K| + |N(K)| - 1)$. This cannot be true for $|R| \geq 2$. Therefore, we must have $|R| \leq 1$. If $R = \{u\}$, the number of vertices in $N(K)$ that are adjacent to u is larger than the number of vertices in $K \cup N(K)$ that are not adjacent to u, i.e., $|N(u) \cap N(K)| > |K| + |N(K)| - |N(u) \cap N(K)|$. This gives immediately $|N(u) \cap N(K)| > (|K| + |N(K)|)/2$.

Moreover, u is the only vertex with $|N(u) \cap N(K)| > (|K| + |N(K)|)/2$: otherwise there would be more than $2((|K| + |N(K)|)/2)$ edges between $N(K)$ and $N_2(K)$, which already exceeds the upper bound $\sum_{v \in N(K)} p_K(v) \leq |K| + |N(K)| - 1$. □

Now we are ready for Rule 3.

Corollary B.7 *Rule 3 is safe.*

PROOF. By the conditions in the rule, $|K| < |N(K)|$ and $\sum_{v \in N(K)} p_K(v) < |K| + |N(K)|$. By lemma C.3, there is an optimal solution $\mathcal{P}$ in which a cluster C_1 contains the entire

$K \cup N(K)$. Moreover, by Lemma C.4, there is at most one vertex u in G that satisfies $|N(u) \cap N(K)| > (|K| + |N(K)|)/2$ and the vertex u is the only possible other vertex in the cluster C_1. Since all edge operations in Rule 3 are contained in the optimal solution $\mathcal{P}$, the rule is safe. □

In reduction rule 3, there are two cases: If no vertex u exists in $N_2(K)$ with $|N(u) \bigcap N(K)| > (|K| + |N(K)|)/2$, $K \bigcup N(K)$ is a cluster in an optimal partition, we can safely remove $K \bigcup N(K)$ from G; On the other hand, if there is a vertex u with $|N(u) \bigcap N(K)| > (|K| + |N(K)|)/2$, we can remove K from G by calling the *pendulum algorithm* with (G, k, u) as input. We will present the details of the algorithm after the kernelization algorithm.

C. The kernelization algorithm

In this section, we present the kernelization algorithm and prove its correctness. Given an input graph G and a parameter k, our kernelization algorithm is the following:

Kernelization Algorithm

Step 1 Repeatedly reduce G according to the reduction rules, until they are not applicable;

Step 2 If the resulting graph G' contains more than $2k$ vertices, output "No"; Otherwise output G';

The following theorem proves the correctness of the kernelization algorithm.

Theorem C.1 (G, k) *is an instance of the* CLUSTER EDITING *problem, if the reduction rules are not applicable on* G*,* G *must contain no more than* $2k$ *vertices, otherwise there is no solution for* G*.*

PROOF. Let $\mathcal{P}$ be an optimal partition for G, $\mathcal{P} = \{C_1, C_2, C_3, \ldots, C_h\}$. We can easily construct a solution of size no larger than k based on $\mathcal{P}$. We say that a vertex v is *touched* if we inserte/delete at least one edge adjacent to v to obtained the disjoint cliques induced by $\mathcal{P}$, and *untouched* otherwise; $\forall v \in V(G)$, $p(v)$ denotes the number of inserted/deleted edges adjacent to v.

We divide clusters in $\mathcal{P}$ into two sub-collections: $\mathcal{P}_1$ contains clusters in which all vertices are touched, $\mathcal{P}_2$ contains other clusters. Since clusters in $\mathcal{P}_1$ contains only touched vertices, $p(v) \geq 1$ for any touched vertex v, so the size of the clusters in $\mathcal{P}_1$ is $\sum_{C \in \mathcal{P}_1} |C| \leq \sum_{v \in C, C \in \mathcal{P}_1} p(v)$.

For a cluster C in $\mathcal{P}_2$, C contains a set K of untouched vertices, we claim that the induced graph on K is a critical clique. Since we do not insert/delete any edges adjacent to vertices in K, the induced graph on K must be a clique, otherwise C does not become a disjoint clique after applying the solution, it is a contradiction. And if there are two vertices in $u, v \in K$ with $N(u) - K \neq N(v) - K$, to make C a disjoint clique, we have to insert/delete at least one edge adjacent to either u or v, and one of the vertices becomes touched, it leads to another contradiction, and proves our claim. Furthermore, the cluster C contains all vertices in $K \bigcup N(K)$ but no other vertices, since we do not insert/delete any edges adjacent to K to make C a disjoint clique.

For a cluster C in $\mathcal{P}_2$, $C = K \bigcup N(K)$, K is a vertex subset and the induced graph on K is a critical clique. Vertices in K are untouched and vertices in $N(K)$ are touched. Since the reduction rules are not applicable, we have $|C| = |K| + |N(K)| \leq \sum_{v \in N(K)} p_K(v) = \sum_{v \in N(K)} p(v) = \sum_{v \in C} p(v)$, the last two quality hold because by the definition of editing degrees, $p(v) = p_K(v)$ for $v \in N(K)$ and $p(v) = 0$ for $v \in K$. Thus $\sum_{C \in \mathcal{P}_2} |C| \leq \sum_{v \in C, C \in \mathcal{P}_2} |p(v)|$

In both cases, we bound the size of clusters by $\sum_{v \in V(G)} p(v)$, and at most k edge insertion/deletion operations are applied, $\sum_{v \in V(G)} p(v)$ is bounded by $2k$, so is the number of vertices in G, it completes our proof. □

D. The pendulum algorithm

In this section, we present the *pendulum algorithm*(Figure 2), it is applied with rule 3, and is of independent interest.

Algorithm **Pendulum Algorithm**

INPUT: (G, k, u): *G is the input graph, and k is the parameter, u is a vertex in G such that K is a critical clique in G, $N(K)$ is a critical clique and $N_2(K) = \{u\}$*

OUTPUT: *A graph G' and a parameter k*

1. If $|K| \geq |N(K)|$, remove $K \bigcup N(K)$ from G, let $k' = k - |N(K)|$;
2. If $|K| < |N(K)|$, remove K from G and delete $|K|$ vertices from $N(K)$, let $k' = k - |K|$.
3. Let G' be the resulted graph, return G' and k'.

Fig. 2. Pendulum algorithm

Lemma D.1 *The pendulum algorithm is correct.*

PROOF. We prove lemma C.5 by showing that there is an solution of size k for G if and only if there is a solution of size k' for G'.

If $|K| \geq |N(K)|$, and $N_2(K) = \{u\}$, $\forall v \in N(K)$, $p_K(v) = 1 \leq |K|$. By theorem B.3, there is an optimal partition which contains $K \bigcup N(K)$ as a cluster, the algorithm is correct.

If $|K| < |N(K)|$, similarly $\forall v \in N(K)$, $p_K(v) = 1$, $\sum_{v \in N(K)} p_K(v) = |N(K)| < |N(K)| + |K|$. By lemma C.3 and C.4, there is an optimal solution consisting of k insertion/deletion operations, and the optimal partition $\mathcal{P}$ containing a cluster S so that

$K \bigcup N(K) \subseteq S \subseteq K \bigcup N(K) \bigcup \{u\}$.

Let R be the set of removed vertices in $N(K)$ by the pendulum algorithm, $|R| = |K|$. We construct an partition $\mathcal{P}'$ of vertices in G' by replacing S by $S' = S - K - R$ in $\mathcal{P}$, and we can show that it costs $k' = k - |K|$ many edge operations to construct the disjoint cliques induced by $\mathcal{P}'$, there are two cases:

(i) If u is in S, the solution induced by $\mathcal{P}$ contains $|K|$ edges insertion operations to connect u and K, which are not in the solution induced by $\mathcal{P}'$;

(ii) If u is not in S, the solution induced by $\mathcal{P}$ contains $|K|$ edge deletion operations to disconnect u and R, which are not in the solution induced by $\mathcal{P}'$.

In both cases, we obtain a solution of size $k' = k - |K|$ for G' by removing $|K|$ edge operations from the solution induced by $\mathcal{P}$.

On the other hand, suppose $\mathcal{P}'$ is an optimal partition of vertices in G', based on $\mathcal{P}'$, a solution of size $k' = k - |K|$ can be easily constructed. $N(K)$ is a critical clique in G, $N(K) - U$ is also a critical clique in G', by proposition B.1, $N(K) - U$ is entirely contained in a cluster S' in $\mathcal{P}'$.

We can show that S' is a subset of $(N(K) - U) \bigcup \{u\}$. Suppose that $D = S' - (N(K) - U) - u \neq \emptyset$. To make S' a disjoint clique, edges between D and $N(K) - U$ are inserted, $|D| * |N(K) - U|$ many edge insertion operations are applied. On the other hand, to construct disjoint cliques induced by $N(K) - U$ and $D \bigcup \{u\}$, $|N(K) - U|$ many edges between u and $N(K) - U$ are deleted. Since $|D| * |N(K) - U| \geq |N(K) - U|$, replacing S' by $N(K) - U$ and $D \bigcup \{u\}$, we obtain a new optimal partition, which contains a cluster S', and S' is a subset of $(N(K) - U) \bigcup \{u\}$.

We construct a partition $\mathcal{P}$ of vertices in G by replacing S' by $S = S' \bigcup K \bigcup U$, and show that the solution induced by $\mathcal{P}$ contains k edge insertion/deletion operations. There

are two cases:

(i) S' contains u, in addition to the solution induced by $\mathcal{P}'$, there are $|K|$ many edge insertion operations to connect u and K to construct disjoint cliques induced $\mathcal{P}$.

(ii) S' does not contain u, in addition to the solution induced by $\mathcal{P}'$, there are $|K|$ many edge deletion operations to disconnect u and R to construct disjoint cliques induced by $\mathcal{P}$.

In both cases, we show that the solution induced by $\mathcal{P}$ consists of k edge insertion/deletion operations and it complete the proof. □

E. Final remarks

The CLUSTERING EDITING problem arises from biological research. In the section, we study the its feasibility in term of parameterized complexity. We present a $2k$ kernel for the problem, it improves the previous best kernel of size $4k$. We introduce the concept of editing degree of vertices, which play a key role in the kernelization algorithm and the analysis of the algorithm. Our kernelization algorithm is simple and easy to implement in practice.

CHAPTER III

AN IMPROVED KERNEL FOR THE D-CLUSTER EDITING PROBLEM

The d-CLUSTER EDITING problem is a variance of the CLUSTER EDITING problem which, in addition, requires that the resulting graph consists of a union of d disjoint cliques. We present a polynomial-time kernelization algorithm for the problem that produces a linear kernel of size bounded by $7k + 2d$, improving the previously best kernel size $(d + 2)k + d$. We also propose a fixed-parameter tractable algorithm for the problem.

A. Introduction

The general CLUSTER EDITING problem is formulated as follows: given a graph G and a parameter k, is it possible to apply at most k edge insertion/deletion operations on G so that the resulting graph becomes a union of disjoint cliques? The d-CLUSTER EDITING problem further requires that the resulting graph contains a union of d disjoint cliques.

In the chapter, we study the parameterized complexity of the d-CLUSTER EDITING problem. The d-CLUSTER EDITING problem arises from certain biological applications [6, 51] where the number d of gene clusters in gene partition is known in advance (e.g., in the study of K-means [80]). Formally, an instance of the d-CLUSTER EDITING problem consists of a graph G and a parameter k, and is asking for at most k edge insertion/deletion operations that convert the graph G into a union of *exactly* d disjoint cliques. The d-CLUSTER EDITING problem is NP-hard [81]. Moreover, it is easy to see that the general CLUSTER EDITING problem can be reduced to the d-CLUSTER EDITING problem by solving the latter for all d, $0 \leq d \leq n$. Therefore, an improved parameterized algorithm for the d-CLUSTER EDITING problem may directly imply an improvement on parameterized algorithms for the general CLUSTER EDITING problem. On the other hand, it is not clear whether the d-CLUSTER

EDITING problem can be reduced to the CLUSTER EDITING problem. Guo, based on the idea of critical cliques, presented a polynomial time constructible kernel of size $(d+2)k+d$ for the d-CLUSTER EDITING problem [53], which implies that the d-CLUSTER EDITING problem is fixed parameter tractable when both d and k are used as parameters.

The major difficulty for reducing the d-CLUSTER EDITING problem to the CLUSTER EDITING problem is that when the number of disjoint cliques in the resulting graph for a solution to the CLUSTER EDITING problem is significantly different from d, it is unclear what is the relationship between this resulting graph and the graph resulted from a desired solution for the d-CLUSTER EDITING problem. To overcome this difficulty, we introduce a new concept of *class-partitions* of a graph G, which is a partition of the vertices in G into classes. Our key observation on the d-CLUSTER EDITING problem is that for each desired solution S of the d-CLUSTER EDITING problem, there is a class-partition C such that the solution S can be obtained from C by simple split or combination of the classes in C. Therefore, a fixed parameter algorithm for the d-CLUSTER EDITING problem can proceed by enumerating the class-partitions, followed by a dynamic programming procedure that implements a proper split/combination process on each obtained class-cluster. Based on this technique, we obtain a kernelization algorithm that gives a kernel of size $7k+2d$ for the d-CLUSTER EDITING problem, improving the previous kernel size $(d+2)k+d$ [53].

The technique also enables us to derive a fixed parameter algorithm of running time $O^*(\max\{2.56^k, 2^{k+d}\})$[1] for the d-CLUSTER EDITING problem.

B. Key lemmas

We first present necessary definitions. A *clique* K is a subgraph of graph G and is complete. A *disjoint clique* K is a clique in which no vertex is adjacent to any other vertex not in K.

[1]We note that the $O^*()$ notation may omit certain insignificant polynomial factors

For a vertex v, denote by $N(v)$ the set of vertices that are adjacent to v. For a subset S of vertices, denote by $G[S]$ the subgraph of G that is induced by S, by $N(S)$ the set of vertices that are not in S but adjacent to some vertex in S, i.e., $N(S) = \bigcup_{v \in S} N(v) - S$, and by $N_2(S)$ the neighbors of $N(S)$ that are not in $S \cup N(S)$, i.e., $N_2(S) = N(N(S)) - (S \cup N(S))$.

Definition ([70]) A *critical clique* K in a graph G is a clique such that for all vertices u and v in K, $N(v) - K = N(u) - K$, and K is maximal under this property.

A *critical clique graph* G_c of G is a graph such that nodes in G_c are critical cliques in G, and two nodes are adjacent in G_c if and only if the subgraph in G induced by the two cliques is a larger clique. Lin et al. [70] proved that every vertex in a graph G belongs to a unique critical clique, therefore the critical clique graph of G is well defined. And it is known [58] that the critical clique graph of a graph can be constructed in linear time. The critical clique graph is an important part in the kernelization algorithm. For a critical clique K, in case there is no confusion, we also denote by K the vertex set of the critical clique.

A *solution* to the d-CLUSTER EDITING problem is a sequence of edge insertion/deletion operations that can convert G into a collection of disjoint cliques. The vertex sets of the disjoint cliques is a partition $\mathcal{S}$ of vertices in G, $\mathcal{S} = \{C_1, C_2, \ldots, C_d\}$, a *cluster* is a vertex subset in $\mathcal{S}$. Given a solution, the partition can be easily derived and vice versa. An *optimal* solution is a solution that uses the minimum number of edge insertion/deletion operations, and an *optimal* d-partition is obtained from an optimal solution. Let S be a vertex subset of G, by *making S a disjoint clique*, we mean to perform the following edge operations to make S a disjoint clique: adding edges between pairs of vertices in S that are not adjacent, and deleting edges that are between a vertex in S and a vertex not in S.

Now we are ready to present the useful lemmas to derive the kernel.

Lemma B.1 *G is a graph and K is a critical clique in G. Suppose that $\mathcal{S} = \{C_1, C_2, \ldots, C_d\}$ is an optimal d-partition in G, there is at most one cluster C_i in the partition $\mathcal{S}$ such that $K \bigcap C_i$ is a proper nonempty subset of C_i.*

PROOF. We prove this lemma by contradiction. Let C_x and C_y are vertex subsets in $\mathcal{S}$, $1 \leq x < y \leq d$ so that $K_x = C_x \bigcap K$ and $K_y = C_y \bigcap K$ are proper nonempty subsets of C_x and C_y respectively. We will derive a new d-partition of G, from which we obtain a solution with less edge operations. This contradicts the fact that $\mathcal{S}$ is an optimal d-partition.

Let $N_x = N(K_x) \bigcap C_x$ and $N_y = N(K_y) \bigcap C_y$, and $R_x = C_x - K_x - N_x$, $R_y = C_y - K_y - N_y$. To make C_x and C_y disjoint cliques, we need to apply

$$|K_x| * |K_y| + |K_x| * |N_y| + |K_y| * |N_x| + |K_x| * |R_x| + |K_y| * |R_y| + M. \quad (1)$$

many edge insertion/deletion operations. We only concern the edge operations on K or related to K, and M denote the number of the others.

There are two cases: $|N_x| - |N_y| \leq |R_x| - |R_y|$ and $|N_x| - |N_y| > |R_x| - |R_y|$. In the first case, we construct a new d-partition $\mathcal{S}'$, in which C_x and C_y are replaced by $N_x \bigcup R_x$ and $K \bigcup N_y \bigcup R_y$, denoted by C'_x and C'_y. To make C'_x and C'_y disjoint cliques, the number of edge operations is:

$$\begin{aligned} &(|K_x| + |K_y|) * |N_x| + (|K_x| + |K_y|) * |R_y| + M \\ =&|K_x| * |N_x| + |K_y| * |N_x| + |K_x| * |R_y| + |K_y| * |R_y| + M. \quad (2) \end{aligned}$$

(1)-(2):

$$\begin{aligned}
&|K_x| * |K_y| + |K_x| * |N_y| - |K_x| * |N_x| + |K_x| * |R_x| - |K_x| * |R_y| \\
=&|K_x| * |K_y| + |K_x| * (|N_y| - |N_x| + |R_x| - |R_y|) \\
=&|K_x| * |K_y| + |K_x| * ((|R_x| - |R_y|) - (|N_x| - |N_y|)) \\
>&0
\end{aligned}$$

Thus to convert G into the graph consisting a disjoint union of d cliques whose vertex sets are exact $\mathcal{S}'$, we will apply less edge operations than we derive the new graph from $\mathcal{S}$.

In the seconde case, $|N_x|-|N_y| > |R_x|-|R_y|$, we construct a new d-partition $\mathcal{S}'$, similarly C_x and C_y are replaced by $K \bigcup N_x \bigcup R_x$ and $N_y \bigcup R_y$, denoted by C'_x and C'_y. To make C'_x and C'_y disjoint cliques, the number of edge operations is:

$$\begin{aligned}
&(|K_x| + |K_y|) * |N_y| + (|K_x| + |K_y|) * |R_x| + M \\
=&|K_x| * |N_y| + |K_y| * |N_y| + |K_x| * |R_x| + |K_y| * |R_x| + M. \quad (3)
\end{aligned}$$

(1)-(3):

$$\begin{aligned}
&|K_x| * |K_y| + |K_y| * |N_x| - |K_y| * |N_y| + |K_y| * |R_y| - |K_y| * |R_x| \\
=&|K_x| * |K_y| + |K_y| * (|N_x| - |N_y| + |R_y| - |R_x|) \\
=&|K_x| * |K_y| + |K_y| * ((|N_x| - |N_y|) - (|R_x| - |R_y|)) \\
>&0
\end{aligned}$$

Therefore in both cases, we construct a new d-partition, from which we apply less edge operations to convert G into a graph consisting of a union of d disjoint cliques, this contradicts the fact that $\mathcal{S}$ is optimal d-partition. □

Intuitively, Lemma B.1 claims that for any graph G, $\mathcal{S} = \{C_1, C_2, \ldots, C_d\}$ is an optimal d-partition of the vertices in G, K is a critical clique such that there exists C_i with $K \bigcap C_i \subset C_i$, then for all other C_j with $j \neq i$, either $K \bigcap C_j = \emptyset$ or $C_j \subseteq K$.

Our main contribution in this paper is the following concept and its applications to fixed-parameter tractable algorithms for the d-CLUSTER EDITING problem.

Definition Let G be a graph. A *class-partition* of the graph G is a partition $\mathcal{P} = \{V_1, V_2, \ldots, V_h\}$ of the vertices in G such that:

- For all i, the subgraph $G[V_i]$ of G induced by the vertex subset V_i is connected;
- Each critical clique in G is entirely contained in a single vertex subset V_i in $\mathcal{P}$.

The importance of class-partitions of a graph G lies in the fact that for each optimal d-partition $\mathcal{S}$ of a graph G, there is a class-partition $\mathcal{P}$ of G such that the vertex subsets in $\mathcal{S}$ can be obtained from the vertex subsets in $\mathcal{P}$ by simple set split or set combination. First we will introduce two important definitions before we present the lemma.

Definition Let G be an input graph, $\mathcal{S} = \{S_1, \ldots, S_d\}$ be an optimal d-partition of the vertices in G.

1. S_i is a cluster in $\mathcal{S}$ such that the induced graph $G[S_i]$ is disconnected, we call S_i a *combining cluster*;
2. S_j is a cluster in $\mathcal{S}$ such that S_j contains a proper subset of some critical clique K, we call S_j a *splitting cluster*.

Combining subsets are obtained by combining some subsets in a class-partition, and Splitting subsets are obtained by splitting some subsets in a class-partition.

Lemma B.2 *$\mathcal{S} = \{C_1, C_2, \ldots, C_d\}$ is an optimal d-partition of the vertices in an input graph G. C_i is a vertex subset in $\mathcal{S}$,*

- *If C_i is a splitting subset, C_i contain a proper subset of a critical clique K, $|C_i \bigcap N(K)| > |C_i - K - N(K)|$;*
- *If C_i is a combining subset, all critical cliques intersecting with C_i are entirely contained in C_i.*

Proof. We prove the lemma by contradiction. Suppose that $|C_i \bigcap N(K)| \leq |C_i - K - N(K)|$. C_i contains a proper subset of a critical clique K, by lemma B.1, there is another subset $C_p \in \mathcal{S}$, s.t. $C_p \subset K$.

To make C_i and C_p disjoint cliques, edges between $C_i \bigcap K$ and $C_i - K - N(K)$ are inserted, and edges between $C_i \bigcap K$ and C_p are deleted, the number of edge operations is

$$|C_i \bigcap K| * |C_i - K - N(K)| + (|C_i \bigcap N(K)| + |C_i \bigcap K|) * |C_p| + M \quad (4)$$

Similarly we concern the edge operations on K or related to K and M denotes the number of other edge operations.

Let $C'_p = C_p \bigcup (C_i \bigcap K)$ and $C'_i = C_i - K$. To make C'_p and C'_i disjoint cliques, edges between $C_i \bigcap K$, C_p and $C_i \bigcap N(K)$ are removed, the number of edge operations is

$$|C_i \bigcap N(K)| * (|C_i \bigcap K| + |C_p|) + M \quad (5)$$

many edge operations are applied.

$$(4) - (5) = |C_i \bigcap K| * |C_p| + |C_i \bigcap K| * (|C_i - K - N(K)| - |C_i \bigcap N(K)|)$$
$$> 0$$

Replacing C_i and C_p by C_i' and C_p', we obtain a new d-partition $\mathcal{S}'$ of G, the solution derived from $\mathcal{S}'$ contains Less edge operations, it contradicts the optimality of $\mathcal{S}$.

We can prove part (ii) by contradiction too. C_i is a combining subset, and contains a proper subset of a critical clique K. By lemma B.1, there is another vertex subset $C_p \in \mathcal{S}$ with $C_p \subset K$.

C_i is a combining subset, the induced graph $G[C_i]$ is disconnected, w.l.o.g. $G[C_i]$ contains two connected components, $G[X]$ and $G[Y]$ respectively, X contains the proper subset of K.

Since C_i contains a proper subset of K, by proof of part (i), $|C_i \bigcap N(K)| > |C_i - K - N(K)| > |X - K - N(K)|$, the last inequality holds since vertices in Y are not adjacent to K, $C_i - K - N(K) = (X - K - N(K)) \bigcup Y$. To make $X \bigcup C_p$ and Y disjoint cliques, edges between $C_i \bigcap K$, C_p and $X - K - N(K)$ are inserted. The number of edge operations is

$$(|C_i \bigcap K| + |C_p|) * |X - K - N(K)| + M. (6)$$

Similarly we only concern edge operations on K or related to K, and M denotes the number of other edge operations.

On the other hand, to make C_i and C_p disjoint cliques, edges between C_p and $C_i \bigcap (K \bigcup N(K))$ are removed, and edges between $C_i \bigcap K$ and $X - K - N(K)$, Y are inserted, the number of edge operations is

$$|C_p| * (|C_i \bigcap K| + |C_i \bigcap N(K)|) + |C_i \bigcap K| * (|X - K - N(K)| + |Y|) + M. (7)$$

$(6) - (7) < 0$, since $|C_i \bigcap N(K)| > |X - K - N(K)|$. So replacing C_i and C_p by C_i' and

C'_p, we obtain a new d-partition $\mathcal{S}'$, and the solution derived from $\mathcal{S}'$ consists of less edge operations than $\mathcal{S}$, it is a contradiction. □

In the following lemma, we show the relation between an optimal d-partition and class-partition.

Lemma B.3 *Let $\mathcal{S} = \{C_1, C_2, \ldots, C_d\}$ be an optimal d-partition of the vertices in a graph G, there is a class-partition $\mathcal{P} = \{V_1, \ldots V_h\}$ of vertices in G such that:*

1. *if $h < d$, then each vertex subset C_i in $\mathcal{S}$ is entirely contained in a vertex subset V_j in $\mathcal{P}$;*
2. *if $h \geq d$, then each vertex subset C_i in $\mathcal{S}$ is a union of some vertex subsets in $\mathcal{P}$.*

To construct disjoint cliques induced by $\mathcal{P}$, less edge operations are applied than to construct disjoint cliques induced by $\mathcal{S}$.

PROOF. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_d\}$ be an optimal d-partition, we can prove that $\mathcal{S}$ can contain either combining subsets, or splitting subsets, but not both, i.e. there do not exist two clusters S_i and S_j in $\mathcal{S}$ such that S_i is a splitting cluster and S_j is a combining cluster.

S_i contains a proper subset of a critical clique K, by lemma B.1, there is a cluster $S_p \in \mathcal{S}$ with $S_p \subset K$. And w.l.o.g. suppose that there are two connected components $G[X]$ and $G[Y]$ in $G[S_j]$, $X \bigcup Y = S_j$.

Replacing S_i, S_p and S_j by $S_i \bigcup S_p$, X and Y, we obtain a new d-partition $\mathcal{S}'$ of vertices in G. By lemma B.2, to construct critical cliques induced by $S_i \bigcup S_p$ is less costly than S_i and S_p. Also since $G[X]$ and $G[Y]$ are not connected, we save the edge insertion operations to connect disjoint cliques induced by X and Y. To construct critical cliques induced by X and Y is less costly than cluster induced by $S_j = X \bigcup Y$. $\mathcal{S}'$ is a better d-partition than $\mathcal{S}$, it contradicts the optimality of $\mathcal{S}$.

