RANDOMIZATION TECHNIQUES IN FPT AND K-PATH PROBLEM

By

Fahad P

The Institute of Mathematical Sciences, Chennai.

A thesis submitted to the
Board of Studies in Theoretical Computer Sciences

In partial fulfillment of the requirements

For the Degree of

Master of Science

of

HOMI BHABHA NATIONAL INSTITUTE

April 2012

CERTIFICATE

Certified that the work contained in the thesis entitled Randomization techniques in FPT and k-path problem, by Fahad P, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Saket Saurabh
Theoretical Computer Science Group,
The Institute of Mathematical Sciences, Chennai

ACKNOWLEDGEMENTS

I would like to thank Professor Saket Saurabh for his immense support in all my works and whole hearted help and guidance throughout the completion of the thesis. I am really grateful to him for entrusting my capabilities.

I would like to thank Professor Venkatesh Raman for his valuable suggestions in the making of the thesis and also sharing his ideas about the topics presented in the thesis.

I am grateful to all my Professors V. Arvind, Kamal Lodaya, Meena Mahajan, R. Ramanujam, Vikram Sharma, and C. R.Subramaniam for giving me a chance to learn from them.

I would also like to thank my colleagues and friends for standing alongside me and helping me whenever I was in trouble.

I would like to thank my wife and my parents for their love.

Last, but not the least, I would like to thank God, the Almighty.

Abstract

In this thesis we study different randomized techniques used in parametrized complexity. These techniques include Color Coding, Divide and Color, Cut and Count and certain algebraic techniques. We illustrate these techniques by doing a case study on the problem of finding a path of length $k-1$ in an undirected graph.

Contents

List of Algorithms

1 Introduction

1.1 Introduction

Good algorithms have fast running time and always output correct answer. Unfortunately, many real world problems do not seem to have such good algorithms. So the algorithm designers have to compromise either on running time or on correctness or on both. Parameterized complexity is one such paradigm where we want the output to be correct but the running time is allowed to be exponential in a carefully chosen parameter. Parameterized complexity allows us to solve the problem more efficiently than brute force: here the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a parameterization of a problem is assigning an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm that solves the problem in time $f(k)|I|^{O(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending on the parameter k only.

In classical complexity the main two resources considered are time and space used for computation. However in the last two decades "randomness" as a resource has been studied in algorithms and complexity. In randomized algorithms we randomly sample an object (like coin toss) and use this information along with the input to compute the output. Here we are interested in randomized algorithms, which run in FPT time, but may give wrong answer with a small constant probability.

In this thesis we study different randomized techniques used in parameterized

complexity. The techniques are

- *Color Coding*
- *Divide and Color*
- *Cut and Count*
- *Koutis Williams approach*
- *Narrow sieve*

We illustrate these techniques by doing a case study on k-PATH, the problem of finding a path of length $k-1$ in an undirected graph, formally defined as follows

k-PATH	
Input:	An undirected graph $G = (V, E)$ and a positive integer k
Parameter:	k
Question:	Does there exist a path of length $k-1$ in G

Color Coding is used to solve the problem of detecting a k-sized subgraph of constant treewidth in an input graph. The idea of this technique is to randomly color the entire graph with a set of colors with the number of colors chosen in a way that if the smaller graph does exist in this graph as a subgraph, with high probability it will be colored in a way that we can find it efficiently. *Divide and Color* is a combination of Divide and Conquer paradigm and Color Coding. *Cut and Count* technique is introduced to solve connectivity type graph problems. The idea of this technique is to reduce the original problem to the task of counting possibly disconnected "cut solutions" modulo 2 by making sure that number of disconnected cut solutions is always even and number of connected solution is odd with good probability in single exponential time on graphs of bounded treewidth. In *Koutis Williams approach* and *Narrow Sieve*, original problem is reduced to algebraic problems. In *Koutis Williams approach* k-PATH is reduced to the problem of detecting an odd multilinear term in a multivariate polynomial and in *Narrow Sieve* k-PATH is reduced to polynomial identity testing.

Organization of the Thesis. In the rest of the chapter we present basic definitions and notations which we will follow in this thesis. In Chapter 2 we

construct pseudo random objects like family of perfect hash functions and universal sets which we use to derandomize algorithms. In chapter 3, we explain the *Color Coding* technique, solve k-PATH and derandomize the algorithm developed using family of perfect hash functions. In chapter 4, we describe *Divide and Color* technique, develop a randomized algorithm for k-PATH and derandomize it using universal sets. In chapter 5, we describe the general framework of *Cut and Count* technique and solve k-PATH using this technique. In chapter 6, we explain two algebraic techniques to solve k-PATH. The techniques are *Koutis Williams approach* and *Narrow Sieve*.