If $\mathcal{S}$ contains combining clusters, we can construct a class partition $\mathcal{P}$ by replacing each combining cluster by the vertex subsets of the connected components in the induced graph on the cluster. To construct disjoint cliques induced by $\mathcal{P}$, we save many edge insertion operations. Less edge operations are applied to construct disjoint cliques induced by $\mathcal{P}$ than $\mathcal{S}$.

If $\mathcal{S}$ contains splitting subsets, for a cluster S_i containing a proper subset of some critical clique K, by lemma B.1, other clusters are either subsets of K, or do not contain vertices in K. We merge the cluster which are subsets of K with S_i and obtain a new vertex subset. By lemma B.2, it is less costly to construct the disjoint clique induced by the new vertex subset. We repeat the process for all splitting clusters and critical cliques, and obtain a class-partition $\mathcal{P}$, to construct disjoint cliques induced by $\mathcal{P}$ is less costly than $\mathcal{S}$. □

The Lemma B.3 has double meaning. If the optimal d-partition contains only combining clusters, no critical clique is split. We could compute the optimal solution based on the critical clique graph G_c. On the other hand, if the optimal d-partition contains only splitting subsets, we can apply the reduction rules on G to reduce the size of G. Combining both cases, a kernel is derived.

C. A kernel of size $7k + 2d$

A *kernelization* for a NP-Hard problem is a "preprocessing" on the instances of the problem to reduce the instances' sizes significantly. In particular, the size of the resulting instances can be bounded by a function of the parameter. In practice, parameters have moderate values, so we could develop practically efficient algorithms for the small instances. Kernelization is an important contribution of parameterized complexity and computation.

In the section we present a kernel of the d-Cluster Editing problem. The kernel has two parts: one part is a weighted graph and the other is unweighted, so that if G admits a

solution of size k if and only if there is a solution of weight k to one of the instances. The size of kernel can be bounded by a linear function of k and d.

As shown in lemma B.3, there are two cases for the optimal d-partition $\mathcal{S}$ for G, the first case is that $\mathcal{S}$ contains combining clusters, the second is that $\mathcal{S}$ contains splitting clusters. The first part of the kernel corresponds to the first case, where $\mathcal{S}$ contains combining clusters. By lemma B.3, any critical clique K is entirely contained in some cluster. It inspires us that the first part of kernel is simply the critical clique graph G_c of G, which is weighted on both vertices and edges. A node in G_c corresponds to a critical clique in G, its weight is the size of the critical clique. And the weight of an edge is simply the product of the weights of the two endpoints. The following lemma provides an upper bound on the size of G_c.

Given a solution for an input graph G, we call a vertex v *touched* if we delete or insert an edge adjacent to v, otherwise we call it *untouched*. Similarly a critical clique is *touched* if we delete or insert edges adjacent to it.

Lemma C.1 *The size of the critical clique graph G_c can be bounded by $2k + d$.*

Proof. Suppose $\mathcal{S}$ is an optimal d-partition for G, an optimal solution can be easily derived from $\mathcal{S}$. Each cluster in $\mathcal{S}$ contains at most one untouched critical clique as a subset: to make the union of two critical cliques a disjoint cliques, one must insert/delete edges adjacent to one of them, for they have at least one different neighbor besides themselves. Thus the total number of untouched critical cliques is at most d.

Each edge operation connects/disconnects at most two critical cliques; and a touched critical clique is adjacent to at least one inserted/deleted edge. So there are at most $2k$ touched critical cliques. Overall at most $2k + d$ many critical cliques are in $\mathcal{S}$, so the size of G_c could be bounded by $2k + d$. □

In the second case, $\mathcal{S}$ contains splitting clusters. By lemma B.3, the induced graph

$G[S_i]$ on any cluster S_i is connected. We apply reduction rules to G to reduce its size. Recall that in chapter II, we present the definition of the *editing degree* of a vertex, roughly speaking, For a vertex v and a critical clique K, $v \in N(K)$, the editing degree $p_K(v)$ of v respect to K is the number of inserted/deleted edges adjacent to v to make $K \bigcup N(K)$ a disjoint clique.

Reduction Rules

Rule 1 If there is a critical clique K of G with $|K| > k$, we make $K \bigcup N(K)$ a disjoint clique and reduce k accordingly.

Rule 2 K is a critical clique with $|K| \geq |N(K)|$ and $|K| + |N(K)| > \sum_{v \in N(K)} p_K(v)$, delete edge between $N(K)$ and $N_2(K)$, reduce k accordingly.

Rule 3 Let $\mathcal{C} = \{C_1, C_2, \ldots C_h\}$ be the collection of isolated cliques in G, $h \leq d$, the cliques are sorted by the size. Remove cliques $\{C_{j+1}, \ldots, C_h\}$ from G where $\sum_{i=1}^{j} |C_i| \geq d$, but $\sum_{i=1}^{j-1} |C_i| < d$, reduce d by $h - j$.

Corollary C.2 *Rule 1 is safe.*

Proof. The correctness of Rule 1 is obvious, since $|K| > k$, to connect a vertex not in $N(K)$ with K, or remove a vertex in $N(K)$ from K, at least $k + 1$ edge insertion/deletion operations are applied. It can not make a solution, so Rule 1 is safe. □

Lemma C.3 *Given an input graph G, suppose $\mathcal{S}$ is an optimal d-partition, which contains only splitting clusters. $\mathcal{P}$ is the class-partition obtained by lemma B.3; For all critical cliques K with $|K| \geq |N(K)|$ and $|K| + |N(K)| > \sum_{v \in N(K)} p_K(v)$, $K \bigcup N(K)$ make a vertex subset in $\mathcal{P}$.*

PROOF. We can prove it by contradiction, suppose $\mathcal{P}$ contains two subsets V_i and V_j, and K a subset of V_i, $V_i \bigcup N(K)$ and $V_j \bigcup N(K)$ are non-empty proper subsets of $N(K)$ and $V_i - K - N(K) \neq \emptyset$, w.l.o.g. $N(K) \subset V_i \bigcup V_j$. It has been proved in chapter II, comparing the number of edge operations to make the subsets $K \bigcup N(K)$, $V_i - K - N(K)$ and $V_j - N(K)$ disjoint cliques and to make V_i and V_j disjoint cliques, the former needs less number of edge operations.

We could construct a new class-partition $\mathcal{P}'$ by replacing V_i and V_j by $K \bigcup N(K)$, $V_i - K - N(K)$ and $V_j - N(K)$. By splitting the same vertex subsets we obtain a d-partition $\mathcal{S}'$ from $\mathcal{P}'$. obviously the solution derived from $\mathcal{S}'$ includes less edge operations than $\mathcal{S}$, it is a contradiction. □

Thus to apply reduction, we remove the critical cliques K with $|K| \geq |N(K)|$ and $|K| + |N(K)| > \sum_{v \in N(K)} p_K(v)$. To obtain the optimal d-partition, vertex subsets in the class-partition are split. We could bound the size of vertex subsets to be split by Rule 3.

Corollary C.4 *Rule 3 is safe.*

PROOF. $\mathcal{C}$ is a collection of isolated cliques in G. To transfer G to a collection of d cliques, it may be necessary to split some disjoint cliques in G to generate exact d cliques.

The greedy approach is the following: pick the smallest disjoint clique and remove one vertex from it, repeat the process, until we obtain d disjoint cliques. Obviously the greedy approach works for disjoint cliques, and at most j cliques is split with $\sum_{i=1}^{j} |C_i| \geq d$ to generate exact d cliques. Rule 3 is correct. □

By reduction rule 1, all critical cliques K with $|K| > k$ are removed. So $|C_i| \leq k \ \forall i$. Since $\sum_{i=1}^{j-1} |C_i| < d$, $\sum_{i=1}^{j} |C_i| < d + k$.

Lemma C.5 *By applying reduction rule 1, 2 and 3, the size of the resulting graph is reduced to* $5k + d$.

PROOF. By applying reduction 1, 2 and 3, the resulting graph consists of a collection of isolated cliques and a subgraph with no critical clique K with $|K| > k$, or $|K| \geq |N(K)|$ and $|K| + |N(K)| > \sum_{v \in N(K)} p_K(v)$, by the corollary above, the size of the isolated cliques is bounded by $k + d$; We will show that the size of subgraph G' can be bounded by $4k$.

after applying an optimal solution, G' is transferred to h disjoint cliques where $h \leq d$. Let $\mathcal{S} = \{S_1, S_2, \ldots S_h\}$ be the vertex set of the disjoint cliques. In $\mathcal{S}$, similarly vertices is $\mathcal{S}$ are either touched or untouched. Since one edge operation can "touch" two vertices, and at most k edge operations are applied, the number of touched vertices is at most $2k$.

K is an untouched critical clique and $K \subseteq S_i$, as shows in chapter II, $S_i = K \bigcup N(K)$. There are two cases: if $|K| \geq |N(K)|$, by reduction rule 2, $|K| \leq \sum_{v \in N(K)} p_K(v) - |N(K)| < \sum_{v \in N(K)} p_K(v)$; If $|K| < |N(K)|$, by the definition of $p_K(v)$, $p_K(v) \geq 1 \ \forall v \in N(K)$, $|K| < |N(K)| \leq \sum_{v \in N(K)} p_K(v)$. In both cases, the size of the size of untouched clique K is bounded by $\sum_{v \in N(K)} p_K(v)$. Since at most k edge operations are applied, $\sum_{v \in N(K), K is untouched} p_K(v)$ is bounded by $2k$, so is the size of untouched vertices. Put it all together, the resulting graph contains at most $5k + d$ vertices. □

Theorem C.6 *The* d-CLUSTER EDITING *problem admits a kernel of size at most* $7k + 2d$.

D. A FPT algorithm for the d-CLUSTER EDITING problem

Branch-and-search approach is a powerful tool to design exact algorithms. In the section, we propose a fixed-parameter tractable algorithm for the d-CLUSTER EDITING problem based on this approach.

1. The branch-and-search algorithm

In the following we present the key lemma for the branch-and-search algorithm.

Lemma D.1 *G is a connected graph, and for any pair of vertices u and v in G, there is at most one vertex w, so that w is connected to either of u and v, but not both. G is a clique, or almost a clique with at most one missing edge.*

PROOF. Suppose u, v and w are vertices in G, w is adjacent to u, but not v. We will show that G is almost a clique with one missing edge $(w,\ v)$.

w must connect to all common neighbors of u and v. Otherwise suppose there is a vertex x which is adjacent to u and v, but not adjacent to w. There are two possible cases: if u connects to v, consider the pair $(u,\ w)$, u is adjacent to v and x, but w is adjacent to neither, it is not possible; If u does not connect to v, considering the pair $(w,\ v)$, w is adjacent to u, v is not, and v is adjacent to x, but w not, it is not possible either. Thus w is adjacent to all common neighbors of u and v.

u and v must be connected. Otherwise suppose u and v are not connected, let x be a common neighbor of u and v. Considering the pair $(x,\ v)$, x is adjacent to u and w, but v is adjacent to neither, it is not possible. Thus u and v are connected.

All the common neighbors of u and v must be connected. Otherwise suppose x and y are two common neighbors of u and v, and x and y are not connected. Considering the pair $(x,\ v)$, x is adjacent to w, v is not; v is adjacent to y, but x not. It is not possible. Thus all common neighbors of u and v are connected.

And there is no vertex which is adjacent to neither u nor v. Otherwise, suppose there is a vertex z which is adjacent to neither u or v. There are two possible cases: If z is connected to w, for the pair $(w,\ u)$, w is connected to z, but u not, u is connected to v but w not. It is not possible. if z is not connected to w, then for the pair $(u,\ z)$, u is connected to w and v, but z is connected to neither, it is not possible either.

Overall the graph G would be a clique with a missing edge (w, v). It finishes the proof.

□

In the following, we present the branch-and-search algorithm. For a four-tuple $\{u, v, w, z\}$, w and z are adjacent to either u or v, but not both. There are two cases: $(u,\ v) \in E$, $(v,\ w) \in E$ and $(u,\ w) \notin E$, and $(v,\ z) \notin E$ and $(u,\ z) \in E$, we call $\{u, v, w, z\}$ **type I** four-tuple; Or $(u,\ v) \in E$, $(v,\ w) \in E$ and $(v,\ z) \in E$, and $(u,\ w) \notin E$ and $(u,\ z) \notin E$, we call $\{u, v, w, z\}$ **type II** four-tuple. In the Figure 3 and Figure 4, we illustrate the type I and II four-tuples and possible ways to eliminate them.

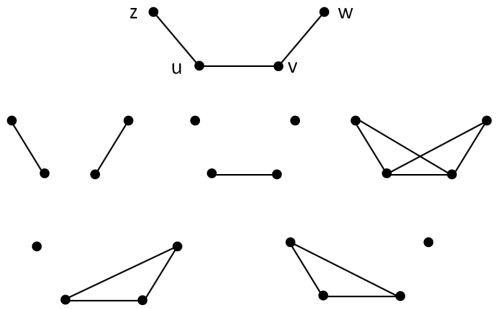

Fig. 3. Type-I four-tuple and possible branches

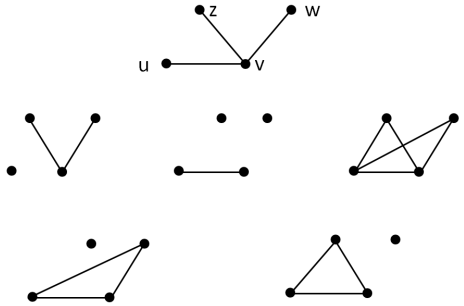

Fig. 4. Type-II four-tuple and possible branches

By lemma D.1, if no such four-tuple exists in G, each connected component in G is almost a clique with at most one missing edge.

We list possible way to eliminate four-tuples of type I in the branch-and-search algorithm, which is shown in Figure 5 four types of type II can be eliminated in the similar way, we omit the details.

Theorem D.2 *The branch-and-search algorithm can enumerate all possible class-partitions of G.*

Proof. By lemma D.1, every connected component of G contain no such four-tuple

Algorithm **BranchAlgo**

INPUT: *a graph $G = (V, E)$, A is a set of edges;*
OUTPUT: *a collection of vertex partitions of G.*

1. If $\sum_{e \in A} \omega_e > k$, where ω_e is the weight of e, return "No solution";
2. In all connected components in G, if there is no four-tuple (u, v, w, z) of type I and II
   2.1 Insert the missing edges in each connected components in G, add the edge to A; If $\sum_{e \in A} \omega_e > k$, return "No solution";
   2.2 $\mathcal{P}$ is the the partition derived from G, where vertices in each connected component make a vertex subset in $\mathcal{P}$, return $\mathcal{P}$;
3. If there is a four-tuple (u, v, w, z) of type I
   3.1 recursively call BranchAlgo$(G - (u,\ v), A \bigcup \{(u,\ v)\})$, let the returned collection be $\mathcal{L}_1$;
   3.2 recursively call BranchAlgo$(G - (v,\ w) - (u,\ z), A \bigcup \{(v,\ w), (u,\ z)\})$, let the returned collection be $\mathcal{L}_2$;
   3.3 recursively call BranchAlgo$(G + (u,\ w) + (v,\ z), A \bigcup \{(u,\ w), (v,\ z)\})$, let the returned collection be $\mathcal{L}_3$;
   3.4 recursively call BranchAlgo$(G - (u,\ z) + (u,\ w), A \bigcup \{(u,\ z), (u,\ w)\})$, let the returned collection be $\mathcal{L}_4$;
   3.5 recursively call BranchAlgo$(G - (v,\ w) + (v,\ z), A \bigcup \{(v,\ w), (v,\ z)\}$, let the returned collection be $\mathcal{L}_5$.
4. Return the union of the collections of partitions.

Fig. 5. The branch-and-search algorithm to list class-partitions.

$\{u, v, w, z\}$ of type I and II, G consists of a collection of disjoint cliques, each clique misses at most one edge. Insert the missing edges to G, G is a collection of disjoint cliques.

$\{u, v, w, z\}$ is a four-tuple of type I, if u and v are in different disjoint cliques in the final graph, edge $(u,\ v)$ will be removed from G. If u and v are in the same disjoint clique, there are four subcases,

(i) w and z are in the same cluster with u and v, edges (u, w) and (v, z) are inserted to G;

(ii) w and z are in different cluster than u and v, edges (u, z), (v, w) are removed from G;

(iii) w is in the same cluster with u and v, but z is not, edge (u, z) is removed and edge (u, w) is inserted;

(iv) z is in the same cluster with u and v, but w is not, edge (v, w) is removed, edge (v, z) is inserted.

In the branch-and-search algorithm, there are five branches, each branch corresponds to a case above, we don't miss a possible partition. □

The running time of the branch algorithm can be bounded by the function $T(k) = T(k - 1) + 4T(k - 2) + O(n)$, since in case 1, k is reduced by 1; And in case 2, 3, 4, 5, k is reduced by 2. Solving it, the running time function is $O^*(2.56^{k_1})$, where k_1 is the number of edge operations applied in branch-and-search algorithm.

For each class-partition $\mathcal{P}$, if $\mathcal{P}$ contains less than d vertex subsets, we apply the *splitting algorithm* to split subsets in $\mathcal{P}$; Otherwise we apply the *combining algorithm* to combine subsets in $\mathcal{P}$.

2. The splitting algorithm

The input of the splitting algorithm is a graph G containing h disjoint clique with $h < d$, k is the parameter. The vertex set of each clique in G makes a vertex subset in a partition

$\mathcal{P}$. The output of the algorithm is a graph with d disjoint cliques. The greedy algorithm is present in the following:

1. Pick the smallest cluster with at least two vertices, remove one vertex from the cluster, reduce k accordingly. Tie are broken arbitrarily;

2. Repeat step 1, until either we obtain d clusters, return "Yes", or $k \leq 0$, return "No solution".

Before proving the correctness of the splitting algorithm, we present a useful lemma.

Lemma D.3 *Given a graph G, G contains h disjoint cliques with $h < d$. Applying least number of edge deletion operations, G is converted to a new graph G' containing d disjoint cliques. There is at most one clique with at most two vertices in G', which is a proper subset of a clique in G.*

PROOF. We could prove it by contradiction. Suppose there are two cliques C_1 and C_2 in G' which are proper subsets of cliques in G, and $|C_1| \geq 2$, $|C_2| \geq 2$. W.l.o.g. we assume that $|C_1| \geq |C_2|$.

By undoing the edge deletion operation to remove one vertex from C_1, and remove one vertex from C_2, we obtain a new graph G''. The difference between edge operations to convert G to G' and edge operations to convert G to G'' is that the former contains $|C_1|$ edge deletion operation to remove one vertex from C_1, and the later contains $|C_2| - 1$ edge deletion operation to remove one vertex from C_2. $|C_1| > |C_2| - 1$, it contradicts the fact that G' is obtained by deleted least number of edges in G. □

Based on the lemma above, we prove that the greedy approach is correct.

Theorem D.4 *The splitting algorithm is correct.*

PROOF. G contains h disjoint cliques, denoted by $\{C_1, C_2, \ldots C_h\}$. The cliques are sorted by the size.

G' contains d cliques and is obtained by deleting the least number of edges from G, C_i is the smallest cluster which is not modified in G'. If C_i is also not modified by the splitting algorithm, G' is same graph returned by the splitting algorithm. Since all cliques C_j with $j < i$ are split in G', by lemma D.3, all except one cliques with index smaller than i are completely split. On the other hand the splitting algorithm split cliques C_j with $j < i$ to obtain d cliques, so $\sum j < i|C_j| \geq d$. Splitting cliques with $j < i$ already producing d cliques, so cliques C_t with $t > i$ are not split in G'. G' is the same graph returned by the splitting algorithm.

If C_i is modified by the splitting algorithm, p vertices are removed from C_i where $p \leq |C_i| - 1$, clusters C_x with $x < i$ are completely split by the splitting algorithm as well, $\sum_{x<i} |C_x| < d$. On the other hand, there must be a cluster C_j with $j > i$, and at least p vertices are removed from C_j in G'. The different edge deletion operations between ones returned by the splitting algorithm and ones to convert G to G' are that, the former remover p vertices from C_i and the later remove p vertices from C_j. Since $j > i$, it takes more edge deletion operations to remove p vertices from C_j from C_i.

In both cases, we could prove that the edges deleted by the splitting algorithm is no more than the number of edge deletion operations to convert G to G', G' is obtained by deleting the least number of edges from G, so the splitting algorithm is correct. □

3. The combining algorithm

In the following, we present the combining algorithm. The input of the combining algorithm is a graph G containing h disjoint cliques with $h > d$, and an integer k, we insert at most k

edges to convert G to a union of d critical cliques.

Since at most k edges can be inserted, applying one insertion operation, the number of clusters is reduced by at most one. If $h > d + k$, there is no solution for G. We can assume that $h \leq d + k$.

We can show that the it is a NP-Hard problem that given h cliques, insert least number of edges to convert them to d cliques, this can be proved by reducing the PARTITION problem to it. We present a $O^*(2^h)$ algorithm in Figure 6

D is a table with 2^h rows and d columns, where S contains a collection of clusters out of the h clusters, $D(S, t)$ is the minimum set of edges inserted to transfer S to t clusters; If S contains less than t clusters, C is empty.

Algorithm **CombiningAlgo**

INPUT: *G: a collection of h clusters with* $d < h \leq d + k$*;*
OUTPUT: *d clusters*

1. $D(S, 1)$ is the number of edges inserted to make S a complete graph.
2. **for** i =2 to d, do
   For all S, $S \subseteq G$ do
   $D(S, i) = \min_{S' \subseteq S}\{D(S', 1) + D(S - S', i - 1)\}$.
3. return $D(T, d)$.

Fig. 6. Combining algorithm to merge cliques

Corollary D.5 *The combining algorithm computes the least number of edge insertion operation to convert G to d cliques.*

To transfer G to d clusters, we insert edges to convert a set S' of cliques to a clique, and convert the rest graph to $d - 1$ cliques. We enumerate all possible sets S' to compute the optimal solution. The combining algorithm is correct.

The running time of combining algorithm is $O(\sum_i \binom{h}{i} * 2^i) = O(3^h)$, and it can be further improved by applying Mobius transformation [12].

Lemma D.6 *By applying Mobius transformation, the running time of combining algorithm is reduced to* $O^*(2^{d+k})$

PROOF. We compute the table column by column. Since for any $S \subseteq G$, $D(S,1)$ can be computed in time $O^*(2^h)$, in the following, we compute $D(S,t)$ for $t = 2, \ldots, d$.

Define function $f(S) = D(S,1)$ and $g(S) = D(S,t-1)$, so

$$D(S,t) = (f * g)(S) = \min_{\substack{U,V \subseteq S \\ U \bigcup V = S \\ U \bigcap V = \emptyset}} f(U) + g(V)$$

It was shown [12] that the subset convolution over the integer min-sum semiring can be computed in $O^*(2^n M)$, where n is the number of elements and M is the maximum possible absolute value of input functions.

By the definition of $(f * g)(S)$, there are h elements and the maximum value of f, g is bounded by k. So $D(S,t)$, $\forall S \subseteq G$ can be computed in time $O^*(2^h k)$, and $D(G,d)$ can be computed in time $O^*(2^h k * d)$. □

Overall, the running of the algorithm is bounded by $O^*(2.56^{k_1} * 2^{d+k-k_1})$. Thus the worst case analysis shows the algorithm takes time at most $O^*(\max\{2.56^k, 2^{k+d}\})$.

E. Final remarks

We discuss the parameterized complexity of a variance of the CLUSTER EDITING problem, – the d-CLUSTER EDITING problem, which requires that the resulting graph contains exactly d disjoint clusters. We introduce a new concept, *class-partitions* of a graph, and shows that the optimal solution can be obtained by split or combination of classes in a class-partition.

We prove that the optimal solution can be obtained by either splitting classes in a class-partition, or combining classes. We construct a kernel consisting of two graphs, either by splitting classes of a class-partition of one graph, or by combining classes of a class-partition of the other graph, we obtain the optimal solution. Totally the kernel contain no more than $7k + 2d$ vertices.

We also develop a branch-and-search algorithm to enumerate all possible class-partitions of the kernel; For the first graph, we apply the splitting-algorithm to split classes to obtain exactly d clusters; For the second graph, we apply the combining-algorithm to combining classes to reduce the number of clusters to exactly d. The optimal solution can be the best of two kinds of solutions. Putting all together, the runtime of the algorithm is $O^*(2.56^k, 2^{k+d})$.

CHAPTER IV

A QUADRATIC KERNEL FOR THE PSEUDO-ACHROMATIC NUMBER PROBLEM

We study the parameterized complexity of the pseudo-achromatic number problem in this chapter, the problems is defined as following: Given an undirected graph and a parameter k, determine if the graph can be partitioned into k groups such that every two groups are connected by at least one edge.

This problem has been extensively studied in graph theory and combinatorial optimization. We show that the problem has a kernel of at most $(k-2)(k+1)$ vertices that is constructable in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in the graph, and k is the parameter. This directly implies that the problem is fixed-parameter tractable. We also study generalizations of the problem and show that they are parameterized intractable.

A. Introduction

The PSEUDO-ACHROMATIC NUMBER problem is to determine whether an undirected graph G can be partitioned into k groups/classes $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ such that every two groups $\mathcal{G}_i$ and $\mathcal{G}_j$, $1 \leq i < j \leq k$, are connected by at least one edge. The problem is also referred to in the literature as the GRAPH COMPLETE PARTITION problem, and is formally defined as follows:

---

*Reprinted with permission from "On the pseudo-achromatic number problem", by J. Chen, I. A. Kanj, J. Meng, X. Ge, F. Zhang, 2009, Theoretical Computer Science, volume 410, issue 8 - 10, pages 818 - 829, Copyright [2009] by Elsevier Limited.

Definition Let G be an undirected graph. The *pseudo-achromatic number* of G is the largest integer p such that there exists a surjective function $f : V(G) \to \{1, \ldots, p\}$ satisfying: for all i, j, where $1 \leq i, j \leq p$ and $i \neq j$, there exist $u \in f^{-1}(i),\ v \in f^{-1}(j)$ such that $(u, v) \in E(G)$, where $f^{-1}(h)$ denotes the preimage set of h under f.

The PSEUDO-ACHROMATIC NUMBER problem is:

> PSEUDO-ACHROMATIC NUMBER. Given an undirected graph G and a positive integer k, determine if the pseudo-achromatic number of G is at least k.

We will be using the informal definition more frequently than the formal one.

It is easy to see that the PSEUDO-ACHROMATIC NUMBER problem is a variation of the graph coloring problem (or the achromatic number problem), the latter problem requiring the groups in the partition to be independent sets, and the number of groups to be as few as possible.

The PSEUDO-ACHROMATIC NUMBER problem was first introduced by Gupta in 1969 [52], and since then it has been studied extensively [7, 8, 9, 17, 37, 66, 77]. The problem is known to be NP-complete even on restricted classes of graphs [9, 37, 66].

Kortsarz et al. [66] studied the approximability of the PSEUDO-ACHROMATIC NUMBER problem. It was proved in [66] that the problem has a randomized polynomial-time approximation algorithm of ratio $O(\sqrt{\lg n})$, which can be de-randomized in polynomial time. This upper bound on the approximation ratio was shown to be asymptotically tight under the randomized model.

The PSEUDO-ACHROMATIC NUMBER problem was also considered from the extremal graph-theoretic point of view on special classes of graphs [8, 17, 77, 86, 87]. Balsubramanian et al. [7] gave a complete characterization of when the pseudo-achromatic number of the join

of two graphs is the sum of the pseudo-achromatic numbers of the two graphs.

In this chapter we study the parameterized complexity of the PSEUDO-ACHROMATIC NUMBER problem. We show that the problem has a kernel of size at most $(k-2)(k+1)$ vertices that is computable in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in the graph. This kernelization result directly gives an algorithm for the PSEUDO-ACHROMATIC NUMBER problem running in time $O(k^{k^2-k+2} + m\sqrt{n})$, thus showing that the problem is fixed-parameter tractable. The upper bound on the kernel size is obtained by developing elegant and highly non-trivial structural results, that are of independent interest.

We also study generalizations of the PSEUDO-ACHROMATIC NUMBER problem and prove that they are parameterized intractable. In particular, we consider the VERTEX GROUPING problem, in which an input instance has the form (G, H, k), where G and H are two graphs, and $k = |V(H)|$. The problem asks for the existence of a surjective function $f : V(G) \longrightarrow V(H)$ satisfying the property that $\forall u, v \in V(H)$, if $(u, v) \in E(H)$ then there exists $x \in f^{-1}(u), y \in f^{-1}(v)$ such that $(x, y) \in E(G)$. The PSEUDO-ACHROMATIC NUMBER problem is a special case of the VERTEX GROUPING problem in which the graph H is the complete graph on k vertices. The VERTEX GROUPING problem falls into the category of grouping problems, where a grouping of the graph G into $|V(H)|$ groups is sought such that the inter-group properties are imposed by the graph H. We prove some (parameterized) intractability results for the VERTEX GROUPING problem. For example, we show that the problem is $W[1]$-hard, even when the graph H is the h-star graph (i.e., $K_{1,h-1}$). We also show that some interesting instances of the VERTEX GROUPING problem can be solved in polynomial time.

B. Preliminaries

We have defined the notion of FPT, W-hierarchy and kernelization in the introduction chapter, the readers are referred to [35] for more details about parameterized complexity theory.

Recall that the parameterized-complexity preserving reduction (FPT-reduction) can be defined as follows: A parameterized problem Q is *FPT-reducible* to a parameterized problem Q' if there exists an algorithm of running time $f(k)|x|^c$ that on an instance (x,k) of Q produces an instance $(x',g(k))$ of Q' such that (x,k) is a yes-instance of Q if and only if $(x',g(k))$ is a yes-instance of Q', where the functions f and g depend only on k, and c is a constant. A parameterized problem Q is $W[i]$-*hard* if every problem in $W[i]$ is FPT-reducible to Q, $i \geq 1$. Many well-known problems have been proved to be $W[1]$-hard including: CLIQUE, INDEPENDENT SET, SET PACKING, DOMINATING SET, HITTING SET and SET COVER.

The fixed-parameter tractability of a problem turns out to be closely related to the notion of the problem having a good data reduction (or preprocessing) algorithm. Recall that it was shown that a parameterized problem is fixed-parameter tractable if and only if it has a kernelization algorithm [34].

For a graph G we denote by $V(G)$ and $E(G)$ the set of vertices and edges of G, respectively. A *matching* M in a graph G is a set of edges such that no two edges in M share an endpoint. A matching M of G is said to be *maximum* if the cardinality of M is maximum over all matchings in G. For a vertex v and a set of vertices Γ in G, we say that v is *connected to* Γ if v is adjacent to some vertex in Γ. Similarly, for two sets of vertices Γ and Γ' in G, we say that Γ is *connected to* Γ' if there exists a vertex in Γ that is connected to Γ'. For a vertex $v \in G$ we denote by $N(v)$ the set of neighbors of v in G. For a set of vertices Γ in G we denote by $N(\Gamma)$ the set of neighbors of all the vertices of Γ in G, i.e., $N(\Gamma) = \bigcup_{v\in\Gamma} N(v)$. We denote by S_h the $(h+1)$-star graph (i.e., $K_{1,h}$). The vertex of degree h in S_h is referred

to as the *root* of the star, and the other h vertices are referred to as the *leaves* of the star. The *size* of the star S_h is the number of vertices in it, which is $h+1$. We say that a graph G contains S_h if S_h is a subgraph (not necessarily induced) of G.

For a background on network flows we refer the reader to [30], or to any standard book on combinatorial optimization.

C. The kernel

In this section we show how to construct a kernel of size (number of vertices) at most $(k-2)(k+1)$ for the parameterized PSEUDO-ACHROMATIC NUMBER problem. We start by presenting some structural results that are essential for the kernelization algorithm, and that are of independent interest on their own.

1. Structural results

The following lemma ascertains that graphs with large matchings have large pseudo-achromatic number.

Lemma C.1 *If a graph G contains a matching of size at least $(k-1)k/2$, then the instance (G,k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

PROOF. Assuming that G contains a matching of at least $(k-1)k/2$ edges, we show how to group the vertices of G into k groups $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ so that every pair of groups is connected. For every pair of groups $(\mathcal{G}_i, \mathcal{G}_j)$ where $1 \leq i < j \leq k$, we use a distinct edge (u,v) of the matching to connect the two groups by mapping the vertex u to $\mathcal{G}_i$ and v to $\mathcal{G}_i$. The remaining vertices of G are mapped arbitrarily to the groups. Since there are exactly $(k-1)k/2$ pairs of groups and at least $(k-1)k/2$ edges in the matching, every pair

of groups is connected under this mapping. It follows that (G, k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem. □

Lemma C.2 *If a graph G contains a set of $k-1$ (mutually) vertex-disjoint stars of sizes $2, \ldots, k$, respectively, then the instance (G, k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

PROOF. Let $\mathcal{S} = \{s_1, \ldots, s_{k-1}\}$ be a set of vertex-disjoint stars in G, where s_i is the star graph S_i. We will map the vertices in $\mathcal{S}$ to k groups $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ such that every pair of groups is connected.

For $i = 1, \ldots, k-1$, we map the root of s_i to group $\mathcal{G}_{i+1}$, and we map its leaves, in a one-to-one fashion, to groups $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_i)$. The remaining vertices in G are mapped arbitrarily to the groups. Since there is no overlap between the vertices of any two stars in S, this mapping is well defined. It is easy to verify now that every two distinct groups in $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ are connected under the defined mapping. It follows that (G, k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem. □

Lemma C.3 *If a graph G contains a collection of (mutually) vertex-disjoint stars each of size at least 2 and at most $k+1$, and such that the total number of vertices in all the stars is more than $(k-2)(k+1)$, then the instance (G, k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

PROOF. Suppose that G contains a collection $\mathcal{P}$ of vertex-disjoint stars, each containing at least two vertices and at most $k+1$ vertices, and such that the total number of vertices of the stars in $\mathcal{P}$ is more than $(k-2)(k+1)$. Assume, to get a contradiction, that (G, k) is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Let s be the star graph S_h and s' be the star graph $S_{h'}$ such that s and s' are vertex-disjoint. By *merging* s and s' we mean creating the star graph $S_{h+h'}$ by identifying the roots of s and s'. Note that the size of the merged star is 1 less than the size of s plus the size of s'.

We construct from $\mathcal{P}$ a sequence of vertex-disjoint stars $\mathcal{S} = \langle s_{k-1}, \ldots, s_r \rangle$, for some integer $r \geq 1$, such that s_i has size at least $i+1$, for $r \leq i \leq k-1$. The procedure that constructs these stars is as follows.

For $i = k-1$ down to 1 do: if the largest star in $\mathcal{P}$ is an S_j, where $j \geq i$, assign it to s_i, and remove it from $\mathcal{P}$; Otherwise, recursively merge the two stars of largest size in $\mathcal{P}$ and add the resulting star to $\mathcal{P}$ until either there is only one star left in $\mathcal{P}$, and in which case the procedure halts, or the largest star in $\mathcal{P}$ is an S_j, where $j \geq i$, and in which case we assign it to s_i, remove it from $\mathcal{P}$, and proceed to the next value of i in the for loop.

If a star s_i in $\mathcal{S}$ was created without merging stars in $\mathcal{P}$, we call s_i a *single star*, otherwise, we call s_i a *merged star*.

Note the following: if s_i is a merged star created from merging a collection of stars, and if s_i is used to produce a valid grouping of G, then clearly the stars that s_i was merged from can replace s_i to produce a valid grouping of G. Therefore, assuming that (G, k) is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem, the last star s_r constructed by the above procedure before halting must satisfy $r \geq 2$. Otherwise, the sequence $\mathcal{S}$ would contain a set of $k-1$ vertex-disjoint stars of sizes $2, \ldots, k$, and by Lemma C.2, the instance (G, k) would be a yes-instance of the problem, contradicting our assumption.

Now assume that the above procedure halts after constructing a sequence of vertex-disjoint stars $\mathcal{S} = \langle s_{k-1}, \ldots, s_r \rangle$, such that s_i has size at least $i+1$, for $2 \leq r \leq i \leq k-1$.

We define a *monotone subsequence* of $\mathcal{S}$ to be a consecutive subsequence $\langle s_i, s_{i-1} \ldots, s_j \rangle$ of $\mathcal{S}$ such that either $s_i, s_{i-1} \ldots, s_j$ are all single stars, or they are all merged stars. A

monotone subsequence $\langle s_i, s_{i-1} \ldots, s_j \rangle$ of $\mathcal{S}$ is *maximal* if it is maximal under containment.

Let $\langle s_i, s_{i-1} \ldots, s_{i-\ell+1} \rangle$, $\ell \geq 1$, be a maximal monotone subsequence of $\mathcal{S}$, and note that $i-\ell+1 \geq 2$ (since $r \geq 2$). We will show that the total number of vertices in the stars of $\mathcal{P}$ that were used to form the subsequence $\langle s_i, s_{i-1} \ldots, s_{i-\ell+1} \rangle$ is at most $2(i+(i-1)+\ldots+(i-\ell+1))$. We distinguish two cases:

- **Case 1.** $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1} \rangle$ consists of single stars. We distinguish two subcases:
  - **Subcase 1.1.** $i = k-1$. Since every single star contains at most $k+1$ vertices by the statement of the lemma, the total number of vertices in the stars in the subsequence is bounded by $\ell(k+1) \leq 2(k-1+k-2+\ldots+k-\ell)$. The last inequality is true because $((k-1)-\ell+1) \geq 2$.
  - **Subcase 1.2.** $i < k-1$. By the maximality of the subsequence, s_{i+1} is a merged star. Since s_i is a single star, it is easy to verify that s_i has size exactly $i+1$. The total number of vertices in the stars in the subsequence is bounded by $\ell(i+1) \leq 2(i+i-1+\ldots+i-\ell+1)$ because $i-\ell+1 \geq 2$.
- **Case 2.** $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1} \rangle$ consists of merged stars. Let s_j be any star in this subsequence, and suppose that s_j was constructed by merging stars $t_1, \ldots, t_q$ in $\mathcal{P}$. By the construction of s_j, the total number of leaves in the stars $t_1, \ldots, t_{q-1}$ is less than j (otherwise these stars would be sufficient to produce s_j), and the size of t_q is not larger than any of the sizes of $t_1, \ldots, t_{q-1}$. Therefore, we have:

$$|t_1| - 1 + |t_2| - 1 + \ldots + |t_{q-1}| - 1 \leq j - 1, \tag{4.1}$$

and

$$|t_q| \leq (|t_1| + |t_2| + \ldots + |t_{q-1}|)/(q-1). \tag{4.2}$$

Combining Inequality (4.1) with Inequality (4.2), and noting that $q \leq j$, we obtain:

$$|t_1| + |t_2| + \ldots + |t_q| \leq 2j. \tag{4.3}$$

Inequality (4.3) shows that the total number of vertices in the stars of $\mathcal{P}$ forming s_j is at most $2j$. By applying this inequality to each star s_j in the maximal monotone subsequence $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1} \rangle$ of merged stars, and by the linearity of addition, we obtain that the total number of vertices of $\mathcal{P}$ used to form the stars in $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1} \rangle$ is at most $2(i + (i-1) + \ldots + (i-\ell+1))$.

It follows from the above that, for any maximal monotone subsequence $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1} \rangle$ of $\mathcal{S}$, the total number of vertices of $\mathcal{P}$ used to form the stars in this subsequence is at most $2(i + (i-1) + \ldots + (i-\ell+1))$. Applying the above bound to every maximal monotone subsequence of $\mathcal{S}$, and by the linearity of addition, we conclude that the total number of vertices in $\mathcal{P}$ forming all the stars in $\mathcal{S}$ is at most $(k-r)(k+r-1)$.

Noting that the number of remaining non-empty stars in $\mathcal{P}$ cannot form an s_{r-1}, the total number of leaves in the remaining stars is at most $r-2$, and consequently, the total number of vertices in the remaining stars is at most $2(r-2)$. Therefore, the total number of vertices in $\mathcal{P}$ is at most $(k-r)(k+r-1) + 2(r-2) = k^2 - k - (r^2 - 3r + 4)$. Since $r \geq 2$, $\mathcal{P}$ has the maximum number of vertices when $r = 2$. It follows that the total number of vertices in $\mathcal{P}$ is at most $(k-2)(k+1)$, contradicting the hypothesis of the lemma.

This completes the proof. □

2. The auxiliary flow network and the graph pseudo-achromatic number

Let G be a graph with pseudo-achromatic number at least k, and let $\mathcal{H}$ be a vertex grouping that partitions the vertices of G into k groups such that every pair of groups is connected.

For each pair of groups in $\mathcal{H}$, pick, arbitrarily, an edge connecting the groups, and

designate that edge as a *critical edge*. Therefore, the set E_c of critical edges consists of exactly $\binom{k}{2} = k(k-1)/2$ edges, each connecting a different pair of groups in $\mathcal{H}$. The tuple $(\mathcal{H}, E_c, k)$ will be called a *valid triple* for the graph G. All the edges in G that are not in E_c are called *noncritical edges*. A vertex in G is *critical* if it is incident to at least one critical edge; otherwise, the vertex is *noncritical*. Note that the existence of the valid triple $(\mathcal{H}, E_c, k)$ for the graph G implies that the pseudo-achromatic number of G is at least k.

Lemma C.4 *Let v be a noncritical vertex in G (with respect to a valid triple $(\mathcal{H}, E_c, k)$). Then either deleting v from G or moving v from its current group to any other group will result in a vertex grouping $\mathcal{H}'$ such that $(\mathcal{H}', E_c, k)$ is a valid triple for the resulting graph.*

PROOF. Since the vertex v is noncritical, v is not incident to any critical edges. Consequently, deleting v from G or moving v from one group to another group will not affect the critical edges. Therefore, in the new vertex grouping $\mathcal{H}'$ in the resulting graph, there are still exactly k groups such that each pair of the groups is connected. □

We will show a nice relationship between the pseudo-achromatic number of a graph and graph matchings.

Let M be a maximum matching in G. Let $I = V(G) \setminus V(M)$, and note that I is an independent set. For a vertex $u \in V(M)$ we denote by $N_I(u)$ the set $N(u) \cap I$. Let M_2 be the set of edges in M whose both ends are connected to I.

Lemma C.5 *Let (u, v) be an edge in M_2. Then $N_I(u) = N_I(v)$ and $|N_I(u)| = 1$.*

PROOF. By definition, both $N_I(u)$ and $N_I(v)$ are nonempty. Therefore, either $N_I(u) \neq N_I(v)$ or $|N_I(u)| > 1$ would imply the existence of two different vertices $w_1 \in N_I(u)$ and $w_2 \in N_I(v)$. However, this would give an augmenting path (w_1, u, v, w_2) with respect to M,

contradicting the maximality of the matching M. □

Let $N_I(M_2)$ be the set $N(V(M_2)) \cap I$, and let $D = I \setminus N_I(M_2)$. We partition the edges of $M \setminus M_2$ into two sets M_1 and M_0, where M_1 consists of all the edges in $M \setminus M_2$ that have exactly one end connected to D, and $M_0 = M \setminus (M_2 \cup M_1)$. Note that the edges in $M_0 \cup M_2$ have no end connected to D (however, an edge in M_0 or in M_1 may have an end connected to $N_I(M_2)$).

The vertices in $V(M_1)$ are further partitioned into R and L, such that R is the set of vertices in $V(M_1)$ that are connected to D, and L is the set of remaining vertices in $V(M_1)$. By definition, each edge in M_1 has exactly one end in R and one end in L. Moreover, by the definition of the set M_0 and by Lemma C.5, the vertices in the set D can only be connected to vertices in R (note that D is an independent set). We refer the reader to Figure 7 for an illustration of the decomposition of G.

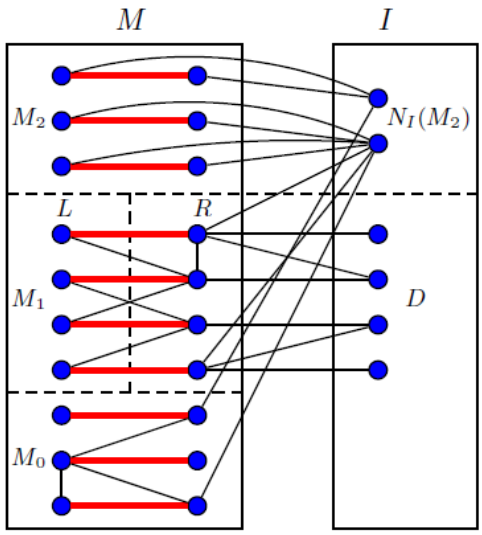

Fig. 7. The decomposition of input G

Let J be the subgraph of G with vertex set $R \cup D$ and edge set $\{(u, v) \mid u \in R \text{ and } v \in D\}$. We construct a flow network J_k from J as follows. Convert each undirected edge (u, v)

in J, where $u \in R$ and $v \in D$, into a directed edge $\langle u, v \rangle$ of capacity 1. Add a source s and a sink t. For each vertex $u \in R$, add a directed edge $\langle s, u \rangle$ of capacity $k-1$; and for each vertex $v \in D$, add a directed edge $\langle v, t \rangle$ of capacity 1. We refer the reader to Figure 8 for an illustration of the flow network J_k.

In the following, we fix a valid triple $(\mathcal{H}, E_c, k)$ for the graph G, a maximum matching M in G, and the corresponding flow network J_k. Let f^* be an integer-valued maximum flow in J_k. In case of no confusion, we will identify the vertices and edges in $J_k - \{s, t\}$ with their counterparts in G. Therefore, an edge is critical and saturated if it is critical with respect to the valid triple $(\mathcal{H}, E_c, k)$ for G and saturated in the flow network J_k under the flow f^*.

For a vertex u, denote by f^*_u the flow through u, i.e., the total outgoing flow from u. We say that a vertex $u \in R$ is *saturated* if $f^*_u = k-1$, and that a vertex $v \in D$ is *saturated* if $f^*_v = 1$.

Let $T_k = \{u \mid u \in D \text{ and } f^*_u = 0\}$. The main result of this subsection is to show that the instance (G, k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem if and only if $(G - T_k, k)$ is.

We further partition the vertices in the set R into two sets R_1 and R_2, where R_1 consists of all saturated vertices (in the flow network J_k under f^*), and $R_2 = R \setminus R_1$.

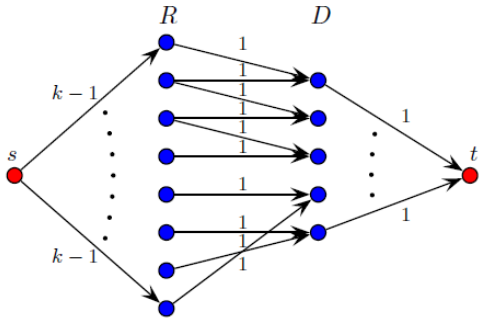

Fig. 8. The flow network J_k.

Lemma C.6 *For each vertex $u \in R_1$, let cri-unsat(u) be the set of critical unsaturated edges going out from u, and let noncri-sat(u) be the set of noncritical saturated edges going out from u. Then there is an injective mapping Φ_u from cri-unsat(u) to noncri-sat(u) (i.e., a mapping Φ_u such that for every $e_1, e_2 \in$ cri-unsat(u), if $e_1 \neq e_2$ then $\Phi_u(e_1) \neq \Phi_u(e_2)$).*

PROOF. It suffices to show that $|\text{cri-unsat}(u)| \leq |\text{noncri-sat}(u)|$. Let $\text{cri}(u)$ be the set of critical edges going out from u, and let $\text{sat}(u)$ be the set of saturated edges going out from u. It is easy to see that the vertex u can be incident to at most $k-1$ critical edges. Thus, $|\text{cri}(u)| \leq k-1$. Moreover, since $u \in R_1$, u is saturated, which gives $|\text{sat}(u)| = k-1$. Now let cri-sat(u) be the set of critical saturated edges going out from u. Then $\text{cri}(u) \setminus \text{cri-sat}(u)$ is the set cri-unsat(u) of critical unsaturated edges going out from u, and $\text{sat}(u) \setminus \text{cri-sat}(u)$ is the set noncri-sat(u) of noncritical saturated edges going out from u. By the above analysis, we have $|\text{cri-unsat}(u)| = |\text{cri}(u) \setminus \text{cri-sat}(u)| \leq |\text{sat}(u) \setminus \text{cri-sat}(u)| = |\text{noncri-sat}(u)|$. □

By Lemma C.6, for each vertex $u \in R_1$ we can correspond an injective mapping Φ_u from the set cri-unsat(u) of critical unsaturated edges going out from u to the set noncri-sat(u) of noncritical saturated edges going out from u.

For the given valid triple $(\mathcal{H}, E_c, k)$, the maximum matching M in G, the flow network J_k, the maximum flow f^* on J_k, and the set of injective mappings $\{\Phi_u \mid u \in R_1\}$, we define a layered structure L that is a subgraph of the flow network J_k, as follows.

Definition The 0-th level of L consists of all vertices in the set T_k. For an integer $i \geq 0$,

(1) the $(2i+1)$-st level of L consists of all vertices $u \in R$ such that $\langle u, v \rangle$ is a critical edge and $v \in D$ is a vertex in the $(2i)$-th level. Every critical edge that is from a vertex in the $(2i+1)$-st level to a vertex in the $(2i)$-th level is also included in L.

(2) the vertices in the $(2i+2)$-nd level are given as follows: for each critical unsaturated

edge $e = \langle u, v\rangle$, where $u \in R_1$ is in the $(2i+1)$-st level and $v \in D$ is in the $(2i)$-th level, if $\Phi_u(e) = \langle u, w\rangle$, then the vertex w is in the $(2i+2)$-nd level, and the edge $\langle u, w\rangle$ is also included in L.

By definition, all vertices in even levels in the layered structure L belong to the set D, and all vertices in odd levels in L belong to the set R. For any integer $i \geq 0$, all edges between the $(2i)$-th level and the $(2i+1)$-st level are critical edges whose direction is from the $(2i+1)$-st level to the $(2i)$-th level; while all edges between the $(2i+1)$-st level and the $(2i+2)$-nd level are noncritical saturated with directions from the $(2i+1)$-st level to the $(2i+2)$-nd level.

Lemma C.7 *The layered structure L has the following properties: (1) all critical edges in L are unsaturated; (2) all vertices in odd levels in L are in the set R_1; and (3) for each vertex v in an even level $2i$, where $i > 0$, there is exactly one edge coming into v from the $(2i-1)$-st level.*

PROOF. (1) Let e be a critical edge in L. If e is a directed edge from the 1-st level to the 0-th level, then the edge e must be unsaturated because all vertices in the 0-th level are in T_k, and hence are unsaturated. If $e = \langle u, v\rangle$ is from the $(2i+1)$-st level to the $(2i)$-th level, for some $i > 0$, then since there is a noncritical saturated edge from the $(2i-1)$-st level to the vertex v in the $(2i)$-th level, and since $v \in D$ has only one out-going edge that has a capacity 1, the critical edge e coming into the vertex v must be unsaturated.

(2) Let v be a vertex in the $(2i+1)$-st level in L, for some $i \geq 0$. By the definition of the layered structure L, $v \in R$, and there is a vertex sequence $(w_0, w_1, \ldots, w_{2i+1})$ in the layered structure L, where $w_0 \in T_k$, $v = w_{2i+1}$, w_j is in the j-th level for all j, and for all h, the edge $\langle w_{2h+1}, w_{2h}\rangle$ is critical (which, by (1) of the current lemma, is also unsaturated), and

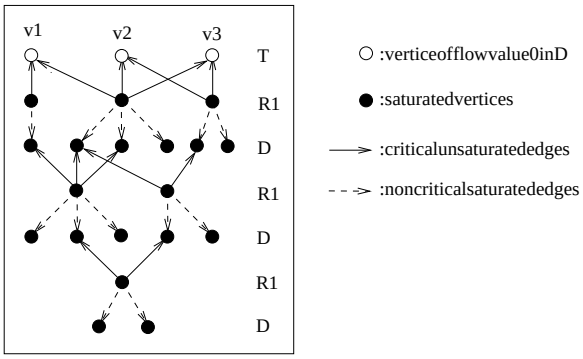

Fig. 9. The layered structure L.

the edge $\langle w_{2h-1}, w_{2h}\rangle$ is noncritical saturated. If the vertex v is not saturated, then the edge $\langle s, w_{2i+1}\rangle$ is unsaturated. Since $w_0 \in T_k$, the edge $\langle w_0, t\rangle$ is also unsaturated. Therefore, the path $(s, w_{2i+1}, w_{2i}, w_{2i-1}, \ldots, w_1, w_0, t)$ would make a flow augmenting path in the residual network of J_k with respect to f^*, contradicting the maximality of the flow f^* in the flow network J_k. This proves that the vertex v must be saturated, i.e., $v \in R_1$.

(3) Let v be a vertex in the $(2i)$-th level in L, for some $i > 0$. By the definition of the layered structure L, $v \in D$, and there is at least one noncritical saturated edge coming into v from the $(2i-1)$-st level. Moreover, since v has only one out-going edge to t that has capacity 1, v cannot have more than one incoming edge from the $(2i-1)$-st level that is saturated. □

By Lemma C.7, all vertices in odd levels in L belong to the set R_1, and for any integer $i \geq 0$, the edges between the $(2i)$-th level and the $(2i+1)$-st level are all critical unsaturated edges. We refer the reader to Figure 9 for the properties of the layered structure L.

We prove next that the layered structure L is finite.

Lemma C.8 *Let v be a vertex in the set D that is at an even level $i > 0$ in the layered structure L. Then v is saturated and does not appear anywhere else in L.*

PROOF. By the definition of the layered structure L, there is a noncritical saturated edge $\langle u, v\rangle$ coming into v from the $(i-1)$-st level. Therefore, the vertex $v \in D$ is saturated.