1.2 Preliminaries and notation

1.2.1 Basic Notations

We assume that the reader is familiar with basic notions like sets, functions, polynomials, matrices, vectors etc.

We use $\mathbb{Z}_n$ and $[n]$ to denote the set $\{0, 1, ..., n-1\}$ and Z_p^* to denote the set $\{1, 2, ..., p-1\}$. The set of k sized subsets of $[n]$ can be denoted by $\binom{[n]}{k}$ and by 2^U we mean the set of subsets of U. If M is a square matrix, then $trace(M)$ is the sum of main diagonal elements of M and M^T represents the transpose of M. By $\vec{0}$ we mean zero vector of any dimension where the notion of dimension will be clear from the context. If $w : U \to \{1, 2, ..., N\}$, we shorthand $w(S) = \sum_{e \in S} w(e)$ for $S \subseteq U$. If f is a function from set X to set Y and g is function from set Y to set Z, then the composite function denoted by $g \circ f$, from X to Z is defined as $(g \circ f)(x) = g(f(x))$. If s is a function from X to Y, then we use $s[x_1 \to y_1]$ to denote a function which agrees on s for all values of X except that it maps x_1 to y_1. If P is a logical proposition, then we use $[P]$ to denote the value (0 or 1) of the proposition P.

Growth of Functions

We employ mainly the big-Oh (O) notation (see [6]) and the big-Oh-star (O^*) notation introduced in [21]. Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ be two functions from Natural numbers to Natural numbers. We say that $f(n) = O(g(n))$ if there

exist constants c, and n_0 such that for all $n \geq n_0$, $f(n) \leq c.g(n)$. The notation O^* is essentially the big-Oh notation which hides polynomial factors and hence is used only for exponential time algorithms. We use $O^*(f(n))$ to denote $O(f(n).n^c)$ where c is some constant. In this thesis, we will use the O^* notation to hide factors polynomial in input size in order to focus on the function of the parameter. Hence, for us, $O^*(f(k))$ denotes $O(f(k).n^c)$ where k is the value of some parameter, n is the size of the input instance, and c is some constant.

Graphs

An undirected graph G is a pair $= (V, E)$ where V and E are unordered sets. The elements of V are called vertices of G. E consists of unordered pairs of vertices and elements of E are called edges of G. An edge between vertices u and v is represented as (u, v). Note that $(u, v) = (v, u)$ in an undirected graph. A vertex u and a vertex v are said to be adjacent if E contains the pair (u, v). The edge (u, v) is said to be incident on the vertices u and v, while u and v are called the endpoints of the edge (u, v). Degree of a vertex v in a graph G, denoted by $degree_G(v)$ is the number of edges incident on it. An undirected graph G is called a simple undirected graph if there is no edge in E of the form (v, v) where v is a vertex of G. In this thesis, the graphs we consider are all simple undirected graphs. Sometimes the set of vertices and the set of edges of a graph G are denoted by $V(G)$ and $E(G)$ respectively. The set of endpoints of edges in $X \subseteq E$ are denoted by $V(X)$. If Y is a subset of V in a graph $G = (V, E)$, then the subgraph induced on Y can be denoted by $G[Y]$. If A is a subset of E in a graph $G = (V, E)$, then the subgraph with vertex set $V(A)$ and edge set A can be denoted by $G[A]$. A walk in the graph G is a sequence $W = v_1, ..., v_t$ of vertices such that $(v_i, v_{i+1}) \in E$ for every $1 \leq i \leq t - 1$ and it is called a walk from v_1 to v_t in G. The length of this walk is $t - 1$. A walk in which any vertex occurs at most once is called a path. By $u \xrightarrow{k} v$ we mean there exists a path of length $k - 1$ from u to v. A walk where the first vertex is same as the last vertex and all the other vertices are distinct is called a cycle. The walks $v_i, v_{i+1}, ..., v_j$, $1 \leq i \leq j \leq t$ are called subwalks of the walk W. Number of connected components in a graph G is represented by $cc(G)$

Strings

Let A be a set whose elements can be viewed as the symbols of an alphabet. A string of length l over A is a sequence $S = s_1 s_2 ... s_l$ with $s_i \in A$ for each $i = 1, 2, ..., l$. We say that s_i is the symbol at position i of the string. The reverse of a string $S = s_1 s_2 ... s_l$ is $\overleftarrow{S} = s_l s_{l-1} ... s_1$. The concatenation of two strings $S = s_1 s_2 ... s_l$ and $T = t_1 t_2 ... t_k$ is $ST = s_1 s_2 ... s_l t_1 t_2 ... t_k$. A palindrome is a string that is identical to its reverse. For $A_1, A_2, ..., A_l \subseteq A$, we say that a string $s_1 s_2 ... s_l$ is an $A_1 A_2 ... A_l$-string if $s_i \in A_i$ holds for every $i = 1, 2, ..., l$. The set of strings of length n over an alphabet A can be represented as A^n.

1.2.2 Inclusion Exclusion Principle

Let U be a finite set (we call U as universe) and $A_1, A_2, ..., A_n \subseteq U$. The number of elements in U which is not in any A_i is given by,

$$\left|\bar{A}_1 \cap \bar{A}_2 \cap ... \cap \bar{A}_n\right| = |U| + \sum_{k=1}^{n} (-1)^k \sum_{i_1 < i_2 < ... < i_k} |A_{i_1} \cap A_{i_2} \cap ... \cap A_{i_k}|$$

1.2.3 Probability

We assume that the reader is familiar with basics of probability theory. In this section we list some definitions and rules in probability theory.

Definition 1 (Conditional probability)**.** *The conditional probability of an event $\mathcal{E}_1$ given another event $\mathcal{E}_2$ is denoted by $\Pr[\mathcal{E}_1 \mid \mathcal{E}_2]$ and is given by*

$$\frac{\Pr[\mathcal{E}_1 \ \cap \ \mathcal{E}_2]}{\Pr[\mathcal{E}_2]}$$

Proposition 2 (Union bound)**.** *In probability theory, union bound says that for any finite number of events, the probability that at least one of the events happen is not greater than the sum of the probabilities of the individual events. That is, for a set of events $\{\mathcal{E}_1, \mathcal{E}_2 ..., \mathcal{E}_k\}$*

$$\Pr[\mathcal{E}_1 \cup \mathcal{E}_2 \cup ... \cup \mathcal{E}_k] \leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + ... + \Pr[\mathcal{E}_k]$$

Definition 3 (k-wise independent)**.** *A random variable $X = X_1X_2...X_n$ chosen uniformly at random from a sample space $H_{n,k,b} \subseteq [b]^n$ is said to be k-wise independent if for any k positions $i_1 < i_2 < ... < i_k$ and any string $\alpha \in [b]^k$, we have*

$$\Pr[X_{i_1}X_{i_2}...X_{i_k} = \alpha] = \frac{1}{b^k}$$

Proposition 4 ([2])**.** *There exists a k-wise independent probability space $H_{n,k,b}$ of size $O(n^k)$ and it can be constructed efficiently in time linear in the output size.*

1.2.4 Treewidth

Definition 5 (Tree Decomposition, [16])**.** *A tree decomposition of a (undirected or directed) graph $G = (V, E)$ is a tree $\mathbb{T}$ in which each vertex $x \in \mathbb{T}$ has an assigned set of vertices $B_x \subseteq V$ (called a bag) such that $\bigcup_{x \in T} B_x = V$ with the following properties:*

- *for any $(u, v) \in E$, there exists an $x \in \mathbb{T}$ such that $u, v \in B_x$.*
- *if $v \in B_x$ and $v \in B_y$, then $v \in B_z$ for all z on the path from x to y in $\mathbb{T}$.*

The *treewidth* $tw(\mathbb{T})$ of a tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one, and the treewidth of a graph G is the minimum treewidth over all possible tree decompositions of G.