To prove the second part of the lemma, suppose that the vertex v has two copies v_1 and v_2 in the layered structure L, which appear at the i_1-th level and the i_2-th level, respectively, where $i_1 \leq i_2$ are even integers. Without loss of generality, assume that the index i_1 is the smallest among all vertices in D that have multiple copies in L. We must have $i_1 < i_2$ since each vertex has at most one copy at each level in L. Moreover, $i_1 \neq 0$, since a vertex at the 0-th level is unsaturated (because it is in the set T_k) while a vertex at any other even level is saturated (by the first part of the current lemma). Therefore, we must have $0 < i_1 < i_2$.

By Lemma C.7(3), for $j = 1, 2$, there is a unique noncritical saturated edge $\langle u_j, v_j\rangle$ from the $(i_j - 1)$-st level to the vertex v_j. Since there is at most one saturated edge coming into a vertex in the set D, and $v_1 = v_2$, we must have $\langle u_1, v_1\rangle = \langle u_2, v_2\rangle$, so $u_1 = u_2$. Note that for each $j = 1, 2$, the edge $\langle u_j, v_j\rangle$ is the image of a unique edge $\langle u_j, w_j\rangle$ under the injective mapping Φ_{u_j}, where w_j is in the $(i_j - 2)$-nd level in L. Since $u_1 = u_2$ and $\langle u_1, v_1\rangle = \langle u_2, v_2\rangle$, we must have $\langle u_1, w_1\rangle = \langle u_2, w_2\rangle$. Thus, $w_1 = w_2$ and the vertex w_1 is at the $(i_1 - 2)$-nd level. However, this contradicts the minimality of the index i_1. This completes the proof of the lemma. □

Corollary C.9 *Each edge in the flow network J_k can appear at most once in L.*

PROOF. Two edges between the same pair of adjacent levels in L cannot correspond to the same edge in J_k because no two vertices in the same level of L correspond to the same vertex. Two edges between two different pairs of adjacent levels in L cannot correspond to the same edge in J_k because either they have different flow saturations, or, they either come into or go out from, respectively, two vertices of D at different levels, which by Lemma C.8, must be different. □

By Lemma C.8 and Corollary C.9, we can conclude that the layered structure L is finite. We note that a vertex in the set R_1 may have multiple copies in the layered structure L, which, however, will not affect our discussion.

We call a vertex v at the i-th level of L a *leaf* if there is no edge in L between v and the $(i+1)$-st level in L. In particular, all vertices in the last level of L are leaves.

Lemma C.10 *All leaves in the layered structure L belong to the set D.*

PROOF. Let u be a vertex in the set R_1 that is at the i-th level in L for some i. The vertex u is in L because of a critical unsaturated edge $e = \langle u, v\rangle$, where $v \in D$ is a vertex in the $(i-1)$-st level. By the definition of L, the edge $\Phi_u(e)$ will become an edge from u to a vertex in the $(i+1)$-st level, which implies that the vertex u cannot be a leaf. □

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

Theorem C.11 *The instance (G, k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem if and only if $(G-T_k, k)$ is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

PROOF. Since $G - T_k$ is a subgraph of G, the pseudo-achromatic number of $G - T_k$ cannot be larger than that of G. Therefore, if (G, k) is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem, then $(G - T_k, k)$ is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Now suppose that (G, k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem. Then there is a valid triple $(\mathcal{H}, E_c, k)$ for the graph G. We fix the maximum matching M of G, the flow network J_k, the maximum flow f^* on J_k, the set R_1 of saturated vertices in R, and the set T_k of unsaturated vertices in D, as we have defined in the above discussion.

Since for each valid triple $(\mathcal{H}, E_c, k)$ for the graph G, we can define the set of injective mappings Φ_u and construct the corresponding layered structure L, we can assume, without loss of generality, that $(\mathcal{H}, E_c, k)$ is a valid triple for G together with a set of injective mappings Φ_u, for which the corresponding layered structure L has the minimum number of vertices (note that the layered structure L is finite). We first show that all the vertices in the set T_k are noncritical under this valid triple $(\mathcal{H}, E_c, k)$ and the injective mappings.

If the set T_k contains critical vertices, then the layered structure L has $h_0 + 1 > 1$ levels. Let v_0 be any vertex in the last level (i.e., the h_0-th level) in L. By Lemma C.10, $v_0 \in D$ and $h_0 > 0$ is an even number. By the definition of the layered structure L and since v_0 is a leaf, the vertex v_0 is not incident to any critical edges (recall that D is an independent set, and the vertices in D can only be connected to the vertices in R). Thus, v_0 is a noncritical vertex. Let $e_1 = \langle u_0, v_0 \rangle$ be the unique noncritical edge from the $(h_0 - 1)$-st level to v_0, and let $e_2 = \langle u_0, w_0 \rangle$ be the critical edge in L such that $\Phi_{u_0}(e_2) = e_1$, where w_0 is at the $(h_0 - 2)$-nd level. Suppose that the vertices u_0 and w_0 belong to the groups H_1 and H_2, respectively, under the grouping $\mathcal{H}$. We perform the following operations on the valid triple $(\mathcal{H}, E_c, k)$: (1) move the vertex v_0 from its current group to the group H_2 and let the new grouping be $\mathcal{H}'$; and (2) designate $e_1 = \langle u_0, v_0 \rangle$ the critical edge between the groups H_1 and H_2 (so the edge $e_2 = \langle u_0, w_0 \rangle$ becomes a noncritical edge), and let $E'_c = E_c - e_2 + e_1$. See Figure 10 for an illustration of these operations.

Since v_0 is noncritical, by Lemma C.4, it is easy to see that the triple $(\mathcal{H}', E'_c, k)$ is a valid triple for the graph G. We also modify the injective mapping Φ_{u_0} at u_0 by simply removing the edge e_2 from the domain of Φ_{u_0} (recall that Φ_{u_0} is an injective mapping from the set cri-unsat(u_0) of critical unsaturated edges going out from u_0 to the set noncri-sat(u_0) of noncritical saturated edges going out from u_0): we had $e_2 \in$ cri-unsat(u_0) and $e_1 = \Phi_u(e_2) \in$ noncri-sat(u_0) under the original valid triple $(\mathcal{H}, E_c, k)$, while under the new valid

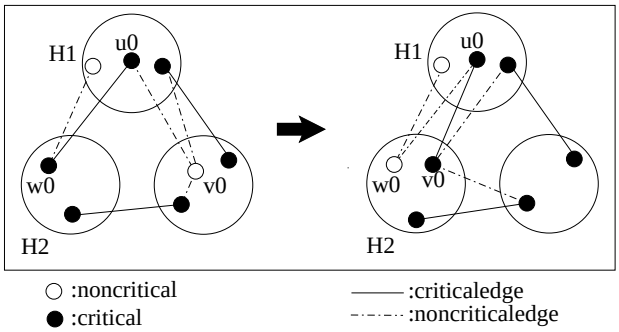

Fig. 10. Moving a noncritical vertex v_0.

triple $(\mathcal{H}', E'_c, k)$, the edge e_1 becomes critical saturated and the edge e_2 becomes noncritical unsaturated, so neither of them is in the set cri-unsat$(u_0) \cup$ noncri-sat(u_0). For all other vertices $u \in R_1$, $u \neq u_0$, in L, we keep the injective mapping Φ_u unchanged.

We consider how the layered structure L is changed under the new valid triple $(\mathcal{H}', E'_c, k)$ and the new injective mapping Φ_{u_0} corresponding to vertex u_0. The layered structure is started from the same set T_k and expanded level by level. An even level is expanded to the next level based on edge saturations and edge criticalities, and an odd level is expanded to the next level based on the injective mapping Φ_u on each vertex u in the current level. Therefore, the layered structure L' under the new valid triple $(\mathcal{H}', E'_c, k)$ and the new injective mapping Φ_{u_0} is exactly the same as the old layered structure L, except when we expand from the vertex w_0 in the $(h_0 - 2)$-nd level to the $(h_0 - 1)$-st level: the edge $e_2 = \langle u_0, w_0 \rangle$ is not included because it is no longer critical. As a consequence, the edge $e_1 = \langle u_0, v_0 \rangle$ will not be added between the $(h_0 - 1)$-st level and the h_0-th level and the vertex v_0 will not appear in the h_0-th level. We emphasize that the above reasoning holds true also because of Corollary C.9, which states that no edge has multiple copies in the layered structure.

Therefore, the layered structure L' under the new valid triple $(\mathcal{H}', E'_c, k)$ and the new injective mapping Φ_{u_0} can be obtained from the layered structure L under the original valid triple $(\mathcal{H}, E_c, k)$ and the original injective mapping Φ_{u_0} by deleting the edges e_1 and e_2 and

deleting the vertex v_0 in the h_0-th level (probably also deleting the vertex u_0 if there is no other critical edge from u_0 to a vertex in the $(h_0 - 2)$-nd level). Thus, L' has at least one fewer vertex than L. However, this contradicts our assumption that the original valid triple $(\mathcal{H}, E_c, k)$, together with the original injective mappings on vertices in R_1, gives the layered structure L of the minimum number of vertices. This contradiction shows that all vertices in the set T_k must be noncritical under the valid triple $(\mathcal{H}, E_c, k)$.

By Lemma C.4, deleting a noncritical vertex in a graph under a valid triple gives a valid triple for the resulting graph. Moreover, note that deleting a noncritical vertex does not convert any noncritical vertices into critical vertices because the critical edges are not changed. Therefore, if we delete all vertices in the set T_k from the graph G under the valid triple $(\mathcal{H}, E_c, k)$, we will obtain a valid triple $(\mathcal{H}', E_c, k)$ for the graph $G - T_k$, which shows that the pseudo-achromatic number of the graph $G - T_k$ is at least k, i.e., $(G - T_k, k)$ is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem.

This completes the proof of the theorem. □

The above theorem shows that the vertex set T_k can be safely removed from the graph G. Moreover, the graph $G - T_k$ has the following nice property.

Lemma C.12 *The vertices in the graph $G' = G - T_k$ can be decomposed into a collection $\mathcal{P}$ of vertex-disjoint stars, each star of size at least 2 and at most $k + 1$.*

PROOF. We will exhibit the collection of vertex-disjoint stars $\mathcal{P}$ in G'. We will denote by $V_{\mathcal{P}}$ the set of vertices of the stars in the collection $\mathcal{P}$, and by $E_{\mathcal{P}}$ the set of edges of the stars in $\mathcal{P}$.

The set of vertices of G' consists of the vertices in the matching M, the vertices in $N_I(M_2)$, and the vertices in D with a non-zero flow value. For a vertex u in R, let $S(u)$ be the star graph formed by the incident edge to u in M_1, together with the set of saturated

edges in G' incident on u. Clearly, each such star $S(u)$ has size at least 2 and at most $k+1$ since the capacity of u in J_k is $k-1$. Moreover, for any two vertices u and v in R, the two star graphs $S(u)$ and $S(v)$ share no vertices; otherwise, there would be a shared vertex $w \in S(u) \cap S(v)$ of capacity 1 in J_k with two saturated edges incident on it, contradicting the flow properties. We add all such stars $S(u)$ to the collection $\mathcal{P}$.

We also include in $\mathcal{P}$ a maximal set of disjoint S_2 stars such that the root of each S_2 star is a vertex in $N_I(M_2)$ and its leaves are the end points of the same edge in M_2. Moreover, for every edge in M_2 whose endpoints are not yet in $V_{\mathcal{P}}$, we include it in $\mathcal{P}$ as an S_1 stars. Finally we include in $\mathcal{P}$ the matching edges in M_0 as S_1 stars.

It is clear that all the stars included in $\mathcal{P}$ are vertex-disjoint, and that each star has size at least 2 and at most $k+1$.

We claim that $V_{\mathcal{P}}$ contains all the vertices of G'. First observe that $V_{\mathcal{P}}$ contains the endpoints of all the edges in M. Second, since every vertex v in $D - T_k$ is incident on a saturated edge in G', v is included in $\mathcal{P}$. Moreover, since by definition every vertex $u \in N_I(M_2)$ forms an S_2 star with two vertices w and v, where (w, v) is an edge in M_2, and since by Lemma C.5 no other vertex in $N_I(M_2)$ can form a star with the vertices w and v, it follows from the construction of $\mathcal{P}$ that $u \in V_{\mathcal{P}}$. Therefore, every vertex u in $N_I(M_2)$ is in $\mathcal{P}$, and V_P contains all the vertices of G' as desired. □

3. Putting it all together: the kernelization algorithm

Consider the decomposition of G defined in Subsection 2, and let M and T_k be as defined in Subsection 2. The kernelization algorithm is given in Figure 11.

Theorem C.13 *Given an instance* (G, k) *of the* PSEUDO-ACHROMATIC NUMBER *problem, the algorithm* **PseudoAchromaticNumberKernel** *either decides the instance* (G, k) *cor-*

Algorithm **PseudoAchromaticNumberKernel**

INPUT: (G, k)
OUTPUT: (G', k')

1. construct a maximum matching M of G;
2. **if** $|M| \geq (k-1)k/2$ **then return** YES;
3. compute the set T_k of vertices as described in Subsection 2; $G' = G - T_k$;
4. **if** $|V(G')| > (k-2)(k+1)$ **then return** YES;
5. **return** $(G', k' = k)$;

Fig. 11. The kernelization algorithm for the PSEUDO-ACHROMATIC NUMBER problem

rectly, or returns an instance (G', k') of the problem such that G' is a subgraph of G, $k' \leq k$, and (G, k) is a yes-instance if and only if (G', k') is. Moreover, the algorithm runs in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in G.

PROOF. If the size of the maximum matching M in G is at least $(k-1)k/2$, then by Lemma C.1, G is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem. Therefore, the algorithm **PseudoAchromaticNumberKernel** makes the right decision in step 2.[2]

By Theorem C.11, (G, k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem if and only if (G', k') is.

It suffices to argue that if $|V(G')| > (k-2)(k+1)$ (note $k' = k$), then (G', k'), and hence (G, k), is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem, and the algorithm makes the right decision in step 4.

By Lemma C.12, the set $V(G')$ can be decomposed into a collection of vertex-disjoint stars $\mathcal{P}$, each star of size at least 2 and at most $k+1$. Since $|V(G')| > (k-2)(k+1)$, it follows

[2]We note that step 2 is not essential to the algorithm and can be omitted. However, since the computation of the maximum matching M is essential to the computation of the set of vertices T_k in step 3, there is no harm in checking the size of the matching M and accepting the instance in case the size is large enough. Moreover, this step makes sense, especially from a practical point of view, as there is no need to carry on further with the computation of a maximum flow, and subsequently of the set of vertices T_k, if the graph contains a large matching and the instance can be accepted.

that the number of vertices in $\mathcal{P}$ is more than $(k-2)(k+1)$. Consequently, $\mathcal{P}$ satisfies the statement of Lemma C.3, and (G', k') is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Finally, to see that the algorithm **PseudoAchromaticNumberKernel** runs in time $O(m\sqrt{n})$, note first that the maximum matching M can be computed in $O(m\sqrt{n})$ time by a standard maximum matching algorithm [30]. Noting that the flow network J_k is a bipartite graph with at most $O(n)$ vertices and $O(m)$ edges, the maximum flow f^* in J_k can be computed in time $O(m\sqrt{n})$ [30]. All other steps can be performed in time $O(m)$, and the theorem follows. □

Corollary C.14 *The* PSEUDO-ACHROMATIC NUMBER *problem has a kernel of at most* $(k-2)(k+1)$ *vertices that is computable in time* $O(m\sqrt{n})$*, where* n *and* m *are the number of vertices and edges, respectively, in the graph, and* k *is the parameter.*

Remark. Note that our upper-bound analysis of the size of the kernel returned by the algorithm **PseudoAchromaticNumberKernel** is tight. This can be seen by considering a graph G that consists of $(k-1)k-2 = (k-2)(k+1)$ vertices which are the endpoints of $(k-1)k/2-1$ edges in a matching. The algorithm **PseudoAchromaticNumberKernel** on input (G, k) will return (G, k) as is, and without any modifications. Clearly, (G, k) is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Using the $(k-2)(k+1)$ upper bound on the kernel size, we can solve the PSEUDO-ACHROMATIC NUMBER problem by enumerating all possible assignments of the vertices in the graph to the k groups, then checking whether any such assignment yields a valid grouping. We have the following corollary:

Corollary C.15 *The* PSEUDO-ACHROMATIC NUMBER *problem can be solved in time* $O(k^{k^2-k+2} + m\sqrt{n})$*, and hence is fixed-parameter tractable, where* n *and* m *are the num-*

ber of vertices and edges, respectively, in the graph.

PROOF. Given an instance (G, k) of the PSEUDO-ACHROMATIC NUMBER problem, where G has n vertices and m edges, we apply the algorithm **PseudoAchromaticNumberKernel** to (G, k). The algorithm runs in $O(m\sqrt{n})$ time and either accepts the instance (G, k) correctly, or returns a kernel (G', k) where G' has at most $(k-2)(k+1)$ vertices. Now if G' can be partitioned into k groups that are mutually connected, then every vertex in G' must belong to one of the k groups. Therefore, there are at most $k^{(k-2)(k+1)}$ ways to partition G' into k groups. For each such partitioning, we can check whether the corresponding groups are mutually connected; this can be done in time $O(k^4)$. If we do not succeed in finding a valid partitioning then clearly the algorithm can reject the instance; otherwise, the algorithm returns a valid partitioning. The total running time of the algorithm is $O(k^4 \cdot k^{(k-2)(k+1)} + m\sqrt{n})$, which is $O(k^{k^2-k+2} + m\sqrt{n})$. □

D. Hardness results for the VERTEX GROUPING problem

Recall from Section A that in the VERTEX GROUPING problem we are given an instance (G, H, k), where G and H are two graphs, and $k = |V(H)|$, and the problem asks for the existence of a surjective function $f : V(G) \longrightarrow V(H)$ satisfying the property that for all $u, v \in V(H)$, if $(u, v) \in E(H)$ then there exist $x \in f^{-1}(u)$ and $y \in f^{-1}(v)$ such that $(x, y) \in E(G)$. The VERTEX GROUPING problem can be defined more intuitively as follows.

Let G be an undirected graph. We define an operation on G, called *vertex grouping*, applied to a subset of vertices S as follows: remove all the vertices in S from G, add a new vertex w, and connect w to all the neighbors of S in $G - S$. The VERTEX GROUPING problem is:

VERTEX GROUPING: Given two graphs G and H, where H is a graph of k vertices,

and k is the parameter, decide if H can be obtained from G by a sequence of vertex grouping operations.

If H in the above definition is the complete graph on k vertices, then the VERTEX GROUPING problem becomes the PSEUDO-ACHROMATIC NUMBER problem, and hence is fixed parameter tractable. The following theorem shows that the VERTEX GROUPING problem is parameterized intractable in general.

Theorem D.1 *The* VERTEX GROUPING *problem is* $W[1]$*-hard.*

PROOF. We reduce the $W[1]$-hard problem INDEPENDENT SET to the VERTEX GROUPING problem.

Let (G, k) be an instance of the INDEPENDENT SET problem. Construct a graph G' by adding a new vertex w to G and connecting w to every vertex in G. Let H be a $(k+1)$-star with root r_H. Define the mapping π that, on an instance (G, k) of INDEPENDENT SET, produces the instance $(G', H, k+1)$ of VERTEX GROUPING. Clearly, the mapping π is computable in polynomial time, and hence π is an FPT-reduction. We show that (G, k) is a yes-instance of INDEPENDENT SET if and only if $(G', H, k+1)$ is a yes-instance of VERTEX GROUPING.

In effect, suppose that (G, k) is a yes-instance of INDEPENDENT SET, and let I be an independent set in G of size k. Consider the function $f : V(G') \longrightarrow V(H)$ that maps the k vertices of I in G' to the k leaves of the star H, in a one-to-one fashion, and maps all other vertices of G' to the root r_H of H. Then it is easy to verify that H is a vertex grouping of G' under the function f.

Conversely, suppose that H is a vertex grouping of G' under a function f. Consider any set of vertices I in G of cardinality k satisfying $f(I) = V(H) \setminus \{r_H\}$. Clearly, such a set I exists by the definition of the vertex grouping. Note that f is a bijection from I to

$V(H) \setminus \{r_H\}$. Now for any two distinct vertices u and v of I, u and v are not adjacent in G, otherwise, by the definition of vertex grouping, $f(u)$ and $f(v)$ would be adjacent in H. It follows that I is an independent set of size k in G. This completes the proof. □

The Exponential Time Hypothesis (ETH) states that many NP-hard problems including 3-SAT, INDEPENDENT SET, and VERTEX COVER, cannot be solved in time $2^{o(n)}$. ETH has become a working hypothesis for many researchers in the area of exact and parameterized algorithms. It was shown in [25] that, unless ETH fails, INDEPENDENT SET cannot be solved in time $n^{o(k)}$. It was also shown in [25] that if a parameterized problem Q is reducible to a parameterized problem Q' by an FPT reduction, called *linear fpt-reduction*, that preserves the order of the parameter and does not increase the size of the instance by more than a polynomial factor, and if Q cannot be solved in time $n^{o(k)}$ then it follows that Q' cannot be solved in time $n^{o(k)}$. Clearly, the reduction from INDEPENDENT SET to VERTEX GROUPING, given in the proof of Theorem D.1, is a linear fpt-reduction. Therefore, we have the following theorem:

Theorem D.2 *Unless ETH fails, the* VERTEX GROUPING *problem cannot be solved in time* $n^{o(k)}$*, where* n *and* k *are the number of vertices in* G *and* H*, respectively.*

Determining the complexity of the GRAPH ISOMORPHISM problem is an outstanding open problem that has been attracting the attention of researchers in theoretical computer science for decades. Although no polynomial time algorithm was developed for the problem, it seems unlikely that the problem is NP-hard [64].

We illustrate a relationship between the GRAPH ISOMORPHISM problem and the VERTEX GROUPING problem. Let G_1 and G_2 be two graphs on n vertices. We are interested in knowing how "similar" G_1 and G_2 are, under the notion of vertex grouping defined above. For this purpose, we introduce the following parameterized problem:

GRAPH STRUCTURAL SIMILARITY: given two graphs G_1 and G_2 on n vertices, and a parameter k, decide if there exists a graph H of k vertices such that both (G_1, H, k) and (G_2, H, k) are yes-instances of the VERTEX GROUPING problem.

Intuitively, the graph structural similarity measures the degree of similarity (i.e., k) between two graphs under the notion of vertex grouping. In particular, if $k = n$, then the GRAPH STRUCTURAL SIMILARITY problem is equivalent to the GRAPH ISOMORPHISM problem. We have the following parameterized intractability result for the GRAPH STRUCTURAL SIMILARITY problem:

Theorem D.3 *The* GRAPH STRUCTURAL SIMILARITY *problem is* $W[1]$*-hard.*

PROOF. As was shown in Theorem D.1, the VERTEX GROUPING problem is $W[1]$-hard when the graph H is a star. An FPT-reduction can be constructed that takes an instance (G, H, k), where G has n vertices and H is a k-star, of the VERTEX GROUPING problem to an instance (G_1, G_2, k) of the GRAPH STRUCTURAL SIMILARITY problem, where $G_1 = G$ and G_2 is the n-star. Observing that any sequence of vertex grouping operations that are applied to G_2 can only result in a star graph, the $W[1]$-hardness of the GRAPH STRUCTURAL SIMILARITY problem follows. □

The reduction described in the proof of the above theorem is clearly a linear fpt-reduction. Therefore, it follows from Theorem D.2 that:

Theorem D.4 *Unless ETH fails, the* GRAPH STRUCTURAL SIMILARITY *problem cannot be solved in time* $n^{o(k)}$*, where* n *is the number of vertices in* G_1 *and* G_2*, and* k *is the parameter.*

E. An easy instance of the VERTEX GROUPING problem

In this section we will show that some instances of the VERTEX GROUPING problem can be solved in polynomial time. We will consider the interesting case when the graph H, in the instances (G, H, k) of the VERTEX GROUPING problem, is the simple path P_k on k vertices.

For two vertices u and v in G, denote by the *distance* between u and v, $d_G(u, v)$, the length of a shortest path between u and v in G. Let G be an undirected graph, and k a positive integer. We start by providing a characterization of when P_k can be obtained from G by a sequence of vertex grouping operations. Equivalently, we provide a characterization of when G can be partitioned in k groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, such that each group $\mathcal{G}_i$, $i = 2, \ldots, k-1$, is connected and only connected to groups $\mathcal{G}_{i-1}$ and $\mathcal{G}_{i+1}$. We consider first the case when G is connected.

Lemma E.1 *Let G be a connected graph. Then G can be partitioned into k groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, such that each group $\mathcal{G}_i$, $i = 2, \ldots, k-1$, is connected and only connected to groups $\mathcal{G}_{i-1}$ and $\mathcal{G}_{i+1}$, if and only if there exist two vertices u and v in G satisfying $d_G(u, v) \geq k - 1$.*

PROOF. Suppose that there exist two vertices u and v in G satisfying $d_G(u, v) \geq k - 1$. Let $(u = u_1, u_1, \ldots, u_h = v)$ be a shortest path between u and v in G, where $h \geq k$. For $i = 1, \ldots, k-1$, let $\mathcal{G}_i = \{w \in G \mid d_G(u, w) = i - 1\}$, and with an abuse of the notation, let $\mathcal{G}_k = \{w \in G \mid d_G(u, w) \geq k - 1\}$, and note that $\mathcal{G}_i$ is nonempty, for $i = 1, \ldots, k$ because $u_i \in \mathcal{G}_i$. Since G is connected, every vertex in G must appear in one of the k groups $\mathcal{G}_1, \ldots, \mathcal{G}_k$. Moreover, by the definition of the groups and the connectedness of G, each group $\mathcal{G}_i$, $i = 2, \ldots, k$, is connected and only connected to groups $\mathcal{G}_{i-1}$ and $\mathcal{G}_{i+1}$.

Conversely, suppose that the vertices in G can be grouped into k groups, $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$,

such that each group $\mathcal{G}_i$, $i = 2, \ldots, k-1$, is connected and only connected to groups $\mathcal{G}_{i-1}$ and $\mathcal{G}_{i+1}$. Let u be a vertex in $\mathcal{G}_1$ and v a vertex in $\mathcal{G}_k$. Since G is connected, there exists a shortest path between u and v in G. Clearly, any path between u and v must pass through at least one vertex in each of the groups $\mathcal{G}_i$, $i = 2, \ldots, k-1$, and hence must have length at least $k-1$. It follows that $d_G(u, v) \geq k-1$. □

Now we address the case when G is not connected.

Lemma E.2 *Let G be an undirected graph, and assume that G is not connected. Let $C_1, \ldots, C_\ell$, where $\ell > 1$, be the connected components of G. Then G can be partitioned into k groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, such that each group $\mathcal{G}_i$, $i = 2, \ldots, k-1$, is connected and only connected to groups $\mathcal{G}_{i-1}$ and $\mathcal{G}_{i+1}$, if and only if there exist vertices u_i and v_i in C_i, for $i = 1, \ldots, \ell$, such that $d_G(u_i, v_i) + \ldots + d_G(u_\ell, v_\ell) \geq k-1$.*

Proof. We prove the statement for the case $\ell = 2$, and the proof for the general case follows by an inductive argument. Let C_1 and C_2 be the connected components of G.