Dynamic programming algorithms on tree decompositions are often presented on nice tree decompositions which were introduced by Kloks [10]. We refer to the tree decomposition definition given by Kloks as *standard nice tree decomposition.*

Definition 6 (Standard Nice Tree Decomposition, [10])**.** *Standard nice tree decomposition is a tree decomposition where:*

- *every bag has at most two children*
- *if a bag x has two children l, r, then $B_x = B_l = B_r$*
- *if a bag x has one child y, then either $|B_x| = |B_y| + 1$ and $B_y \subseteq B_x$ or $|B_x| + 1 = |B_y|$ and $B_x \subseteq B_y$*

Definition 7 (Nice Tree Decomposition, [7])**.** *A nice tree decomposition is a tree decomposition with one special bag z called the root with $B_z = \emptyset$ and in which each bag is one of the following types:*

- ***Leaf bag:*** *a leaf x of $\mathbb{T}$ with $B_x = \emptyset$.*
- ***Introduce vertex bag:*** *an internal vertex x of $\mathbb{T}$ with one child vertex y for which $B_x = B_y \cup \{v\}$ for some $v \notin B_y$. This bag is said to introduce v.*
- ***Introduce edge bag:*** *an internal vertex x of $\mathbb{T}$ labeled with an edge $(u, v) \in E$ with one child bag y for which $u, v \in B_x = B_y$. This bag is said to introduce (u, v).*
- ***Forget bag:*** *an internal vertex x of $\mathbb{T}$ with one child bag y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$. This bag is said to forget v.*
- ***Join bag:*** *an internal vertex x with two child vertices l and r with $B_x = B_r = B_l$*

We additionally require that every edge in E is introduced exactly once.

Given a tree decomposition, a *standard nice tree decomposition* of equal width can be found in polynomial time [10] and in polynomial time, it can easily be modified to meet extra requirements of *nice tree decomposition*, as follows: add a series of forget bags to the old root, and add a series of introduce vertex bags below old leaf bags that are nonempty; Finally, for every edge $(u, v) \in E$ add an introduce edge bag above the first bag with respect to the in-order traversal of $\mathbb{T}$ that contains u and v. By fixing the root of $\mathbb{T}$, we associate with each bag x in a tree decomposition $\mathbb{T}$ a vertex set $V_x \subseteq V$ where a vertex v belongs to V_x if and only if there is a bag y which is a descendant of x in $\mathbb{T}$ with $v \in B_y$ (recall that x is its own descendant). We also associate with each bag x of $\mathbb{T}$ a subgraph of G as follows:

$$G_x = (V_x, E_x = \{e \in E | e \text{ is introduced in a descendant of } x\})$$

1.2.5 Algebraic structures

Definition 8 (Group)**.** *A group is a set G, together with a binary operation $\cdot$ (also called group operation) that combines any two elements a and b to form another element, denoted $a \cdot b$ or ab. To qualify as a group, the set and operation, $(G, \cdot)$, must satisfy four requirements known as the group axioms:*

- ***Closure:*** *For all $a, b \in G$, the result of the operation, $a \cdot b$, is also in G.*
- ***Associativity:*** *For all $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
- ***Identity element:*** *There exists an element e in G, called the identity element, such that for every element $a \in G$, $e \cdot a = a \cdot e = a$. The identity element of a group G is often written as 1 or 1_G.*
- ***Inverse element:*** *For each $a \in G$, there exists an element $b \in G$ such that $a \cdot b = b \cdot a = 1_G$.*

If the group operation of a group G is commutative (i.e, $a \cdot b = b \cdot a$ for all $a, b \in G$), then we say G is an abelian group.

In this thesis we are interested in the following groups

1. $\mathbb{Z}_2$: Set $\{0, 1\}$ with binary operation addition modulo 2.
2. $\mathbb{Z}$: Set $\{0, 1, 2, ...\}$ with normal addition operation.
3. $\mathbb{Z}_2^k$: Set of $\{0, 1\}$ vectors of length k with the group operation being entry-wise addition modulo 2.

Definition 9 (Ring)**.** *A ring is a set R equipped with two binary operations $+ : R \times R \to R$ and $\cdot : R \times R \to R$, called addition and multiplication. To qualify as a ring, the set and two operations, $(R, +, \cdot)$, must satisfy the following requirements.*

- *$(R, +)$ is required to be an abelian group.*
- ***Closure under multiplication:*** *$\forall a, b \in R$, $a \cdot b \in R$*
- ***Associativity of multiplication:*** *$\forall a, b, c \in R, (a \cdot b) \cdot c = a \cdot (b \cdot c)$*
- ***Existence of multiplicative identity:*** *There exists an element $1 \in R$, such that for all elements $a \in R$, $1 \cdot a = a \cdot 1 = a$*
- ***The distributive laws:***
  1. *$\forall a, b, c \in R$, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.*
  2. *$\forall a, b, c \in R$, $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$.*

Rings that also satisfy commutativity for multiplication are called commutative rings.