Let u_1, v_1 be two vertices in C_1, and u_2, v_2 be two vertices in C_2 such that $d_G(u_1, v_1) + d_G(u_2, v_2) \geq k-1$. By Lemma E.1, we can group the vertices in C_1 into groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_r$, where $r = d_G(u_1, v_1) + 1$, such that each group $\mathcal{G}_j$, $j = 2, \ldots, r-1$, is connected and only connected to groups $\mathcal{G}_{j-1}$ and $\mathcal{G}_{j+1}$. Similarly, we can group the vertices of C_2 into groups $\mathcal{G}_{r+1}, \mathcal{G}_{r+2}, \ldots, \mathcal{G}_{r+s}$, where $s = d_G(u_2, v_2)+1$, such that each group $\mathcal{G}_j$, $j = r+2, \ldots, r+s-1$, is connected and only connected to groups $\mathcal{G}_{j-1}$ and $\mathcal{G}_{j+1}$. Now by grouping the vertices in $\mathcal{G}_r$ and $\mathcal{G}_{r+1}$ together, we obtain a grouping for G into groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{r+s-1}$, where $r + s - 1 = d_G(u_1, v_1) + d_G(u_2, v_2) + 1 \geq k$, such that each group $\mathcal{G}_j$, $j = 2, \ldots, r+s-2$ is connected and only connected to groups $\mathcal{G}_{j-1}$ and $\mathcal{G}_{j+1}$. Finally, by grouping the vertices in all the groups $\mathcal{G}_j$, where $j \geq k$, together, and calling the resulting group, without loss of

generality, $\mathcal{G}_k$, we obtain a grouping of G into groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, such that each group $\mathcal{G}_j$, $j = 2, \ldots, k-1$ is connected and only connected to groups $\mathcal{G}_{j-1}$ and $\mathcal{G}_{j+1}$.

To prove the converse, suppose that the vertices in G can be grouped into groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, such that each group $\mathcal{G}_j$, $j = 2, \ldots, k-1$, is connected and only connected to groups $\mathcal{G}_{j-1}$ and $\mathcal{G}_{j+1}$. Since each of C_1 and C_2 is connected, the vertices of C_1 must appear in a consecutive subsequence $\mathcal{G}_p, \mathcal{G}_{p+1}, \ldots, \mathcal{G}_{p+x}$ of the groups in $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$. Similarly, the vertices of C_2 must appear in a consecutive subsequence $\mathcal{G}_q, \mathcal{G}_{q+1}, \ldots, \mathcal{G}_{q+y}$ of the groups in $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$. Since every vertex in G must appear in C_1 or in C_2, and since any two adjacent groups in the sequence $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$ are connected, we have $(x+1)+(y+1)-1 \geq k$, which implies that $x + y \geq k - 1$. Let $u_1 \in \mathcal{G}_p$ and $v_1 \in \mathcal{G}_{p+x}$. Since C_1 is connected, there is a shortest path between u_1 and v_1 of length at least x. Similarly, there exists a shortest path from a vertex $u_2 \in \mathcal{G}_q$ to a vertex $v_2 \in \mathcal{G}_{q+y}$ of length at least y. It follows that $d_G(u_1, v_1) + d_G(u_2, v_2) \geq x + y \geq k - 1$.

This completes the proof. □

Theorem E.3 *let G be a graph on n vertices and m edges. Then in time $O(nm)$ it can be decided whether P_k can be obtained from G by a sequence of vertex grouping operations.*

PROOF. The proof follows from Lemma E.1 and Lemma E.2, and the fact that the shortest distance between all pairs of vertices in a graph can be computed in $O(nm)$ time by running a breadth first search algorithm at every vertex in the graph. □

F. Concluding remarks

In this chapter we studied the PSEUDO-ACHROMATIC NUMBER problem from the parameterized complexity point of view. Using interesting and non-trivial techniques from matching theory and network flows, we were able to show that the problem admits a kernel of

quadratic size that is computable in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in the graph, and k is the parameter. The result directly implies that the PSEUDO-ACHROMATIC NUMBER problem is fixed-parameter tractable, and gives a straight-forward brute-force algorithm that runs in $O(k^{k^2-k+2} + m\sqrt{n})$ time for the problem. Improving on this trivial upper bound for solving the problem remains an interesting open problem.

We also considered a generalization of the PSEUDO-ACHROMATIC NUMBER problem: the VERTEX GROUPING problem. Although the PSEUDO-ACHROMATIC NUMBER problem, which is a special case of the VERTEX GROUPING problem, is fixed-parameter tractable, we showed that the VERTEX GROUPING problem is in general $W[1]$-hard. We also showed that an interesting special case of the VERTEX GROUPING problem is solvable in polynomial time.

CHAPTER V

FIX-PARAMETER ENUMERABILITY

In the practice of computing, it is often required to generate a collection of good solutions, instead of a single best solution, for a given instance of an optimization problem. Motivated by this, we propose a new framework to systematically study the complexity of enumerating a given number K of best solutions for an instance of an NP optimization problem. Using elegant enumeration techniques and effective data structures, we show that many algorithm-design techniques for fixed-parameter tractable problems, such as branch-and-search, color coding, and bounded treewidth, can be adopted for the design of effective enumeration algorithms. In particular, we show that for a large class of well-known NP optimization problems, it takes fixed parameter tractable average time per solution to enumerate any required number K of best solutions for any given instance. The proposed framework is different from the previously-proposed ones, which either studied the complexity of counting the number of solutions or studied the complexity of enumerating all solutions for an instance of a given problem. For example, even though counting the number of k-paths in a graph is fixed-parameter infeasible, we present an efficient fixed-parameter enumeration algorithm for the problem.

A. Introduction

Most computational problems are concerned with finding a single solution for a problem instance. For example, decision problems ask for the existence of *a* solution to a given instance, while optimization problems seek *a* solution of the optimal value to a given instance [74].

On the other hand, many computational problems in practice seek a number of good

solutions rather than a single good solution. It is a natural practice in many branches in science and technology that one would like to identify a collection of "good solutions" and then (possibly) pick the most proper ones based on her own expertise in the discipline. Examples of such cases include seeking certain sub-structures in biological networks [67, 79], studying sequence motifs and alignments [75], studying evolutionary trees [49], list decoding [56], and finding the best-K queries [82] or top-K discords [16] in database systems.

Several approaches have been proposed towards meeting this need. The most notable one is the study of the counting complexity of a problem, which is the computational complexity of counting all the solutions to a given instance of the problem. Since its initialization by Valiant [84], significant work has been done on the study of counting complexity. Most of this work has focused on the hardness side, i.e., proving the intractability of certain counting problems. For example, Valiant [84] proved that counting the number of perfect matchings in a bipartite graph is #P-complete. Hunt et al. [60] proved the #P-hardness for a number of counting problems on planar graphs. Flum and Grohe [41] studied the parameterized complexity of counting problems and, in particular, proved that the problem of counting the number of k-paths in a graph is #W[1]-complete. Positive results along this line of research lead to a number of exact algorithms (e.g., [4, 31, 76]) and approximation algorithms (e.g., [29, 36]) for a number of counting problems that are intractable.

Another approach along this line of research studied the complexity of enumerating *all* solutions to a given problem instance. Tomita, Tanaka, and Takahashi [83] presented an exponential time algorithm that enumerates all maximal cliques in a graph. Gramm and Niedermeier [49] gave an algorithm that enumerates all minimum solutions for the QUARTET INCONSISTENCY problem. Fernau [42] considered a number of enumeration paradigms and studied their respective complexities.

None of the above approaches, however, has sufficiently met the practical needs of the

corresponding applications of the considered problems. For example, the study of counting complexity does not provide hints on how the solutions to a given instance of the problem can be generated. Moreover, the counting complexity of a problem can be significantly different from that of generating a single solution. For instance, in contrast to the hardness results in [84] and [41], finding a perfect matching in a bipartite graph is polynomial time solvable and constructing a k-path in a graph is fixed parameter tractable. The enumeration approach (i.e., enumerating all solutions to a given instance) may easily become computationally infeasible, not because of the difficulty of generating each single solution, but simply because the number of solutions is too large. For example, the problem of constructing a vertex cover of k vertices in a graph is practically feasible for small values of k [24], but the problem of enumerating all vertex covers of k vertices in a graph is computationally infeasible simply because there can be too many such vertex covers in the graph [42].

On the other hand, many computational applications *do not* ask for the entire set of solutions, instead, they only seek a certain number of "best" solutions [49, 67, 75, 79].

Motivated by this observation, we propose a new framework to study the effective enumerability of NP optimization problems. Needless to say, in order to be able to effectively enumerate a set of solutions, we must be able to generate a single solution. Therefore, we will be mainly interested in the NP optimization problems that have effective algorithms for generating a single solution. In particular, we will be seeking solutions of small size k, and study the enumerability of problems whose first solution can be generated in time $f(k)n^{O(1)}$ for a recursive function f — this is precisely the class of fixed parameter tractable problems studied in parameterized complexity theory [35]. We associate each problem solution with a "weight" that measures the quality/ranking of the solution. We say that an NP optimization problem is *fixed-parameter enumerable* (resp. *fixed-parameter linearly enumerable*) if there is an algorithm that, for a given problem instance (x, k) and an integer K, generates the K

best solutions of size k to x in time $f(k)n^{O(1)}K^{O(1)}$ (resp. in time $f(k)n^{O(1)}K$).

We argue that our enumeration model is meaningful from both the theoretical and practical perspectives. Indeed, generating K solutions takes time at least $\Omega(K)$ – therefore, it should be acceptable to require that generating the K best solutions take time polynomial in K. Besides the polynomial factor in K, we require that generating each of the best solutions takes an average time $f(k)n^{O(1)}$, which is feasible for small values of k. The model is especially suitable for applications that require a moderate number of best solutions, i.e, in which $K = n^{O(1)}$.

We investigate a number of popular techniques used in developing fixed-parameter tractable algorithms, including branch-and-search, color coding, and bounded tree-width. Using elegant enumeration techniques, combined with effective data structures, we show that these algorithm-design techniques can be translated into effective enumeration algorithms, and derive the fixed-parameter enumerability for many NP optimization problems. In particular, we show that a large class of well-known NP optimization problems are fixed-parameter linearly enumerable, for which, it takes fixed parameter tractable average time per solution to enumerate any given number K of the best solutions. Our construction also shows that our formulation is significantly different from the previous ones. For example, we present a fixed-parameter enumerable algorithm for the k-PATH problem, while counting the number of k-paths in a graph is known to be fixed-parameter intractable [41].

We note some differences between our approach and the previously-proposed approaches.

- As a natural extension of the theory of fixed parameter tractability [35], which studies the problem complexity of finding a single solution in terms of a single parameter k (which in most cases is the solution size), our approach studies the problem complexity parameterized in multi-dimensions, i.e., in terms of solution size k as well as the number K of solutions. In particular, the algorithms developed in our study are

output-sensitive.

- Different from the previous research [21, 45, 65] that studies the complexity of generating K best solutions for a specific problem whose corresponding search and optimization versions are polynomial time solvable, our concentration is on developing systematic and general enumerating techniques for a large class of optimization problems that most are NP-hard.

- Compared to the study of hardness of counting and other related problems [41, 60, 84, 31, 36, 50], which emphasizes the complexity issues of the problems, our research is focused on algorithmic aspects of problems. Algorithmic techniques for multi-solution enumeration have not been studied as extensively as that for single solution searching and optimization. Our research intends to develop effective algorithmic techniques for generating multiple solutions to meet the demands from practical computation. Also, this study avoids in many cases running into the realm of infeasibility simply resulting from the existence of a large number of solutions.

B. Definitions and preliminaries

We have presented the definition of FPT, W-Hierarchy in the introduction chapter, for more details readers are referred to the book [35].

1. Fixed parameter enumerability

We extend the standard definition of NP optimization problems [3] to encompass their parameterized versions.

Definition A *parameterized NP optimization problem* Q is a 4-tuple (I_Q, S_Q, f_Q, opt_Q) where

1. I_Q is the set of input instances of the form (x, k), where $x \in \Sigma^*$ for a fixed finite alphabet set Σ, and k is a non-negative integer called the *parameter*. The input instances are recognizable in polynomial time.
2. For each instance (x, k) in I_Q, $S_Q(x, k)$ is the set of *feasible solutions* for (x, k), which is defined by a polynomial p and a polynomial time computable predicate Φ (p and Φ depend only on Q) as $S_Q(x, k) = \{y : |y| \leq p(|x|) \text{ and } \Phi(x, k, y)\}$.
3. $f_Q(x, k, y)$ is the objective function mapping a pair $(x, k) \in I_Q$ and $y \in S_Q(x, k)$ to a real number. The function f_Q is computable in polynomial time.
4. $opt_Q \in \{\max, \min\}$.

Note that since the length of a solution y to an instance (x, k) in Q is bounded by a polynomial of $|x|$, the number of solutions to the instance (x, k) is bounded by $2^{q(|x|)}$ for some fixed polynomial q. Therefore, the values of the solutions in the set $S_Q(x, k)$ can be given in a finite sorted list $\tau_{x,k} = [f_Q(x, k, y_1), f_Q(x, k, y_2), \ldots]$, in a non-decreasing order when $opt_Q = \min$, and in a non-increasing order when $opt_Q = \max$. We say that the solutions $y'_1, \ldots, y'_K$ are the K *best solutions* for the instance (x, k) if the values $f_Q(x, k, y'_1)$, $\ldots$, $f_Q(x, k, y'_K)$, when sorted accordingly are identical to the first K values in the list $\tau_{x,k}$.

Definition A parameterized NP optimization problem Q is *fixed-parameter enumerable* if there are two algorithms A_1 and A_2 such that the following are true.

1. Given an instance (x, k) of Q, the algorithm A_1 generates a structure $\pi_{x,k}$ in time $f(k)n^{O(1)}$, where f is a recursive function independent of $n = |x|$.
2. Given the structure $\pi_{x,k}$ and an integer $K \geq 0$, the algorithm A_2 generates the K best

solutions to the instance (x, k) in time $|\pi_{x,k}|^{O(1)}K^{O(1)}$.[1]

The algorithm A_1 will be called the *structure algorithm*, and the algorithm A_2 will be called the *enumerating algorithm*. We say that the problem Q is *fixed parameter linearly enumerable* if the running time of the enumerating algorithm A_2 is $|\pi_{x,k}|^{O(1)}K$.

We comment on the above definitions. Since the algorithm A_1 runs in time $f(k)n^{O(1)}$, the size $|\pi_{x,k}|$ of the structure $\pi_{x,k}$ is bounded by $f(k)n^{O(1)}$. In consequence, the running time of the enumerating algorithm A_2 is bounded by $f_1(k)n^{O(1)}K^{O(1)}$, where f_1 is a recursive function independent of n. Moreover, we require that each input instance (x, k) of a fixed parameter enumerable problem Q have a small structure $\pi_{x,k}$ whose size is independent of the number K of solutions to be generated.

We argue that our definitions are practically meaningful and significant. First of all, the definitions are consistent with the normal sense of tractability and intractability. In particular, the definition requires that generating a single best solution (i.e., $K = 1$) can be done in a feasible amount of time $f_1(k)n^{O(1)}$ for a fixed function f_1, which coincides with the standard definition of fixed parameter tractability. Moreover, if one asks for a large number K of best solutions, then it seems reasonable to require that he pay computational time polynomial in K (besides the factor $f_1(k)n^{O(1)}$). In particular, for fixed parameter linearly enumerable problems, the average time for generating each best solution is bounded by $f_1(k)n^{O(1)}$ for a fixed function f_1 independent of n, which is feasible for moderate values of the parameter k.

We remark that there has been research in the literature that is related to the above formulation. For example, Chegireddy and Hamacher [21] developed algorithms for finding

[1]Note that it is possible that the total number $|S_Q(x, k)|$ of solutions is smaller than K. To avoid repeatedly distinguishing the two different cases, we will simply use the symbol K to refer to the value $K_0 = \min\{K, |S_Q(x, k)|\}$.

the K largest perfect matchings in a weighted graph, and Kapoor and Ramesh [65] studied the complexity of generating the K smallest spanning trees in a weighted graph. For a more comprehensive summary in this line of research, the readers are referred to [45]. However, to the authors' knowledge, all these works are on optimization problems that are solvable in polynomial time, and each of them deals with a very specific problem. On the other hand, this chapter is mainly focused on NP-hard optimization problems, and on developing systematical techniques for effective solution enumerations for a large class of NP-hard optimization problems.

2. Fixed parameter tractable problems and fixed parameter enumerable problems

It is natural to ask the relationship between the class of fixed parameter tractable problems and the class of fixed parameter enumerable problems. We first note a difference between standard parameterized problems and parameterized NP optimization problems. A standard parameterized problem [35] is a decision problem which asks for the existence of a solution. On the other hand, an instance of a parameterized NP optimization problem is associated with a set of solutions that can be ranked by solution weights, and asks for constructing the best solutions. Nevertheless, we have the following simple relationship between the two notions.

Theorem B.1 *If a parameterized NP optimization problem Q is fixed parameter enumerable then it is fixed parameter tractable.*

PROOF. Suppose that Q is fixed parameter enumerable and let (x, k) be an instance of Q. We use the structure algorithm A_1 to construct the structure $\pi_{x,k}$ in time $f(k)n^{O(1)}$. Then we apply the enumerating algorithm A_2 to $\pi_{x,k}$ with $K = 1$; this takes time $f_1(k)n^{O(1)}$ and correctly tests whether (x, k) has a solution or not. The whole process takes time

$(f(k) + f_1(k))n^{O(1)}$. Therefore, the problem Q, when regarded as a standard parameterized decision problem, is fixed parameter tractable. □

Therefore, a necessary condition for a problem Q to be fixed parameter enumerable is that it is fixed parameter tractable. On the other hand, as observed by Hans Bodlaender, the converse of this fact is false under the parameterized complexity hypothesis. In fact, there are natural parameterized problems that are fixed parameter tractable but not fixed parameter enumerable. In the following, we describe such a problem, which is a modification of the construction suggested by Hans Bodlaender.

Recall that a set I of vertices in a graph G is an *independent set* if no two vertices in I are adjacent, and that a set D of vertices in G is a *dominating set* if every vertex in G is either in D or adjacent to at least one vertex in D. A set B of vertices in the graph G is an *independent dominating set* if B is both an independent set and a dominating set in the graph G.

Let h be an integer. A graph $G = (V, E)$ is an *h-tent* if there is a vertex v in G such that: (1) the set $N(v)$ of neighbors of v forms an independent set; (2) $|N(v)| = h$; and (3) every vertex in $N(v)$ is adjacent to all vertices in $V - N(v)$. Intuitively, an h-tent is the complete bipartite graph $K_{h,s} = (V_h \cup V_s, E)$ for some integer s with proper edges added among $s - 1$ vertices in the set V_s.

Consider the following parameterized problem.

> TENT INDEPENDENT DOMINATING SET (TENT-IDS): given a graph G and an integer k, decide if the graph G is a k-tent and has an independent dominating set of size k.

The TENT-IDS problem can be regarded as a parameterized NP optimization problem if we assign each vertex in the graph weight 1 and look for size-k independent dominating

sets of the maximum weight in the graph.

Theorem B.2 *There are fixed parameter tractable problems that are not fixed parameter enumerable unless* $W[2] = FPT$.

PROOF. Bodlaender proved this theorem by giving a fixed parameter tractable problem whose fixed parameter enumerability implies $W[2] = \text{FPT}$. Here we follow his idea and prove that the TENT-IDS problem is fixed parameter tractable but not fixed parameter enumerable unless $W[2] = \text{FPT}$. First we show that TENT-IDS is fixed parameter tractable. In fact, the problem is polynomial time solvable: for a given instance (G, k), we check whether there is a vertex v in G such that: (1) the set $N(v)$ of neighbors of v forms an independent set; (2) v has exactly k neighbors; and (3) each neighbor of v is adjacent to all vertices in $G - N(v)$. This process obviously takes polynomial time. Moreover, if no such a vertex exists in G then G is not a k-tent, and (G, k) is a no-instance of the TENT-IDS problem. On the other hand, if there is such a vertex v in G, then the neighbors of v obviously form an independent dominating set of size k for the graph G, and (G, k) is a yes-instance of the TENT-IDS problem.

To prove that the TENT-IDS problem is not fixed parameter enumerable, consider the standard INDEPENDENT DOMINATING SET problem (IDS): given a graph G and an integer k, decide if G has a size-k independent dominating set.

Given an instance (G, k) of the IDS problem, construct a new graph G' as follows: (1) add $k + 2$ new vertices v_0, v_1, ..., v_{k+1} to G; (2) add an edge between v_0 and each v_i, for $1 \leq i \leq k + 1$; and (3) add an edge between v_i and w for each i, $1 \leq i \leq k + 1$, and for each vertex w in G. It is easy to see that the graph G' is a $(k + 1)$-tent, and that the set $B_1 = \{v_1, \ldots, v_{k+1}\}$ is a size-$(k + 1)$ independent dominating set in G'. We show that the graph G' has more than one size-$(k+1)$ independent dominating set if and only if the graph

G has a size-k independent dominating set.

If the graph G has a size-k independent dominating set B', then the set $B' \cup \{v_0\}$ is obviously a size-$(k+1)$ independent dominating set for the graph G'. Therefore, the graph G' has at least two size-$(k+1)$ independent dominating sets: B_1 and $B' \cup \{v_0\}$.

On the other hand, suppose that the graph G' has a size-$(k+1)$ independent dominating set B_2 that is different from the set B_1. We have the following simple facts. First, no vertex in B_1 can be in B_2. In fact, if a vertex v_i in B_1 is in B_2, then neither v_0 nor vertices in the original graph G can be in B_2 because v_i is adjacent to all these vertices and B_2 is an independent set. This would imply that $B_2 \subseteq B_1$, and hence $B_2 = B_1$ because B_1 and B_2 are of the same cardinality. Second, the vertex v_0 is in B_2. This is because: the vertex v_0 is only adjacent to vertices in B_1, no vertex in B_1 is in B_2, and the vertex v_0 must be either in B_2 or adjacent to a vertex in B_2. It follows from the above facts that the set $B_2 - \{v_0\}$ is entirely contained in the original graph G. Moreover, it is now easy to verify that the set $B_2 - \{v_0\}$ is a size-k independent dominating set in the graph G.

Now we can show that if the TENT-IDS problem is fixed parameter enumerable, then the IDS problem is fixed parameter tractable. Given an instance (G, k) of the IDS problem, we construct the graph G' as described above, and consider the instance (G', k', K), where $k' = k + 1$, and $K = 2$, of the enumeration problem TENT-IDS. If the TENT-IDS problem is fixed parameter enumerable, then by definition, there is an algorithm $\mathcal{A}$ that, on the instance (G', k', K) where $k' = k + 1$ and $K = 2$, generates the largest two size-k' (i.e., size-$(k+1)$) independent dominating sets in the graph G' in time $f(k')n^{O(1)}K^{O(1)} = f_1(k)n^{O(1)}$ for some function f_1. By the discussion above, the algorithm $\mathcal{A}$ on instance (G', k', K) returns more than one size-$(k+1)$ independent dominating set for the graph G' if and only if the graph G has a size-k independent dominating set. Therefore, it is decidable in time $f_1(k)n^{O(1)}$ whether (G, k) is a yes-instance for the IDS problem, and in consequence, the IDS problem is

fixed parameter tractable.

It is well-known that the IDS problem is W[2]-hard [35]. Therefore, the fixed parameter tractability of the IDS problem would imply that W[2] = FPT. This completes the proof that the fixed parameter enumerability of the TENT-IDS problem would imply W[2] = FPT. □

In the rest of chapter, we examine the techniques that have been widely used in the development of fixed parameter tractable algorithms. We show that these techniques, when carefully revised, give nice structure algorithms. We then also develop new techniques for effective enumerating algorithms. We demonstrate our techniques based on some of the best-known fixed parameter tractable problems and show that the corresponding parameterized NP optimization problems are fixed parameter enumerable. This indicates that the research on fixed parameter tractable algorithms may also have a direct impact on the study of effective enumeration algorithms for NP optimization problems.

The following simple technique will be useful in our enumerating algorithms. Suppose that we have a list of n real numbers. By first finding the K-th largest (or the K-th smallest) number a in the list in time $O(n)$ [30] then partitioning the list using a as a "pivot", we can generate the K largest (or the K smallest) numbers in the list in time $O(n)$.

C. Effective enumerations based on branch-and-search

The branch-and-search method based on bounded search-trees has been a very popular and powerful technique in the development of efficient exact and parameterized algorithms [35].

A typical parameterized algorithm A that uses a branch-and-search process works in the following recursive way: given an input (x, k), the algorithm A constructs a sequence of sub-instances (x_1, k_1), ..., (x_s, k_s), where $k_j < k$ for all j, recursively solves these sub-instances, then constructs a solution for the original instance (x, k) based on the solutions for the sub-instances. This process can be described as a search-tree: the root is labeled with

the original input instance (x, k) and its children are labeled with the sub-instances (x_1, k_1), ..., (x_s, k_s), each is recursively represented by a corresponding search sub-tree. Each leaf of the search-tree is labeled with an instance that is easy enough to be solved directly.

The computational time of the parameterized algorithm A based on the branch-and-search process can be measured by the number of leaves (or equivalently, the number of nodes) in the search tree if the computation and construction at each node in the search tree are relatively easy (e.g., can be done in polynomial time).

We discuss how to use this technique in effective enumeration algorithms for parameterized NP optimization problems. As a running example, we describe our algorithm with VERTEX COVER as the underlying problem. Recall that a vertex set C in a graph G is a *vertex cover* for G if each edge in G has at least one end in C. A vertex cover of k vertices will be called a *k-vertex cover*. The VERTEX COVER problem is a well-known fixed parameter tractable problem, and parameterized algorithms for this problem have been extensively studied (e.g., [22, 24]). Moreover, the counting complexity and the complexity of enumerating all solutions of the problem have also been examined. Arvind and Raman [4] (see also [41]) showed that counting the total number of k-vertex covers can be done in time $O(2^{k^2+k}k + 2^k n)$. The complexity of enumerating all k-vertex covers, however, depends on whether k is the size of a minimum vertex cover of the graph. Fernau [42] showed that if k is equal to the size of a minimum vertex cover, then enumerating all k-vertex covers can be done in time $O(2^k k^2 + kn)$, while if k is not equal to the size of a minimum vertex cover, then no algorithm of running time $f(k)n^{O(1)}$ for any recursive function f can enumerate all k-vertex covers. The latter fact holds simply because in such case the number of k-vertex covers can be too large to be enumerated in such time.

We investigate the fixed parameter enumerability of the problem. We assume that the input graph G is weighted in which each vertex is associated with a real number (the *vertex*

weight). The *weight* of a vertex cover C is the sum of the weights of the vertices in C. Thus, a vertex cover C_1 is *smaller than* a vertex cover C_2 if the weight of C_1 is smaller than the weight of C_2.

WEIGHTED VERTEX COVER: Given a weighted graph G of n vertices, and integers k and K, generate the K smallest k-vertex covers in G.

1. The structure algorithm

Let (G, k) be an instance of the WEIGHTED VERTEX COVER problem, where G is a graph of n vertices. Since a vertex of degree larger than k must be in every k-vertex cover of G, we can first remove all vertices of degree larger than k from the graph and then work on the remaining graph. This preprocess can be done in time $O(kn)$ even when the number of edges in G is larger than kn. Now the resulting instance (G', k') has a graph G' of $O(n)$ vertices and $O(kn)$ edges and a parameter $k' \leq k$. Thus, without loss of generality, we can assume that our input graph G has n vertices and $O(kn)$ edges.

Our structure algorithm for WEIGHTED VERTEX COVER is a recursive algorithm based on the branch-and-research method, which on an input instance (G, k) returns a collection $\mathcal{L}(G, k)$ of triples (I, O, R), where each (I, O, R) is a partition of the vertex set of the graph G, representing the set of all k-vertex covers that include all vertices in I and exclude all vertices in O. Moreover, we require that in the subgraph induced by the vertex set R, all vertices have degree bounded by 2. The structure algorithm is given in Figure 12.

Theorem C.1 *On an input (G, k), the algorithm* **structure-vc** *runs in time $O(1.47^k n)$, and returns a collection $\mathcal{L}(G, k)$ of at most 1.466^k triples.*

PROOF. We first prove the second claim. Let $L(k)$ be the number of triples in the collection $\mathcal{L}(G, k)$ returned by the algorithm **structure-vc** on the input (G, k). If the input (G, k)

Algorithm **structure-vc**

INPUT: *G: a weighted graph; k: an integer;*

1. **if** $(k < 0)$ or $(k = 0$ but the edge set of G is not empty$)$ **then return** $\mathcal{L}(G, k) = \emptyset$;
2. **if** there is no vertex of degree larger than 2 in G **then return** $\mathcal{L}(G, k) = \{(\emptyset, \emptyset, V)\}$;
3. pick any vertex v of degree $d \geq 3$;
4. let $G_1 = G - v$ and $G_2 = G - (v \cup N(v))$ where $N(v)$ is the set of neighbors of v;
5. recursively call **structure-vc**$(G_1, k-1)$ and **structure-vc**$(G_2, k-d)$; let the returned collections be $\mathcal{L}(G_1, k-1)$ and $\mathcal{L}(G_2, k-d)$, respectively;
6. $\mathcal{L}(G, k) = \emptyset$;
7. **for** each triple (I_1, O_1, R_1) in $\mathcal{L}(G_1, k-1)$ **do** add $(I_1 \cup \{v\}, O_1, R_1)$ to $\mathcal{L}(G, k)$;
8. **for** each triple (I_2, O_2, R_2) in $\mathcal{L}(G_2, k-d)$ **do** add $(I_2 \cup N(v), O_2 \cup \{v\}, R_2)$ to $\mathcal{L}(G, k)$;

Fig. 12. The structure algorithm for WEIGHTED VERTEX COVER.

satisfies the conditions in step 1 or step 2, then $L(k) \leq 1$. In particular, $L(k) \leq 1$ for $k \leq 0$. Otherwise, the value $L(k)$ satisfies the recurrence relation $L(k) \leq L(k-1) + L(k-d)$, where $d \geq 3$. Using the standard technique for solving this recurrence relation (see [22] for a detailed discussion on this technique), we get $L(k) \leq \alpha^k$, where $\alpha = 1.4655\cdots < 1.466$ is the unique positive root of the polynomial $x^k - x^{k-1} - x^{k-3}$. This proves the second claim in the theorem.

Let $T(k, G)$ be the running time of the algorithm **structure-vc** on the input (G, k). If (G, k) satisfies the conditions in step 1 or step 2, then $T(k, G) = O(kn)$ (recall that we can assume the size of the graph G is $O(kn)$). In particular, $T(k, G) = O(kn)$ for $k \leq 0$. Otherwise, the value $T(k, G)$ satisfies the following recurrence relation

$$T(k, G) \leq T(k-1, G-v) + T(k-d, G-(v \cup N(v))) + O(1.466^k n),$$

where v is a vertex of degree $d \geq 3$ in G, $N(v)$ is the set of neighbors of v in G, and the term $O(1.466^k n)$ is the time for constructing the graphs $G_1 = G - v$ and $G_2 = G - (v \cup N(v))$ and for constructing the collection $\mathcal{L}(G, k)$ from the collections $\mathcal{L}(G_1, k-1)$ and $\mathcal{L}(G_2, k-d)$

(here we have used the fact that $\mathcal{L}(G,k)$ contains at most 1.466^k triples). Using induction on k and n, one can easily verify that $T(k,G) = O(1.466^k kn) = O(1.47^k n)$. □

We say that a vertex cover C of the graph G is *consistent with* a partition (I,O,R) of the vertex set of G if C contains all vertices in I and excludes all vertices in O.

Lemma C.2 *Let $\mathcal{L}(G,k)$ be the collection returned by the algorithm* **structure-vc** *on input (G,k). Then every k-vertex cover of G is consistent with exactly one triple in $\mathcal{L}(G,k)$.*

PROOF. If (G,k) satisfies the conditions in step 1, then the graph G has no k-vertex cover. If (G,k) satisfies the condition in step 2, then clearly every k-vertex cover of G is consistent with the unique triple $(\emptyset,\emptyset,V)$ in $\mathcal{L}(G,k)$.

Now assume that (G,k) does not satisfy the conditions in step 1 and step 2. Let C be any k-vertex cover of G, and let v be the vertex picked in step 3. It should be true that the vertex cover C either contains v or does not contain v but contains all neighbors of v.

If C contains v, then the set $C_1 = C - v$ is a $(k-1)$-vertex cover of the graph $G_1 = G - v$. By the inductive hypothesis, C_1 is consistent with exactly one triple in the collection $\mathcal{L}(G_1, k-1)$. Thus, the set $C = C_1 + v$ is consistent with exactly one of the triples constructed in step 7. Moreover, since C contains v, C cannot be consistent with any triples constructed in step 8. In conclusion, in this case the k-vertex cover C is consistent with exact one triple in $\mathcal{L}(G,k)$.

If C does not contain v but contains all vertices in $N(v)$, then the set $C_2 = C - N(v)$ is a $(k-d)$-vertex cover of the graph $G_2 = G - (v \cup N(v))$, and C_2 is consistent with exactly one triple in $\mathcal{L}(G_2, k-d)$. Thus, the set $C = C_2 \cup N(v)$ is consistent with exactly one of the triples constructed in step 8 but is not consistent with any triples constructed in step 7. In conclusion, the k-vertex cover C is consistent with exactly one triple in $\mathcal{L}(G,k)$. □

The collection $\mathcal{L}(G,k)$ forms the structure $\tau_{G,k}$ for the instance (G,k) of the WEIGHTED VERTEX COVER problem. By Theorem C.1, the structure $\tau_{G,k}$ can be constructed in time $O(1.47^k n)$.

2. The enumerating algorithm

Let $\mathcal{L}(G,k)$ be the structure returned by the algorithm **structure-vc** on input (G,k). By Lemma C.2, every k-vertex cover C of G is consistent with a unique triple (I,O,R) in $\mathcal{L}(G,k)$ in the sense that C contains all vertices in I and excludes all vertices in O. Thus, the k-vertex cover C must consist of the vertex set I plus a vertex cover of $k-|I|$ vertices for the subgraph $G(R)$ induced by the vertex set R. Therefore, the K smallest k-vertex covers of the graph G that are consistent with the triple (I,O,R) can be generated by generating the K smallest $(k-|I|)$-vertex covers for the induced subgraph $G(R)$. Finally, the K smallest k-vertex covers of the original graph G can be obtained by performing the above process on all triples in the structure $\mathcal{L}(G,k)$.

From the algorithm **structure-vc**, all vertices in the induced subgraph $G(R)$ have degree bounded by 2. Therefore, we first discuss how we deal with this kind of graphs.

Lemma C.3 *Let G be a graph of n vertices in which all vertices have degree bounded by 2. Then the K smallest k-vertex covers of G can be generated in time $O(Kkn)$.*

PROOF. Since all vertices in G have degree bounded by 2, every connected component of G is either an isolated vertex, a simple path, or a simple cycle. Order the vertices of G in a list $W=[v_1,v_2,\ldots,v_n]$ such that the vertices of each connected component of G appear in a consecutive subsegment in W. In particular, the vertices of a simple path appear in W in the order by which we traverse the path from one end to the other end, and the vertices of a simple cycle appear in W in the order by which we traverse the entire cycle (starting

from an arbitrary vertex in the cycle). A vertex v_i is a *type-1 vertex* if it has degree 0 in G, a *type-2 vertex* if it is in a connected component of G that is a simple path of length larger than 0, and a *type-3 vertex* if it is in a connected component of G that is a simple cycle.

For each i, $1 \leq i \leq n$, let G_i be the subgraph of G induced by the vertex set $\{v_1, v_2, \ldots, v_i\}$. For each induced subgraph G_i, we form a list $L_i = [S_{i,0}, S_{i,1}, \ldots, S_{i,k}]$, where $S_{i,j}$ is a set of j-vertex covers of G_i, defined as follows:

(1) If v_i is of type-1, then $S_{i,j}$ is the set of the K smallest j-vertex covers for G_i (recall that this really means "the K smallest j-vertex covers or all j-vertex covers if the total number of j-vertex covers is smaller than K"– this remark is also applied to the following discussions);

(2) If v_i is of type-2, then $S_{i,j}$ consists of two sets $S'_{i,j}$ and $S''_{i,j}$, where $S'_{i,j}$ contains the K smallest j-vertex covers of G_i that contain v_i, and $S''_{i,j}$ contains the K smallest j-vertex covers of G_i that do not contain v_i;

(3) If v_i is of type-3 and in a simple cycle $[v_h, \ldots, v_i, \ldots, v_t]$ in G, then $S_{i,j}$ consists of four sets $S'_{i,j}$, $S''_{i,j}$, $S'''_{i,j}$ and $S''''_{i,j}$, where $S'_{i,j}$ is the set of the K smallest j-vertex covers of G_i that contain both v_h and v_i, $S''_{i,j}$ is the set of the K smallest j-vertex covers of G_i that contain v_h but not v_i, $S'''_{i,j}$ is the set of the K smallest j-vertex covers of G_i that contain v_i but not v_h, and $S''''_{i,j}$ is the set of the K smallest j-vertex covers of G_i that contain neither v_h nor v_i.

Note that since each set $S_{i,j}$ contains at most $4K$ j-vertex covers, the set $S^0_{i,j}$ of the K smallest j-vertex covers of the graph G_i can always be constructed from $S_{i,j}$ in time $O(K)$.

The list L_1 can be trivially constructed: (1) if v_1 is of type-1, then all $S_{1,j}$ are empty except $S_{1,0} = \{\emptyset\}$ and $S_{1,1} = \{(v_1)\}$; (2) if v_i is of type-2, then all $S'_{1,j}$ and $S''_{1,j}$ are empty except $S''_{1,0} = \{\emptyset\}$ and $S'_{1,1} = \{(v_1)\}$; and (3) if v_i is of type-3, then all $S'_{1,j}$, $S''_{1,j}$, $S'''_{1,j}$, and $S''''_{1,j}$ are empty except $S''''_{1,0} = \{\emptyset\}$ and $S'_{1,1} = \{(v_1)\}$.

Inductively, suppose that we have built the list L_{i-1}. To construct the list L_i, we distinguish the cases based on the type of the vertex v_i.

Case 1. The vertex v_i is of type-1.

Then the graph G_i is the graph G_{i-1} plus an isolated vertex v_i. For each j, $0 \leq j \leq k$, let $S^0_{i-1,j}$ be the set of the K smallest j-vertex covers of the graph G_{i-1}, which can be constructed in time $O(K)$. Since each vertex cover of G_i is either a vertex cover of G_{i-1}, or a vertex cover of G_{i-1} plus the vertex v_i, the set $S_{i,j}$ in L_i can be constructed as follows: take each $(j-1)$-vertex cover of G_{i-1} from $S^0_{i-1,j-1}$ and add the vertex v_i to it to make a j-vertex cover of G_i. This gives a set F of K j-vertex covers for G_i. It is clear that the K smallest j-vertex covers of G_i must be contained in the union $F \cup S^0_{i-1,j}$, which is a set of $2K$ j-vertex covers for G_i. Thus, the K smallest j-vertex covers in the union $F \cup S^0_{i-1,j}$ will make the set $S_{i,j}$. Each set $S_{i,j}$ can be constructed in time $O(K)$, and the list L_i can be constructed from the list L_{i-1} in time $O(Kk)$.

Case 2. The vertex v_i is of type-2.

Then v_i is in a connected component $[v_h, \ldots, v_i, \ldots, v_t]$ of G, where $h < t$, that is a simple path. As in Case 1, for each j, let $S^0_{i-1,j}$ be the set of the K smallest j-vertex covers of G_{i-1}.

If $v_i = v_h$ is the first vertex in the path, then the graph G_i is the graph G_{i-1} plus an isolated vertex v_i. Thus, the set $S'_{i,j}$ can be obtained from $S^0_{i-1,j-1}$ by adding the vertex v_i to each $(j-1)$-vertex cover of G_{i-1} in $S^0_{i-1,j-1}$, and the set $S''_{i,j}$ is equal to the set $S^0_{i-1,j}$.

If $h < i$ and v_i is not the first vertex in the path, then the graph G_i is the graph G_{i-1} plus the vertex v_i and the edge $[v_{i-1}, v_i]$. Therefore, each vertex cover of G_i is either a vertex cover of G_{i-1} plus v_i, or a vertex cover of G_{i-1} that contains v_{i-1}. Thus, the set $S'_{i,j}$ is again obtained from $S^0_{i-1,j-1}$ by adding the vertex v_i to each $(j-1)$-vertex cover of G_{i-1} in $S^0_{i-1,j-1}$. On the other hand, now the set $S''_{i,j}$ is equal to the set $S'_{i-1,j}$.

Again in this case, the list L_i can be constructed from the list L_{i-1} in time $O(Kk)$.

Case 3. The vertex v_i is of type-3.

Then v_i is in a connected component $[v_h, \ldots, v_i, \ldots, v_t]$ of G that is a simple cycle. Again for each j, let $S^0_{i-1,j}$ be the set of the K smallest j-vertex covers of G_{i-1}.

If $v_i = v_h$ is the first vertex in the cycle, then the graph G_i is the graph G_{i-1} plus an isolated vertex v_i. Thus, the set $S'_{i,j}$ can be obtained from $S^0_{i-1,j-1}$ by adding the vertex v_i to each $(j-1)$-vertex cover of G_{i-1} in $S^0_{i-1,j-1}$, and the set $S''''_{i,j}$ is equal to the set $S^0_{i-1,j}$. By definition, the sets $S''_{i,j}$ and $S'''_{i,j}$ are empty.

If $h < i < t$, then the graph G_i is the graph G_{i-1} plus the vertex v_i and the edge $[v_{i-1}, v_i]$. Therefore, the set $S'_{i,j}$ can be obtained by adding the vertex v_i to each $(j-1)$-vertex cover in the union $S'_{i-1,j-1} \cup S''_{i-1,j-1}$ then selecting the K smallest ones; the set $S''_{i,j}$ is equal to the set $S'_{i-1,j}$; the set $S'''_{i,j}$ is obtained by adding the vertex v_i to each $(j-1)$-vertex cover in the union $S'''_{i-1,j-1} \cup S''''_{i-1,j-1}$ then selecting the K smallest ones; and the set $S''''_{i,j}$ is equal to the set $S'''_{i-1,j}$.

If $v_i = v_t$ is the last vertex in the cycle, then the graph G_i is the graph G_{i-1} plus the vertex v_i and two edges $[v_h, v_i]$ and $[v_{i-1}, v_i]$. In this case, the set $S'_{i,j}$ can be obtained by adding the vertex v_i to each $(j-1)$-vertex cover in the union $S'_{i-1,j-1} \cup S''_{i-1,j-1}$ then selecting the K smallest ones; the set $S''_{i,j}$ is equal to the set $S'_{i-1,j}$; the set $S'''_{i,j}$ is obtained by adding the vertex v_i to each $(j-1)$-vertex cover in the union $S'''_{i-1,j-1} \cup S''''_{i-1,j-1}$ then selecting the K smallest ones; and the set $S''''_{i,j}$ is empty because $[v_h, v_i]$ is an edge in G_i.

The correctness of these constructions can be easily verified using the definitions of the sets $S'_{i,j}$, $S''_{i,j}$, $S'''_{i,j}$, and $S''''_{i,j}$. Moreover, it is also easy to see that the list L_i can be constructed from the list L_{i-1} in time $O(Kk)$.

Summarizing all the above, we conclude that the list L_n can be constructed in time $O(Kkn)$. Now the K smallest k-vertex covers of the graph $G = G_n$ can be easily obtained

in time $O(K)$ from the set $S_{n,k}$ in the list L_n. This completes the proof of the lemma. □

Now it is obvious in principle how we can generate the K smallest k-vertex covers for the graph G: they can be obtained by first generating the K smallest consistent k-vertex covers for each triple in $\mathcal{L}(G,k)$. Moreover, we can further speedup the enumeration process as shown by the following theorem.

Theorem C.4 *Let (G,k) be an instance of the* WEIGHTED VERTEX COVER *problem, and let $\mathcal{L}(G,k)$ be the structure returned by the algorithm* **structure-vc** *on (G,k). Then the K smallest k-vertex covers of the graph G can be generated in time $O(1.47^k n + 1.22^k Kn)$.*

PROOF. Let (I,O,R) be a triple in $\mathcal{L}(G,k)$ and let $k_1 = k - |I|$. By Lemma C.3, the K smallest k_1-vertex covers of the induced subgraph $G(R)$ can be constructed in time $O(Kk_1n)$. Then the vertex set I plus each of these k_1-vertex covers for $G(R)$ makes one of the K smallest k-vertex covers consistent with (I,O,R) for the graph G. Thus, the K smallest k-vertex covers of G consistent with (I,O,R) can be constructed in time $O(Kkn)$. Moreover, by Lemma C.2, every k-vertex cover of G is consistent with a triple in $\mathcal{L}(G,k)$. Therefore, if we generate K smallest consistent k-vertex covers for each triple in $\mathcal{L}(G,k)$, and pick the K smallest among all these generated k-vertex covers, we will get the K smallest k-vertex covers for the graph G.

Let L be the total number of triples in $\mathcal{L}(G,k)$.

If $K \leq \sqrt{L}$, then let $K_1 = 1$, and construct the K_1 smallest k-vertex covers (i.e., *the* smallest k-vertex cover) consistent with each triple in $\mathcal{L}(G,k)$, and then make the set S_1 of the K smallest k-vertex covers from these $LK_1 = L$ k-vertex covers. This takes time $O(LK_1kn) = O(Lkn)$. Note that if the set S_1 does not contain the smallest k-vertex cover consistent with a triple (I,O,R), then no k-vertex cover consistent with (I,O,R) can be among the K smallest k-vertex covers for G. Thus, we can remove all the triples that have no

consistent k-vertex covers in the set S_1. After this, the total number L_1 of remaining triples is bounded by $K \leq \sqrt{L}$. Now on each of these L_1 triples, we apply Lemma C.3 and construct the K smallest consistent k-vertex covers. This takes time $O(L_1Kkn) = O(\sqrt{L}Kkn)$. Now picking the K smallest among these L_1K k-vertex covers takes time $O(L_1K) = O(\sqrt{L}K)$ and gives the K smallest k-vertex covers for the graph G. In summary, in this case, the K smallest k-vertex covers of G can be generated in time $O(Lkn + \sqrt{L}Kkn)$.

Now suppose that $K > \sqrt{L}$. Then let $K_2 = K/\sqrt{L}$, and construct the K_2 smallest consistent k-vertex covers for each triple in $\mathcal{L}(G, k)$, and make the set S_2 of the K smallest k-vertex covers among all these LK_2 k-vertex covers. This takes time $O(LK_2kn) = O(\sqrt{L}Kkn)$. For each triple (I, O, R) whose K_2 smallest consistent k-vertex covers are not all in the set S_2, only those k-vertex covers consistent with (I, O, R) that are already in the set S_2 can be possibly among the K smallest k-vertex covers of the graph G. Thus, once we get the set S_2, we can remove those triples whose K_2 smallest consistent k-vertex covers are not all in the set S_2. Since no k-vertex cover is consistent with more than one triple in $\mathcal{L}(G, k)$, there are at most $\sqrt{L}$ triples in $\mathcal{L}(G, k)$ for which the K_2 smallest consistent k-vertex covers are all in the set S_2. Therefore, the number L_2 of the remaining triples is bounded by $\sqrt{L}$. Now in time $O(L_2Kkn) = O(\sqrt{L}Kkn)$, we can apply Lemma C.3 to these L_2 triples and generate the K smallest consistent k-vertex covers from each of these triples. Now the K smallest k-vertex covers among these L_2K k-vertex covers and those in the set S_2 are the K smallest k-vertex covers of the graph G. In summary, in this case, the K smallest k-vertex covers of the graph G can be generated in time $O(\sqrt{L}Kkn)$.

In conclusion, given the structure $\mathcal{L}(G, k)$, the K smallest k-vertex covers of the graph G can be generated in time $O(Lkn + \sqrt{L}Kkn)$. The theorem follows now from Theorem C.1 because $L \leq 1.466^k$ thus $Lk = O(1.47^k)$ and $\sqrt{L}k = O(1.22^k)$. □

Corollary C.5 *The* WEIGHTED VERTEX COVER *problem is fixed parameter linearly enumer-*

able. More specifically, given an instance (G, k) *and an integer* K*, the* K *smallest* k*-vertex covers of the graph* G *can be generated in time* $O(1.47^k n + 1.22^k Kn)$*.*

We give some remarks before we close this section. Constructing a single k-vertex cover in a graph in which all vertices have degree bounded by 2 is trivial [22]. On the other hand, Theorem C.3 shows that we need to be much more careful when we generate the K smallest k-vertex covers in such a graph. Moreover, Corollary C.5 may be a little surprising in the sense that the "average running time" for generating each of the K smallest k-vertex covers is $O(1.22^k n)$, which is actually better than the fastest known algorithm for generating a single k-vertex cover [24]. In particular, this corollary shows that the cost of generating even many smallest k-vertex covers for a graph (e.g., K can be as large as 1.22^k) is still comparable with that of generating a single k-vertex cover of the graph.

D. Effective enumeration based on color coding

Recent research in parameterized computation has shown that the *color coding* technique [5] is very powerful and useful in the development of efficient parameterized algorithms. In particular, the technique has been used in developing improved parameterized algorithms for the k-PATH problem [5, 27], for matching and set packing problems [39, 69], and for problems in computational biology [79]. In this section, we show that the color coding technique is also very effective for developing enumeration algorithms for parameterized NP optimization problems. We illustrate our techniques by presenting an enumeration algorithm for the k-PATH problem.

A simple path in a graph G is a *k-path* if it contains exactly k vertices. The *weight* of a path in a weighted graph is the sum of weights of the vertices in the path. The problem can be formally defined as follows.

WEIGHTED k-PATH: given a weighted graph G and integers k and K, generate the K largest k-paths in G.

1. The structure algorithm

A *k-coloring* of a set S is a function from S to $\{1, 2, \ldots, k\}$. A collection $\mathcal{F}$ of k-colorings of S is a *k-color coding scheme* for S if for any subset W of k elements in S, there is a k-coloring f_W in $\mathcal{F}$ such that no two elements in W are assigned the same color by f_W. The *size* of the k-color coding scheme $\mathcal{F}$ is equal to the number of k-colorings in $\mathcal{F}$. Alon, Yuster, and Zwick [5] showed that there is a k-color coding scheme of size $2^{O(k)}n$ for a set of n elements. This bound has been improved recently to $O(6.4^k n)$ [27]. In the following discussion, we will assume a k-color coding scheme $\mathcal{F}$ of size $O(6.4^k n)$ for a set of n elements.

On a given instance (G, k, K) of the WEIGHTED k-PATH problem, where G is a graph of n vertices, our structure algorithm for WEIGHTED k-PATH produces $h = O(6.4^k n)$ copies $\{G_1, G_2, \ldots, G_h\}$ of the graph G, where each copy G_i is colored by a k-coloring in the k-color coding scheme $\mathcal{F}$. Note that by the definition of k-color coding schemes, every k-path in the graph G has all its vertices colored with different colors in at least one of these copies. The list $\tau_{G,k} = \{G_1, G_2, \ldots, G_h\}$ is the structure returned by the structure algorithm for the WEIGHTED k-PATH problem, whose running time is $O(6.4^k n^2)$.

2. The enumerating algorithm

The enumerating algorithm for WEIGHTED k-PATH is a careful and non-trivial generalization of the dynamic programming algorithm described in [5] which finds a k-path in a k-colored graph. We first discuss how we deal with each copy G_i of the colored graphs in the list $\tau_{G,k}$. We say that a k-path in a k-colored graph is *properly colored* if no two vertices on the path are colored with the same color. Consider the algorithm given in Figure 13, where we have

used $c(w)$ for the color assigned to the vertex w in the k-colored graph G. Inductively, before the j-th execution of the loop 2.1-2.5, we assume that each vertex w is associated with a collection $\mathcal{C}_j(w)$ of pairs (C, P), where C is a subset of j colors in the k-color set, and P is the set of up to K largest properly colored j-paths ending at w that use exactly the colors in C. Then the j-th execution of steps 2.1-2.5 will produce, for each vertex w, a similar collection $\mathcal{C}_{j+1}(w)$ for $(j+1)$-paths in G based on these collections for j-paths.

enumerate-path(G, k, K)
input: a k-colored graph G, and integers k and K
output: the K largest properly colored k-paths ending at each vertex in G
1. **for** each vertex w in G **do** $\mathcal{C}_1(w) = [(\{c(w)\}; \{w\})]$;
2. **for** $j = 1$ **to** $k - 1$ **do**
2.1. **for** each edge $[v, w]$ in G **do**
2.2. **for** each pair (C, P) in $\mathcal{C}_j(v)$ **do**
2.3. **if** $(c(w) \notin C)$ **then**
2.4. construct $|P|$ $(j+1)$-paths ending at w by extending each path in P to the vertex w;
2.5. add these $(j+1)$-paths to P' in the pair $(C \cup \{c(w)\}, P')$ in $\mathcal{C}_{j+1}(w)$ and only keep the K largest $(j+1)$-paths in P';
3. **return** the K largest k-paths in the union of the collections $\mathcal{C}_k(w)$ over all vertices w in G.

Fig. 13. The enumerating algorithm for WEIGHTED k-PATH

Note that at the end of the algorithm **enumerate-path**(G, k, K), for each vertex w in the k-colored graph G, the collection $\mathcal{C}_k(w)$ is either empty or contains a single pair (C, P) where C is the set of all k colors and P is a set of properly colored k-paths ending at w in G.

Lemma D.1 *For each vertex w in the k-colored graph G, the unique pair (C, P) in the collection $\mathcal{C}_k(w)$ constructed by the algorithm* **enumerate-path**(G, k, K) *contains the K largest properly colored k-paths ending at w. Thus, the algorithm returns the K largest*

properly colored k-paths in the graph G. The running time of the algorithm **enumerate-path**(G, k, K) *is* $O(2^k(kn)^2K)$.

PROOF. We prove by induction on j the following claim:

> After the j-th execution of the loop 2.1-2.5, for each vertex w, each pair (C, P) in the collection $\mathcal{C}_{j+1}(w)$ must contain the K largest $(j+1)$-paths ending at w and properly colored by the color subset C.

The claim is obviously true for $j = 0$ because of step 1 of the algorithm. For $j > 0$, note that each p of the K largest $(j+1)$-paths that end at the vertex w and colored properly by the color subset C is a concatenation, by an edge $[v, w]$, of the vertex w and a j-path p' that ends at the vertex v and properly colored by the color subset $C' = C - c(w)$. The j-path p' must be among the K largest j-paths ending at v and properly colored by C' (otherwise, p would not be among the K largest $(j+1)$-paths ending at w and properly colored by C). By the inductive hypothesis, the j-path p' (or a j-path of the same weight) must be contained in the pair (C', P') in the collection $\mathcal{C}_j(v)$. Therefore, when the edge $[v, w]$ is considered in the j-th execution of the loop 2.1-2.5, the path p (or a path of the same weight) will be constructed and included in the pair (C, P) in the collection $\mathcal{C}_{j+1}(w)$.

We must verify that it is not possible that a pair (C, P) in the collection $\mathcal{C}_{j+1}(w)$ contains many copies of the same path so that some other paths among the K largest $(j+1)$-paths are missing in the pair. Inductively, suppose that for the vertex v, all j-paths in P' in the pair $(C - c(w), P')$ in the collection $\mathcal{C}_j(v)$ are distinct. Then when the edge $[v, w]$ is considered in step 2.1, the $(j+1)$-paths constructed in step 2.4 from the j-paths in P' and the vertex w are all different. Moreover, note that any $(j+1)$-path constructed from a collection $\mathcal{C}_j(v)$ and any $(j+1)$-path constructed from a collection $\mathcal{C}_j(v')$, where $v \neq v'$, cannot be the same since the second vertices on the paths are different. This proves that all paths in P in the

pair (C, P) in the collection $\mathcal{C}_{j+1}(w)$ are different.

Since there is only one color subset that contains all the k colors, the above claim implies the first part of the lemma by setting $j = k-1$. Therefore, from the collection $\mathcal{C}_k(w)$ for each vertex w, we get the K largest properly colored k-paths ending at w. Collecting these paths over all vertices in G, we get a set P_0 of $O(Kn)$ properly colored k-paths that obviously contains the K largest properly colored k-paths in G, from which we can find the K largest properly colored k-paths in the graph G in time $O(Kn)$. This is the second part of the lemma.

The complexity of the algorithm is dominated by step 2. Since each of the sets (C, P) and $(C \cup \{c(w)\}, P')$ contains at most K paths, step 2.5 of the algorithm can be executed in time $O(Kk)$. Since each collection $\mathcal{C}_j(v)$ may have up to $\binom{k}{j} = O(2^k)$ j-subsets of colors, and the number of edges in G is bounded by $O(n^2)$, we conclude that the running time of the algorithm is bounded by $O(2^k(kn)^2K)$. □

Since there are $O(6.4^k n)$ k-colored graphs in the list $\tau_{G,k} = \{G_1, G_2, \ldots, G_h\}$, we perform the above process on each of these k-colored graphs. This takes $O(12.8^k k^2 n^3 K)$ time. We get a set P' of $O(6.4^k nK)$ k-paths, each is properly colored in some k-colored graphs in the list $\tau_{G,k}$. Since the k-colorings we used to color the graph vertices come from the k-color coding scheme $\mathcal{F}$, every k-path among the K largest k-paths in G is among the K largest properly colored k-paths in some k-colored graph G_i in the list $\tau_{G,k}$, and hence is contained in the set P'. To find the K distinct largest k-paths in P', we first use BucketSort to sort all k-paths in P' (using the vertex names as ordered along a path as the key for the path). This sorting takes time $O(6.4^k knK)$ and removes duplicate copies of each path in P'. Finally, we find, in time $O(6.4^k nK)$, the K-th largest k-path in the remaining set, which are the K distinct largest k-paths in the graph G. Summarizing this discussion, we conclude with the following theorem.

Theorem D.2 *Given the structure $\tau_{G,k}$ and an integer K, the K largest k-paths in the graph G can be generated in time $O(12.8^k k^2 n^3 K)$.*

Corollary D.3 *The* WEIGHTED k-PATH *problem is fixed parameter linearly enumerable.*

Corollary D.3 may look a bit surprising. Although the k-PATH problem is fixed parameter tractable [5], Flum and Grohe [41] proved that counting the number of k-paths in a graph G is $\#W[1]$-hard. This means that it is unlikely that there is an algorithm of running time $f(k)n^{O(1)}$, where f is a function of k, that can count the number of k-paths in a graph of n vertices precisely. On the other hand, Corollary D.3 shows that enumerating the K largest k-paths in the graph G takes time $f(k)n^{O(1)}K$, where f is a function independent of n. This means that in a feasible amount of average time $f(k)n^{O(1)}$ per path, we can generate the paths in decreasing order of the path weights, which shows that the hardness of the problem of counting the number of k-paths is mainly due to the (possible) large number of such paths in the graph.

E. Effective enumeration based on graph tree decompositions

Graph tree decompositions have played an important role in algorithmic graph theory [10]. More recently, there has been significant research in investigating the concept for developing more efficient exact and parameterized algorithms for graph problems (e.g. [1]). In this section, we discuss how this approach can be used to develop effective structure algorithms for graph problems, and how efficient enumerating algorithms can be achieved based on such structures.

A set D of vertices in a graph G is a *dominating set* of G if every vertex in G is either in D or adjacent to a vertex in D. A dominating set of k vertices will be called a *k-dominating set*. Our running example is the following problem.

WEIGHTED PLANAR DOMINATING SET: given a weighted planar graph G and integers k and K, generate the K smallest k-dominating sets in the graph G.

The problem of deciding whether a given planar graph has a k-dominating set is among the most extensively studied parameterized problems [1]. Flum and Grohe [41] pointed out that counting the total number of k-dominating sets in a planar graph is fixed parameter tractable. On the other hand, no fixed parater tractable algorithm exists that can enumerate all k-dominating sets of a planar graph because the number of such dominating sets can be simply too large to be enumerated in such time.

1. The K smallest elements in a Cartesian Sum

Before we present the structure and enumerating algorithms for WEIGHTED PLANAR DOMINATING SET, we first consider a combinatorial problem, which is also of independent interest.

Let A be a set of n numbers and B be a set of m numbers. The *Cartesian Sum* of A and B, written as $A + B$, is the set $\{a + b \mid a \in A \text{ and } b \in B\}$ of $n \cdot m$ numbers (strictly speaking, $A + B$ is a *multiset* that allows repeated elements). We will say that a pair (a, b), where $a \in A$ and $b \in B$, is an *AB-pair corresponding* to the element $a + b$ in $A + B$. We are interested in finding the K AB-pairs that correspond to the K smallest elements in the Cartesian Sum $A + B$.

We need some notations for our discussion. We say that a list $B = [b_1, b_2, \ldots, b_m]$ is *h-split* if b_h is the h-th smallest number in B, and $b_i \leq b_h$ for all $i < h$ and $b_h \leq b_j$ for all $h < j$. The list B is *semi-sorted* if B is h-split for all $h = 2^q$, where $q = 0, 1, \ldots, \lfloor \log m \rfloor$.

Recall that the following linear time algorithm makes a given set B an h-split list: first find the h-th smallest number w in B in linear time [30], then partition B using w as a "pivot" (this process also gives the h smallest elements in B). For Cartesian Sums, Frederickson and Johnson [44] developed an efficient algorithm that finds the h-th smallest element w in a

Cartesian Sum $A + B$. However, the process of finding the K smallest elements in $A + B$ by simply partitioning $A + B$ using the pivot w will be less efficient: the size of $A + B$ is $n \cdot m$. The following theorem shows how such a partition can be efficiently implemented.

Theorem E.1 *Let A and B be two sets, each consisting of at most K numbers. Then the K AB-pairs corresponding to the K smallest elements in $A + B$ can be constructed in time $O(K)$.*

PROOF. The proof of the theorem is based on the algorithm given in Figure 14. Let A be a set of n numbers and let B be a set of m numbers, where both n and m are bounded by K.

To see the correctness of the algorithm, observe that after the set B is semi-sorted in step 2, B becomes 2^q-split for all q, $0 \leq q \leq \lfloor \log m \rfloor$. Since w is the K-th smallest element in $A + B$, if $a_i + b_{2^q} > w$, no elements b_j in B, where $j \geq 2^q$, can make $a_i + b_j$ among the K smallest elements in $A + B$. Thus, step 4.2 in fact examines all possible AB-pairs that may correspond to any of the K smallest elements in $A + B$. In the following, we study the complexity of the algorithm.

Step 1 of the algorithm takes time $O(K)$ if we use the algorithm by Frederickson and Johnson [44]. To semi-sort the set B in step 2, we first make the set B a $2^{\lfloor \log m \rfloor}$-split list, then, recursively, make the first 2^q elements in B a 2^{q-1}-split list, for each $q = \lfloor \log m \rfloor, \lfloor \log m \rfloor - 1, \ldots, 2, 1$, in this order. As explained above, making a size-t list h-split for any $h \leq t$ takes time $O(t)$. Therefore, to semi-sort the set B in step 2 of the algorithm takes time of the order

$$m + 2^{\lfloor \log m \rfloor} + 2^{\lfloor \log m \rfloor - 1} + \cdots 4 + 2 = O(m) = O(K).$$

The main step, step 4 of the algorithm **partition**, has its running time proportional to the number of times steps 4.2.1-4.2.2 are executed. Note that since q is the smallest

integer such that $a_i + b_{2^q} > w$, we must have $a_i + b_{2^{q-1}} \leq w$. Since the set B is also 2^{q-1}-split, we have $a_i + b_j \leq w$ for all $j \leq 2^{q-1}$. Therefore, for a fixed a_i, at least half of the $2^q - 1$ executions of steps 4.2.1-4.2.2 generate AB-pairs corresponding to elements among the K smallest elements in $A + B$. In conclusion, the total number of executions of steps 4.2.1-4.2.2 is at most twice of the number of AB-pairs generated by the algorithm. Since the algorithm stops when K AB-pairs are generated, the total time spent by step 4 of the algorithm **partition** is bounded by $O(m+n+K) = O(K)$. This completes the proof of the theorem. □

partition(A, B, K)
input: an integer K, and two sets A and B, each consists of at most K numbers
output: K AB-pairs corresponding to the K smallest elements in $A + B$

1. find the K-th smallest element w in $A + B$;
2. semi-sort the set B, let the semi-sorted list be $B = [b_1, b_2, \ldots, b_m]$;
3. $K_0 = 0$;
4. **for** $i = 1$ **to** n **do** (*suppose $A = [a_1, a_2, \ldots, a_n]$.*)
4.1. find the smallest q such that $a_i + b_{2^q} > w$;
4.2. **for** $j = 1$ **to** $2^q - 1$ **do**
4.2.1 **if** $a_i + b_j \leq w$ **then** output (a_i, b_j); $K_0 = K_0 + 1$;
4.2.2 **if** $(K_0 = K)$ **then** stop.

Fig. 14. Finding the K smallest elements in a Cartesian set

2. The structure algorithm

To describe the structure algorithm for WEIGHTED PLANAR DOMINATING SET, we review some related terminologies. For more detailed discussions on this topic, the reader is referred to [10].

Definition Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(\mathcal{V}, \mathcal{T})$ where

$\mathcal{V}$ is a collection of subsets of V such that $\bigcup_{X_i \in \mathcal{V}} = V$, and $\mathcal{T}$ is a tree whose node set is $\mathcal{V}$, such that:

1. for every edge $[u, v] \in E$, there is an $X_i \in \mathcal{V}$, such that $\{u, v\} \subseteq X_i$;

2. for all $X_i, X_j, X_k \in \mathcal{V}$, if the node X_j lies on the path between the nodes X_i and X_k in the tree $\mathcal{T}$, then $X_i \cap X_k \subseteq X_j$.

The *width* of the tree decomposition $(\mathcal{V}, \mathcal{T})$ is defined to be $\max\{|X_i| \mid X_i \in \mathcal{V}\} - 1$. The *treewidth* of the graph G is the minimum tree width over all tree decompositions of G.[2]

For a given graph of treewidth k, a tree decomposition of width k for G can be constructed in time $f(k)n$, where the function value $f(k)$ is very large even for small values of k [10]. Alternatively, for planar graphs that have k-dominating sets, tree decompositions of small width can be constructed using more practical algorithms, as given in the following theorem [1].

Theorem E.2 ([1]) *If a planar graph G of n vertices has a k-dominating set, then a tree decomposition of treewidth $O(\sqrt{k})$ and $O(n)$ nodes for G can be constructed in time $O(\sqrt{k}n)$.*

A tree decomposition $(\mathcal{V}, \mathcal{T})$ is *nice* if it satisfies the following conditions:

1. Each node in the tree $\mathcal{T}$ has at most two children;
2. If a node X_i has two children X_j and X_k in the tree $\mathcal{T}$, then $X_i = X_j = X_k$;
3. If a node X_i has only one child X_j in the tree $\mathcal{T}$, then either $|X_i| = |X_j| + 1$ and $X_j \subset X_i$, or $|X_i| = |X_j| - 1$ and $X_i \subset X_j$.

Theorem E.3 ([68]) *There is a linear time algorithm that, for a given tree decomposition of treewidth h and n nodes for a graph G, constructs a nice tree decomposition of treewidth h and $O(n)$ nodes for the graph G.*

[2]To avoid confusion, we will use "nodes" for the trees in tree decompositions, and use "vertices" for the underlying graphs.

Therefore, for an instance (G, k, K) of WEIGHTED PLANAR DOMINATING SET, we first call the algorithms in Theorem E.2 and Theorem E.3 on the graph G. If the algorithms do not return a desired tree decomposition, then G has no k-dominating set. Otherwise, the returned nice tree decomposition $(\mathcal{V}, \mathcal{T})$ is the structure $\tau_{G,k}$ used for our enumerating algorithm, which has $O(n)$ nodes and of treewidth $h = O(\sqrt{k})$, and can be constructed in time $O(\sqrt{k}n)$.

3. The enumerating algorithm

Let (G, k, K) be an instance of WEIGHTED PLANAR DOMINATING SET, where $G = (V, E)$ is a weighted and planar graph of n vertices. Let $\tau_{G,k} = (\mathcal{V}, \mathcal{T})$ be a nice tree decomposition of G with $O(n)$ nodes and treewidth $h = O(\sqrt{k})$, which is the structure produced by the structure algorithm in the previous subsection. In this subsection we show how the structure $\tau_{G,k}$ can be used to enumerate the K smallest k-dominating sets in the graph G.

Let $X_i = \{v_1, \ldots, v_q\}$ be a node in the tree $\mathcal{T}$, where each v_j is a vertex in the graph G. In the following discussion in this subsection, we will always denote by Y_i the set of vertices in G that are contained in any node in the subtree rooted at X_i in the tree $\mathcal{T}$. For a given subset D of Y_i, we assign each vertex v_j in X_i a value $c(v_j)$ according to its relation to D, as follows:

1. $c(v_j) = 1$ if v_j is in D;
2. $c(v_j) = -1$ if v_j is not in D but is adjacent to a vertex in D;
3. $c(v_j) = 0$ if v_j is neither in D nor adjacent to any vertex in D.

With these values, we say that the set D and the value assignment $A = [c(v_1), \ldots, c(v_q)]$ to X_i are *consistent*. Note that there can be many subsets of Y_i that are consistent with the same value assignment A to X_i. For a value assignment A to X_i and an integer $r \leq k$, a subset D of Y_i is an (A, r)-*subset* of Y_i if (1) D has exactly r vertices, (2) D is consistent

with the value assignment A, and (3) for each vertex w in $Y_i - X_i$, either w is in D or w is adjacent to a vertex in D. Intuitively, an (A, r)-subset D is a potential set of r vertices that is part of a k-dominating set D_0 for the graph G such that $D_0 \cap Y_i = D$.

For the node X_i that contains q vertices in the graph G, there are 3^q possible value assignments to X_i. For each value assignment A to X_i, we attach to A a collection of $k+1$ lists $\mathcal{L}_A = [L_0, L_1, \ldots, L_k]$, where L_r is a list containing the K smallest (A, r)-subsets of Y_i (the collection $\mathcal{L}_A$ will be called the *spectrum* of A). Observe that since no vertex w in $Y_i - X_i$ can be adjacent to any vertex not in Y_i, the selection of the vertices in a dominating set from the set $V - Y_i$ will be totally independent of the status of w, but may (only) depend on the status of the vertices in X_i. Therefore, if in each list L_r we simply record the K smallest (A, r)-subsets of Y_i that are consistent with the value assignment A, then for the k-dominating sets of G consistent with the value assignment A, only these (A, r)-subsets of Y_i can make the K smallest k-dominating sets with vertices in $V - Y_i$.

Using dynamic programming, we can construct for each node X_i in the tree $\mathcal{T}$ all the valid value assignments to X_i, and for each valid value assignment A to X_i, we construct the corresponding spectrum $\mathcal{L}_A$. We proceed from the leaves of the tree $\mathcal{T}$ in a bottom-up manner. For each leaf X_i of q vertices, we construct each of the 3^q value assignments to X_i. Note that in this case, $Y_i = X_i$, so it is fairly easy to determine if a value assignment is valid, and for each valid value assignment A, the spectrum $\mathcal{L}_A = [L_0, L_1, \ldots, L_k]$ can be directly constructed.

Now we discuss how the induction proceeds. Suppose that the value assignments and the corresponding spectra have been constructed for all children of a node X_i in the tree $\mathcal{T}$. To construct the value assignments and the corresponding spectra for the node X_i, we distinguish three different cases.

Case 1. X_i has a single child X_j, $|X_j| = |X_i| - 1$, and $X_j \subset X_i$.

Let $v \in X_i - X_j$, then $v \notin Y_j$. For each value assignment A_j to X_j, we can get three different value assignments for X_i by assigning $c(v) = -1$, 0, and 1, respectively. Since v is not adjacent to any vertex in $Y_j - X_j$, it is easy to check the validity of these value assignments. For example, if we assign $c(v) = 1$, then any vertex w in X_i that is adjacent to v in G cannot have value $c(w) = 0$. To construct the corresponding spectrum $\mathcal{L}_{A_i}$ for a valid value assignment A_i for X_i, suppose that $c(v) = 1$ and that A_i is obtained from a value assignment A_j for X_j. Then each (A_j, r)-subset in the spectrum $\mathcal{L}_{A_j}$ plus the vertex v becomes an $(A_i, r+1)$-subset in the spectrum $\mathcal{L}_{A_i}$. The cases for the other values of $c(v)$ can be handled similarly. Finally, we remove the larger (A, r)-subsets from a list L_r in $\mathcal{L}_{A_i}$ if the list contains more than K subsets.

Case 2. X_i has a single child X_j, $|X_j| = |X_i| + 1$, and $X_i \subset X_j$.

Let $v \in X_j - X_i$, then any value assignment to X_j with the value $c(v)$ dropped makes a value assignment to X_i. Again, we can check the validity of these value assignments to X_i. For example, if $c(v) = 0$ in a value assignment A_j to X_j, then A_j with $c(v)$ dropped does not induce a valid value assignment A_i to X_i because no (A_j, r_j)-subset for any r_j in the spectrum $\mathcal{L}_{A_j}$ is a valid (A_i, r_i)-subset for any r_i in Y_i: the vertex v in $Y_i - X_i$ is neither contained in nor adjacent to the (A_j, r_j)-subset. If A_j to X_j with $c(v)$ dropped induces a valid value assignment A_i to X_i, then each (A_j, r)-subset in $\mathcal{L}_{A_j}$ becomes an (A_i, r)-subset in $\mathcal{L}_{A_i}$. Again we need to remove the larger (A_i, r)-subsets if the list L_r in $\mathcal{L}_{A_i}$ contains more than K subsets.

Case 3. X_i has two children X_j and X_h and $X_j = X_i = X_h$.

We say that a value assignment A_j to X_j and a value assignment A_h to X_h are "mergeable" if for any $v \in X_j = X_h$, either v has the same value in A_j and in A_h, or one of A_j and A_h assigns v value -1 and the other assigns v value 0. A value assignment A_i to X_i is obtained from the two mergeable value assignments A_j and A_h such that the value of v in

A_i is equal to that of v in both A_j and A_h (when the values are equal) or equal to -1 (if the values are not equal).

A valid value assignment A_i to X_i has in its spectrum $\mathcal{L}_{A_i}$ a collection of (A_i, r_i)-subsets in Y_i, where $0 \leq r_i \leq k$. Since $Y_i = Y_j \cup Y_h$, each (A_i, r_i)-subset D_i in $\mathcal{L}_{A_i}$ is a union of a subset D_j in Y_j and a subset D_h in Y_h. Let A_j and A_h be the value assignments to X_j and to X_h that are consistent with D_j and D_h, respectively. Then from $D_i = D_j \cup D_h$, $D_j \cap D_h \subseteq X_i$, and $(Y_j - X_i) \cap (Y_h - X_i) = \emptyset$, it is not difficult to verify that (1) D_j is an (A_j, r_j)-subset for some r_j in Y_j; (2) D_h is an (A_h, r_h)-subset for some r_h in Y_h; and (3) A_j and A_h are mergeable. In particular, this means that the value assignment A_i to X_i can be obtained from the two mergeable (and valid) value assignments A_j and A_h. In consequence, by examining all possible mergeable value assignments to X_j and to X_h, we will construct the value assignment A_i to X_i. Moreover, let $X_i^1 = D_i \cap X_i$, then $D_i = D_j \cup D_h = (D_j - X_i^1) \cup (D_h - X_i^1) \cup X_i^1$. Thus, the weight of D_i is equal to the weight of D_j plus the weight of D_h minus the weight of X_i^1. Therefore, D_j must be among the K smallest (A_j, r_j)-subsets in Y_j (otherwise D_i would not be in the spectrum $\mathcal{L}_{A_i}$), i.e., D_j must be contained in the list L_{r_j} in the spectrum $\mathcal{L}_{A_j}$. Similarly, D_h must be contained in the list L_{r_h} in the spectrum $\mathcal{L}_{A_h}$. In summary, by examining all possible unions of the subsets in the spectra $\mathcal{L}_{A_j}$ and $\mathcal{L}_{A_h}$, the (A_i, r_i)-subset D_i (or an (A_i, r_i)-subset of the same weight) in the spectrum $\mathcal{L}_{A_i}$ can be re-constructed.

Since A_i is an arbitrary valid value assignment to X_i and D_i is an arbitrary (A_i, r_i)-subset in the spectrum $\mathcal{L}_{A_i}$, we conclude that by examining all pairs of mergeable (and valid) value assignments A_j and A_h (to X_j and X_h, resp.), and by examining for each such a pair A_j and A_h all possible unions of the subsets in $\mathcal{L}_{A_j}$ and $\mathcal{L}_{A_h}$, we can construct all valid value assignments to X_i and their corresponding spectra. This completes the proof of the induction for Case 3.

In conclusion, by the above dynamic programming process, starting from the leaves

of the tree $\mathcal{T}$, we can correctly construct the value assignments and their spectra for each node in the tree $\mathcal{T}$, in particular, for the root node X_0 of the tree $\mathcal{T}$. Note that for a value assignment A_0 to X_0, in order for an (A_0, k)-subset in $\mathcal{L}_{A_0}$ to be a k-dominating set for the graph G, it is sufficient and necessary that the assignment A_0 does not assign value 0 to any vertex in X_0. Thus, for each value assignment A_0 that does not assign value 0 to any vertex in X_0, the list L_k in the spectrum $\mathcal{L}_{A_0}$ contains the K smallest k-dominating sets for the graph G that are (A_0, k)-subsets. Since every k-dominating set of the graph G must be consistent with a value assignment to X_0, by examining all value assignments to X_0 and their spectra, we will be able to construct the K smallest k-dominating sets of the graph G.

Lemma E.4 *Given a tree decomposition $(\mathcal{V}, \mathcal{T})$ of treewidth q and N nodes for a weighted graph G, and integers k and K, the K smallest k-dominating sets of the graph G can be generated in time $O(9^q k^3 NK)$.*

PROOF. The discussion given above describes a dynamic programming process that generates the K smallest k-dominating sets of the graph G when the tree decomposition $(\mathcal{V}, \mathcal{T})$ is given. What remains is to analyze the complexity of this process.

The most complicated case in the dynamic programming is Case 3, in which a node X_i in the tree $\mathcal{T}$ has two children X_j and X_h. We analyze the complexity for this case in detail. The other two cases are simpler and have their running time dominated by that of Case 3.

Suppose that a value assignment A_j to X_j and a value assignment A_h to X_h are mergeable and merged into a value assignment A_i to X_i. From a list L_{r_j} in $\mathcal{L}_{A_j}$ and a list L_{r_h} in $\mathcal{L}_{A_h}$, we need to identify the K smallest unions of the form $D_i = D_j \cup D_h$, where D_j is an (A_j, r_j)-subset in L_{r_j} and D_h is an (A_h, r_h)-subset in L_{r_h}. Let $X_i^1 = D_i \cap X_i$. Then

$$\text{weight}(D_i) = \text{weight}(D_j) + \text{weight}(D_h) - \text{weight}(X_i^1)$$

Since X_i^1 is fixed when A_i is given, finding these K smallest unions is equivalent to finding the K smallest elements in the Cartesian Sum $W_j + W_h$, where W_j is the set of the weights of the (A_j, r_j)-subsets in L_{r_j} and W_h is the set of the weights of the (A_h, r_h)-subsets in L_{r_h}. Since each of L_{r_j} and L_{r_h} contains at most K subsets, by Theorem E.1, these K smallest unions can be identified in time $O(K)$, and constructed in time $O(kK)$ (since each of the subsets in L_{A_j} and L_{A_h} contains at most k vertices, the union of such two sets can be constructed in time $O(k)$). Finally, note that updating the list L_{r_i} in $\mathcal{L}_{A_i}$, where $r_i = r_j + r_h - |X_i^1|$, with these K new (A_i, r_i)-subsets, i.e., picking the K smallest from the current subsets in L_{r_i} and these new subsets, can also be done in time $O(kK)$.

Therefore, updating a list in the spectrum of a value assignment to X_i by the set unions from two lists in the spectra of two mergeable value assignments to X_j and X_h takes time $O(kK)$. Each spectrum of a value assignment to the nodes X_j and X_h has at most $k+1$ lists, and each of the nodes X_j and X_h has at most 3^q value assignments (since the treewidth of $\mathcal{T}$ is q). Since the dynamic programming process examines all unions of subsets from all possible pairs of lists in the spectra of all mergeable value assignments to X_j and X_h, the process takes time $O(9^q k^3 K)$ to proceed from the nodes X_j and X_h to to node X_i in the tree $\mathcal{T}$. The lemma now follows since the tree $\mathcal{T}$ has N nodes. □

We conclude with our main result in this section.

Theorem E.5 *The* WEIGHTED PLANAR DOMINATING SET *problem is fixed parameter linearly enumerable. More specifically, for an instance (G, k, K) of the problem, a structure $\pi_{G,k} = (\mathcal{V}, \mathcal{T})$ can be constructed in time $O(\sqrt{k}n)$, and from the structure $\pi_{G,k}$, the K smallest k-dominating sets of the planar graph G can be generated in time $O(2^{O(\sqrt{k})}k^3 nK)$.*

PROOF. The conclusion for the structure $\pi_{G,k} = (\mathcal{V}, \mathcal{T})$ follows directly from Theorem E.2

and Theorem E.3, which also claim that the tree $\mathcal{T}$ in the structure $\pi_{G,k}$ has $O(n)$ nodes and treewidth $O(\sqrt{k})$. The last conclusion then follows from this fact and Lemma E.4. $\square$

F. Final remarks

We have introduced the concept of effective fixed parameter enumerability of NP optimization problems. Our objective is to solve enumeration problems that seem to have an increasing demand in recent research in computational science. We split the task of solving an enumeration problem into two stages: the structure stage and the enumerating stage. We showed that many popular techniques developed in parameterized algorithms can be modified and enhanced to provide effective algorithms for the structure stage. We developed new algorithms for the enumerating stage that exploit the structure produced by the structure stage to enumerate the desired number of best solutions efficiently.

Further investigation on the relationship between fixed parameter tractability and fixed parameter enumerability may open up an interesting research direction. It is quite natural that fixed parameter enumerability implies fixed parameter tractability (generating a number of best solutions cannot be easier than checking the existence of a single solution). We showed that the converse is not true in general by exhibiting an example of a problem which is fixed parameter tractable but not fixed parameter enumerable (under the parameterized complexity hypothesis). On the other hand, our study shows that most techniques used in fixed parameter tractability are also applicable for fixed parameter enumerability. Studying the relationship between these two classes seems interesting and important, from both theoretical and practical points of view.

We finally indicate that even though we illustrated our results by picking specific problems for each technique, each of the considered problems is a representative for a large set of problems to which the techniques are applicable as well.

CHAPTER VI

SUMMARY AND FUTURE RESEARCH

In this dissertation, we present kernels for several parameterized problems, and propose a framework to study enumerability. The results in the dissertation not only improve the previous known results, but are enlightening for the future research.

A. Summary

We present kernels for several parameterized problems. The CLUSTER EDITING problem arises from biological research, where researchers are looking for "similar" genes. In algorithmic research, the instances of the problem are modeled as graphs, where vertices correspond to the genes and two vertices are connected if the genes they represent are similar. The ideal instance is a graph consisting of a union of disjoint cliques, however, the practical instances might have "errors", so the graphs contain other edges. The CLUSTER EDITING problem is focusing on fixing the errors by removing least number of edges in the graphs. The problem is fix-parameter tractable and admits a $4k$ kernel size previously. We improve the result and develop a $2k$ kernel for the problem.

We also study the d-CLUSTER EDITING problem, a variant of the CLUSTER EDITING problem. We provide a $7k + 2d$ kernel and develop the first fix-parameter tractable algorithm for this problem. The difference from its original version is that after applying the edge operations in the solutions, the resulting graphs contain exactly d clusters. However the solutions with least edge insertion/deletion operations may not result in d clusters in the resulting graph, and are not valid. We introduce the *class-partition* to overcome this difficulty. We show that there are two possible cases: either we combine some of the classes in some class-partitions to decrease the number of clusters, or we split classes in some class-

partitions to increase the number of clusters. In both cases, we present smaller instances and combining both instances we obtain a kernel containing no more than $7k + 2d$ vertices. We also design a branch-and-search algorithm to enumerate all class-partitions, and apply combinatorial combinating/spliting algorithms on the class-partitions to find the optimal solutions to the d-CLUSTER EDITING problem.

The PSEUDO-ACHROMATIC NUMBER problem is a variance of the GRAPH COLORING problem. An equivalent definition of the problem is that given an input graph, it asks if we can "group" vertices in the graph to convert the graph to a complete graph with k vertices. By grouping vertices, basically we merge vertices, which may not be necessarily adjacent. Previously no results in parameterized complexity is known for this problem, we present a quadratic kernel for this problem and it implies that the problem is in FPT. We also study the more general problem, the VERTEX GROUPING problem: the input consists of two graphs, G and H, it asks if one can apply a set of vertex grouping operations on G to convert G to a new graph with $|V(G)|$ vertices which is isomorphic to H. We show that the general problem is fix-parameter intractable.

We present a framework to study fix-parameter enumerability of the parameterized problems, the approach is of general interest. We prove that the class FPE is a proper subset of FPT, to further explore the relation between them and illustrate the approach, we study three classical parameterized problems, each of which can be solved efficiently by a typical algorithm-design technique for parameterized problems, i.e. branch-and-search, color coding and treewidth. Specifically, we show that the Z smallest k-vertex covers of the instance graphs of the WEIGHTED VERTEX COVER problem can be generated in time $O(1.47^k n + 1.22^k nZ)$; For the WEIGHTED k-PATH problem, the Z largest k-paths in the graph can be generated in time $O(12.8^k k^2 n^3 Z)$; And for the WEIGHTED PLANAR DOMINATING SET problem, the Z smallest k-dominating sets of the planar graph can be generated in time

$O(2^{O(\sqrt{k})}k^3nZ)$.

We note that the runtime of the algorithms we have shown is linear in Z, the number of desired solutions. Thus the solutions to the problems can be generated in time $f(k)n^{O(1)}$ per solution; we call this kind of problems *fix-parameter linearly enumerable*, it extends the definition of fix-parameter enumerability of parameterized problems. Readers may find similar definition "polynomial time delay" for the problems in P, but our approach is to study the enumerability for NP-Hard problems.

B. Open problems

In the section, we describe the future research topic and open problems in the dissertation.

1. The CLUSTER EDITING problems

In biological research, an important task is to find "similar" genes. The corresponding problem in algorithmic research is the GENE CLUSTERING problem. In the ideal case, the graph model contains a union of disjoint cliques. However due to the difficulty in biological research, we can merely obtain the data expressed by a graph which are "close" to disjoint cliques. To correct the errors, researchers study the CLUSTER EDITING problem. This model still has limitations, since the clusters are well structured, to apply only k edge insertion/deletion operations may not be enough to correct all the "errors". Different approaches are proposed, for example, Guo [54] present the s-PLEX EDITING problem. In practice the quality of data could be very poor so that the graph models are far away from disjoint cliques. Motivated by this, we relax the requirement for clusters: the resulting graph contains a union of special structures instead of disjoint cliques, which are "almost" complete. A concrete model is s-defective cluster. A *s-defective cluster* is a complete graph except that there are at most s edges missing in the graph. Up to now no kernelization algorithm and fix-parameter

tractable algorithm are known for this model. A more general model is that we skip the "errors" inside each structure, but focus on the inter-structure "errors", providing that the structure are really dense graphs. Another variance is to allow certain edges with zero cost, i.e. the cost to insert/delete the edges is ignored, the CLUSTER EDITING problem with some "don't care" edges are unknown about its fix-parameter tractability.

In our kernelization algorithm, *critical clique* play an important role. The interesting part about critical cliques is that vertices in the same critical clique have the same set of neighbors besides themselves. A generalization of the idea is *modular decomposition*, [57] the basic definition is as follows. Two sets E and F are *overlap* if $E \bigcap F$ is not empty, and both $E - F$ and $F - E$ are not empty. A subset M of V is a *module* if for any $v \in (V - M)$, M does not overlap with neighbors of v: either v connects to all of the vertices in M or v connects to none. Every set with one or all vertices is a *trivial* module, a graph with no non-trivial module is *prime*, and a *strong module* is a module that does not overlap with any other modules. The critical clique is one special case in modular decomposition. Using the idea of modular decomposition, we might be able to develop better kernel for the CLUSTER EDITING problem, and provide different approaches to overcome the difficult of the poor quality of data.

We study the d-CLUSTER EDITING problem, a variance of the CLUSTER EDITING problem. We present a linear kernel for this problem, the kernel size is linear in both k and d. One difficulty in developing kernelization algorithm for the d-CLUSTER EDITING problem is that it asks that the resulting graph contains exactly d disjoint cliques. We introduce the idea of *class-partition* to overcome the difficulty and design a kernel and a fix-parameter tractable algorithm. An open problem is that if we can develop a kernel whose size only depends on k, and similarly, can we design a fix-parameter tractable algorithm with a solo parameter k for this problem?

2. The GRAPH COLORING problems

The PSEUDO-ACHROMATIC NUMBER problem is a variance of the GRAPH COLORING problem. The *vertex grouping* operation, which is defined in the VERTEX GROUPING problem, can merge any vertices, even they are not adjacent. It is different from the *edge contraction* operation in the graph minor theory, where only adjacent vertices can be merged. The VERTEX GROUPING problem is fix-parameter intractable. On the other hand, its special case, the PSEUDO-ACHROMATIC NUMBER problem, admits a quadratic kernel, it implies its feasibility in term of the parameterized complexity.

To author's knowledge, no non-trivial fix-parameter tractable algorithms is known for the PSEUDO-ACHROMATIC NUMBER problem, even its instances can be reduced to smaller instances with at most k^2 vertices. The trivial algorithm simply enumerates all possible ways to partition vertices to k groups, merge vertices in each group, then check if the resulting graph is complete. The runtime of the trivial algorithm is $O(k^{k^2})$. To design non-trivial algorithms with runtime $O(c^{k^2})$ is still an open problem, further investigation on the property of the problem can help to develop better algorithms. For example, we can apply the color coding approach, coloring the edges of the input instances with k^2 colors takes c^{k^2} time, but it is unknown that how to identify the useful edges.

A variance of the color coding technique is proposed by Alon et al, the algorithm in the paper [2] illustrates the new approach. The basic idea is that given a graph G with k edges, we randomly color vertices in G with $O(\sqrt{k})$ colors. With probability $c^{-\sqrt{k}}$, the endpoints of any edges in G are colored by different colors, where c is some constant; And the randomized approach can be de-randomized with slightly increasing on the running time. The approach shows a new way to color edges in the given graph, previously it is hard to construct a nice structure after coloring edges directly. By coloring vertices with less colors, the resulting graph reserves all the edges we are interested in. We don't know how to apply the idea to

the PSEUDO-ACHROMATIC NUMBER problem, it is still open whether we can develop efficient parameterized algorithms with this approach.

3. Parameterized enumerability

Enumerability is a well-studied topic in the classical complexity research, but the previous approach [45] is targeting the problems in P. In the theory of parameterized complexity, we study the enumerability of the fix-parameter tractable problems. We show that FPE is a subset of FPT by providing a concrete problem which is in FPT, but not enumerable in term of parameterized complexity.

Many algorithms for the NP-Hard problems essentially enumerate all solutions to find the optimal. Motivated by this observation, we study three popular algorithm-design techniques, and show that with little revision, the three techniques can be adapted to structural algorithms and produce certain structures; Then we apply elegant enumeration techniques on the structures to efficiently enumerate solutions to the parameterized problems. Specifically, we define *fix-parameter linearly enumerable* problems, solutions to which could be enumerated in time $f(k)n^{O(1)}$ per solution.

With rapid developments on algorithm-design techniques, it is not possible to list all techniques in this dissertation. We design three enumeration algorithms, surely it is possible to further explore the design of fix-parameter enumerable algorithms. Kernelization algorithm has attracted much attention recently, we know that problems in FPT admit kernelization algorithms, it is unknown if we can adapt kernelization to the design of fix-parameter enumeration algorithm. One important strategy in kerneliation is to repeatedly reduce the size of the instances, it is possible that some sub-optimal solutions are removed. It would be interesting to develop kernels for fix-parameter enumerable problems and to explore the relation between kernelization algorithms and enumeration algorithms .

REFERENCES

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for Dominating Set and related problems on planar graphs, Algorithmica 33(4) (2002) 461 - 493.

[2] N. Alon, D. Lokshtanov, S. Saurabh, Fast FAST, 36th International Colloquium on Automata, Languages and Programming (1) (2009) 49 - 58.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, M. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation, Springer, Berlin, New York, 1999.

[4] V. Arvind, V. Raman, Approximation algorithms for some parameterized counting problems, Lecture Notes in Computer Science 2518 (2002) (ISAAC2002) 453 - 464.

[5] N. Alon, R. Yuster, U. Zwick, Color-coding, Journal of the ACM 42(4) (1995) 844 - 856.

[6] A. Alizadeh, M. Eisen, R. Davis, C. Ma, I. Lossos, A. Rosenwald, J. Boldrick, H. Sabet, T. Tran, X. Yu, J. Powell, L. Yang, G. Marti, T. Moore, J. Hudson Jr, L. Lu, D. Lewis, R. Tibshirani, G. Sherlock, W. Chan, T. Greiner, D. Weisenburger, J. Armitage, R. Warnke, R. Levy, W. Wilson, M. Grever, J. Byrd, D. Botstein, P. Brown, L. Staudt, Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling, Nature 403 (2000) 503 - 511.

[7] R. Balasubramanian, V. Raman, V. Yegnanarayanan, On the pseudoachromatic number of join of graphs, International Journal of Computer Mathematics, 80(9) (2003) 1131 - 1137.

[8] V. Bhave, On the pseudoachromatic number of a graph, Fundamenta Mathematicae 102(3) (1979) 159 - 164.

[9] H. L. Bodlaender, Achromatic number is NP-complete for cographs and interval graphs, Information Processing Letters 32(3) (1989) 135 - 138.

[10] H. L. Bodlaender, Treewidth: Algorithmic techniques and results, Mathematical Foundations of Computer Science 1997 (1997) 19 - 36.

[11] S. Böcker, S. Briesemeister, Q. B. A. Bui A. Truss, Going weighted: Parameterized algorithms for cluster editing, Theoretical Computer Science 410(52) (2009) 5467 - 5480.

[12] A. Bjorklund, T. Husfeldt, P. Kaski, M. Koivisto, Fourier meets mobius: Fast subset convolution, 39th ACM Symposium on Theory of Computing (2007) 67 - 74.

[13] H. L. Bodlaender, Kernelization: New Upper and Lower Bound Techniques, International Workshop on Exact and Parameterized Computation (2009) 17 - 37.

[14] H. L. Bodlaender, R. G. Downey, M. R. Fellows D. Hermelin, On problems without polynomial kernels, Journal of Computer and System Sciences 75(8) (2009) 423 - 434.

[15] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, D. M. Thilikos, (Meta) kernelization, 50th Annual IEEE Symposium on Foundations of Computer Science (2009).

[16] Y. Bu, O. T.-W. Leung, A. W.-C. Fu, E. J. Keogh, J. Pei, S. Meshkin, WAT: Finding top-K discords in time series database, Proceedings of the Seventh SIAM International Conference on Data Mining (2007).

[17] B. Bollobás, B. Reed, A. Thomason, An extremal function for the achromatic number, Graph Structure Theory 1991 (2005) 161 - 166.

[18] A. Ben-dor, R. Shamir, Z. Yakhini, Clustering gene expression patterns, Journal of Computational Biology 6(3-4) (1999) 281 - 297.

[19] L. Cai, J. Chen, On fixed-parameter tractability and approximability of NP optimization problems, Journal of Computer and System Sciences 54 (1997) 465 - 474.

[20] M. Charikar, V. Guruswami, A. Wirth, Clustering with qualitative information, Journal of Computer and System Science 71(3) (2005) 360 - 383.

[21] C. R. Chegireddy, H. W. Hamacher, Algorithms for finding K-best perfect matchings, Discrete Applied Mathematics 18 (1987) 155 - 165.

[22] J. Chen, I. Kanj, W. Jia, Vertex cover: Further observations and further improvements. Journal of Algorithms 41 (2001) 280 - 301.

[23] J. Chen, H. Fernau, I. A. Kanj, and G. Xia, Parametric duality and kernelization: Lower bounds and upper bounds on kernel size, SIAM Journal on Computing 37 (2007) 1077 - 1106.

[24] J. Chen, I. A. Kanj, G. Xia, Improved parameterized upper bounds for vertex cover. Lecture Notes in Computer Science 4162 (2006) (MFCS 2006) 238 - 249.

[25] J. Chen, X. Huang, I. Kanj, G. Xia, Strong computational lower bounds via parameterized complexity, Journal of Computer and System Sciences 72(8) (2006) 1346 - 1367.

[26] J. Chen, F. V. Fomin, Y. Liu, S. Lu, Y. Villanger, Improved algorithms for feedback vertex set problems, Journal of Computer and System Sciences 74(7) (2008) 1188 - 1198.

[27] J. Chen, S. Lu, S. Sze, F. Zhang, Improved algorithms for path, matching, and packing problems, ACM-SIAM Symposium on Discrete Algorithms (2007) 298 - 307.

[28] Z. Chen, T. Jiang, G. Lin, Computing phylogenetic roots with bounded degrees and errors, SIAM Journal on Computing 32(4) (2003) 864 - 879.

[29] S. Chien, A determinant-based algorithm for counting perfect matchings in a general graph, ACM-SIAM Symposium on Discrete Algorithms (2004) 728 - 735.

[30] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, McGraw-Hill Book Company, Boston, MA, 2001.

[31] V. Dahllöf, P. Jonsson, An algorithm for counting maximum weighted independent sets and its applications, ACM-SIAM Symposium on Discrete Algorithms (2002) 292 - 298.

[32] F. Dehne, M. Langston, X. Luo, S. Pitre, P. Shaw, Y. Zhang, The cluster editing problem: implementations and experiments, Lecture Notes in Computer Science 4169 (2006) (IWPEC 2006) 13 - 24.

[33] M. Dom, J. Guo, F. Huffner, R. Niedermeier, Extending the tractability border for closest leaf powers, Lecture Notes in Computer Science 3787 (2005) (WG 2005) 397-408.

[34] R. Downey, M. Fellows, U. Stege, Parameterized complexity: A framework for systematically confronting computational intractability, in Contemporary Trends in Discrete Mathematics, (R. Graham, J. Kratochvíl, J. Nešetřil, and F. Roberts eds.), Proceedings of DIMACS-DIMATIA Workshop, Prague 1997, AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science 49 (1999) 49 - 99.

[35] R. Downey, M. Fellows, Parameterized Complexity, Springer-Verlag, New York, 1999.

[36] M. E. Dyer, Approximate counting by dynamic programming, 35th ACM Symposium on Theory of Computing (2003) 693 - 699.

[37] K. Edwards, C. McDiarmid, The complexity of harmonious coloring for trees, Discrete Applied Mathematics 57 (1995) 133 - 144.

[38] M. Fellows, The lost continent of polynomial time: Preprocessing and kernelization, Lecture Notes in Computer Science 4169 (2006) (IWPEC 2006) 276 - 277.

[39] M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. A. Rosamond, U. Stege, D. M. Thilikos, S. Whitesides, Faster fixed-parameter tractable algorithms for matching and packing problems, Lecture Notes in Computer Science 3221 (2004) (ESA 2004) 311 - 322.

[40] M. Fellows, M. Langston, F. Rosamond, P. Shaw, Efficient parameterized preprocessing for cluster editing, Lecture Notes in Computer Science 4639 (2007) (FCT 07) 312 - 321.

[41] J. Flum, M. Grohe, The parameterized complexity of counting problems, SIAM Journal on Computing 33(4) (2004) 892 - 922.

[42] H. Fernau, On parameterized enumeration, Lecture Notes in Computer Science 2387 (2002) (COCOON 2002) 564 - 573.

[43] L. Fortnow, R. Santhanam, Infeasibility of instance compression and succinct PCPs for NP, 40th ACM Symposium on Theory of Computing (2008) 133 - 142.

[44] G. Frederickson, D. Johnson, The complexity of selection and ranking in X + Y and matrices with sorted columns, Journal of Computer and System Sciences 24(2) (1982) 197 - 208.

[45] K. Fukuda, T. Matsui, Y. Matsui, Algorithms for combinatorial enumeration problems, available at : http://dmawww.epfl.ch/roso.mosaic/kf/enum/comb/combenum.html, 1996.

[46] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, W. H. and Company, New York, 1979.

[47] J. Gramm, J. Guo, F. Huffner, R. Niedermeier, Automated generation of search tree algorithms for hard graph modification problems, Algorithmica 39(4) (2004) 321 - 347.

[48] J. Gramm, J. Guo, F. Huffner, R. Niedermeier, Graph-modeled data clustering: Exact algorithms for clique generation, Theory of Computing Systems 38(4) (2005) 373 - 392.

[49] J. Gramm, R. Niedermeier, Minimum quartet inconsistency is fixed-parameter tractable, Lecture Notes in Computer Science 2089 (2001) (CPM 2001) 241 - 256.

[50] M. Grohe, Generalized model-checking problems for first-order logic, Lecture Notes in Computer Science 2010 (2001) (STACS 01) 12 - 26.

[51] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caliguri, C. Bloomfield, E. Lander, Molecular classication of cancer: class discovery and class prediction by gene expression monitoring, Science 286 (1999) 531 - 537.

[52] R. P. Gupta, Bounds on the chromatic and achromatic numbers of complementary graphs. in Recnet Progress in Combinatorics, 3rd Waterloo Conference on Combinatorics, Waterloo (ed. W.T. Tutte), Academic Press, New York, 229 - 235, 1969.

[53] J. Guo, A more effective linear kernelization for cluster edting, Theoretical Computer Science 410(8-10) (2009) 718 - 726.

[54] J. Guo, C. Komusiewicz, R. Niedermeier, J. Uhlmann, A more relaxed model for graph-based data clustering: s-plex editing, Lecture Notes in Computer Science 5564 (2009) (AAIM 2009) 226 - 239.

[55] G. Gan, C. Ma, J. Wu, Data Clustering: Theory, Algorithms, and Applications, ASA-SIAM Series on Statistics and Applied probability, SIAM, Philadelphia, ASA, Alexandria, VA, 2007.

[56] V. Guruswami, List Decoding of Error-Correcting Codes (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition), Lecture Notes in Computer Science 3282, Springer, New York, 2004.

[57] M. Habib, C. Paul, A survey on algorithmic aspects of modular decomposition, CoRR abs/0912.1457: (2009).

[58] W. Hsu, T. Ma, Substitution decomposition on chordal graphs and applications, Lecture Notes in Computer Science 557 (1991) (ISA 91) 52 - 60.

[59] F. Huffner, C. Komusiewicz, M. hannes, R. Niedermeier, Fixed-parameter algorithm for cluster vertex deletion, Lecture Notes in Computer Science 4957 (2008) (LATIN 2008) 711 - 722.

[60] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, R. E. Stearns, The complexity of planar counting problems, SIAM Journal on Computing 27(4) (1998) 1142 - 1167.

[61] A. Jain, R. Dubes, Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs, NJ, 1988.

[62] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis, On generating all maximal independent Sets, Information Processing Letter 27(3) (1988) 119 - 123.

[63] S. Kratsch, Polynomial kernelizations for MIN F^+ Π_1 and MAX NP. Proceedings of 26th International Symposium on Theoretical Aspects of Computer Science (2009) 601 - 612.

[64] J. Köbler, U. Schöning, J. Torán, The Graph Isomorphism Problem: Its Structural Complexity, Birkhäuser, Boston, 1993.

[65] S. Kapoor, H. Ramesh, Algorithms for enumerating all spanning trees of undirected and weighted graphs, SIAM Journal on Computing 24(2) (1995) 247 - 265.

[66] G. Kortsarz, J. Radhakrishnan, S. Sivasubramanian, Complete partitions of graphs, 16th Annual ACM-SIAM Symposium on Discrete Algorithms (2005) 860 - 869.

[67] 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, Proceedings of the National Academy of Sciences of the United States of America 100 (2003) 11394 - 11399.

[68] T. Kloks, Treewidth, Computations and Approximations, Lecture Notes in Computer Science 842, Springer-Verlag, New York, 1994.

[69] I. Koutis, A faster parameterized algorithm for set packing, Information Processing Letters 94(1) (2005) 7 - 9.

[70] G. Lin, P. E. Kearney, T. Jiang, Phylogenetic k-root and steiner k-root, Lecture Notes in Computer Science 1969 (2000) (ISAAC 2000) 539 - 551.

[71] Z. Michalewicz, D. Fogel, How to Solve It: Modern Heuristics, Springer, New York, 2000.

[72] R. Motwani, and P. Raghavan, Randomized Algorithms, Cambridge University Press, New York, 1995.

[73] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, New Jersey, 1982.

[74] C. H. Papadimitriou, Computational Complexity, Addison Wesley, Reading, MA, 1994.

[75] P. A. Pevzner, S. H. Sze, Combinatorial approaches to finding subtle signals in DNA sequences, ISMB (2000) 269 - 278.

[76] S. S. Ravi, H. B. Hunt III, An application of the planar separator theorem to counting problems, Information Processing Letters 25(5) (1987) 317 - 322.

[77] E. Sampathkumar and V. Bhave, Partition graphs and coloring numbers of graphs, Discrete Mathematics 16 (1976) 57 - 60.

[78] M. Sipser, Introduction to the Theory of Computation, Course Technology, Boston, 2005.

[79] J. Scott, T. Ideker, R. M. Karp, R. Sharan, Effcient algorithms for detecting signaling pathways in protein interaction networks, Lecture Notes in Computer Science 3500 (2005) (RECOMB) 1 - 13.

[80] R. Shamir, R. Sharan, Algorithmic approaches to clustering gene expression data, pp. 269-299, in Current Topics in Computational Molecular Biology, eds. T. Jiang, Y. Xu, M. Zhang, MIT press, Cambridge, MA, 2002. Lecture Notes in Computer Science 2573 (2002) (WG 02) 379 - 390.

[81] R. Shamir, R. Sharan, D. Tsur, Cluster graph modification problems, Discrete Applied Mathematics 144 (2004) 173 - 182.

[82] T. Tao, C. Zhai, Best-k queries on database systems, Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management (2006) 790 - 791.

[83] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques, Lecture Notes in Computer Science 3106 (2004) (COCOON 2004) 161 - 170.

[84] L. G. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (1979) 189 - 201.

[85] V. V. Vazirani, Approximation Algorithms, Springer-Verlag, New York, 2001.

[86] V. Yegnanarayanan, On pseudocoloring of graphs, Utilitas Mathematica, 62 (2002) 199 - 216.

[87] V. Yegnanarayanan, The pseudoachromatic number of a graph, Southern Aian Bulletin of Mathematics 24 (2002) 129 - 136.

[88] A. Zuylen, D. Williamson, Deterministic algorithms for rank aggregation and other ranking and clustering problem, Workshop on Approximation and Online Algorithms (2007) 260 - 273.

VITA

Name: Jie Meng

Email address: jmeng@cse.tamu.edu

Address: Department of Computer Science
Texas A&M University
TAMU 3112
College Station, TX 77843-3112

Education: Ph.D. Computer Science, Texas A&M University, 2010
M.S. Computer Science and Engineering, Fudan University, 2004
B.S. Computer Science and Engineering, Fudan University, 2001