Definition 10 (Field)**.** *A field is a set $\mathbb{F}$ with two binary operations – addition, denoted by $+$ and multiplication, denoted by $\cdot$ satisfying following properties*

1. *$(\mathbb{F}, +)$ is an abelian group.*
2. *$(\mathbb{F} \setminus \{\mathbf{0}\}, \cdot)$ is an abelian group where $\mathbf{0}$ is the identity element of the group $(\mathbb{F}, +)$ which is called additive identity or zero element of the field.*
3. *Multiplication is distributive over addition, i.e,*
   $\forall\ a, b, c \in \mathbb{F},\ a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

In this thesis we are interested in the following fields.

1. $\mathbb{Z}$: Set $\{0, 1, 2, ...\}$ with operations addition and multiplication.
2. $\mathbb{Z}_p$: Set $\{0, 1, ..., p-1\}$ with operations addition modulo p and multiplication modulo p where p is a prime.
3. $\mathbb{F}_{2^n}$: Elements of this field are univariate polynomials of degree less than n over $\mathbb{Z}_2$ and the operations are addition modulo R and multiplication modulo R where R is an irreducible polynomial of degree n over $\mathbb{Z}_2$.

Definition 11. *The characteristic of a field $\mathbb{F}$ is defined to be the smallest number of times one must use the field's multiplicative identity element in a sum to get the additive identity element; the field is said to have characteristic zero if this repeated sum never reaches the additive identity.*

Definition 12 (Polynomial ring)**.** *The set of all polynomials with coefficients in the field K forms a commutative ring denoted $K[X]$ and is called the ring of polynomials over K where X is a set of variables. The two ring operations of $K[X]$ are polynomial addition and polynomial multiplication.*

Definition 13 (Group algebra)**.** *The group algebra $K[G]$ where K is a field and G is a group, is the set of mappings $f : G \to K$ of finite support (i.e, finitely many elements from G are mapped to non-zero elements in K). Each element of*

$K[G]$, $f : G \to K$ can be denoted as linear combinations of elements of G, with coefficients in K:

$$\sum_{g \in G} f(g)g$$

The following operations are defined in group algebra

(i) The addition operator of $K[G]$ is defined by

$$\sum_{g \in G} f_1(g)g + \sum_{g \in G} f_2(g)g = \sum_{g \in G} \left(f_1(g) + f_2(g)\right) g$$

(ii) Multiplication by a scalar $\alpha \in K$ is defined by

$$\alpha \left(\sum_{g \in G} f(g)g \right) = \sum_{g \in G} (\alpha f(g))g$$

(iii) The multiplication operator of $K[G]$ is defined by

$$\left(\sum_{g \in G} f_1(g)g \right) \left(\sum_{g \in G} f_2(g)g \right) = \sum_{g_1, g_2 \in G} \Big(f_1(g_1) f_2(g_2) \Big)(g_1 g_2)$$

We are interested in group algebras $\mathbb{Z}[\mathbb{Z}_2^k]$ and $\mathbb{Z}_2[\mathbb{Z}_2^k]$. In case of $\mathbb{Z}_2[\mathbb{Z}_2^k]$ an element can also be seen as a subset of $\mathbb{Z}_2^k$

Representation of $\mathbb{Z}_2^k$ and $\mathbb{Z}[\mathbb{Z}_2^k]$

Invertible matrices of dimension d with integer entries form a group $M^{d \times d}$ with matrix multiplication and an algebra $M^{d \times d}$ with matrix addition, multiplication by a scalar, and matrix multiplication. A representation of a group G is a group homomorphism $\rho : G \to M^{d \times d}$, i.e, ρ should hold following properties:

- $\rho(g_1 g_2) = \rho(g_1)\rho(g_2)$
- $\rho(1_G) = I$, where I is identity matrix

It is well known ([19]) that there is a one-to-one homomorphism $\rho : \mathbb{Z}_2^k \to M^{2^k \times 2^k}$. For $\mathbb{Z}_2$, the map $\rho : \mathbb{Z}_2 \to M^{2 \times 2}$ is defined by the representations of the $\mathbb{Z}_2$ elements:

$$\rho(0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \rho(1) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The map ρ for $\mathbb{Z}_2^k$ can be inductively defined as

$$\rho(v_1 v_2 ... v_k) = \begin{cases} \begin{pmatrix} X & 0 \\ 0 & X \end{pmatrix} & \text{if} \quad v_k = 0 \\ \\ \begin{pmatrix} 0 & X \\ X & 0 \end{pmatrix} & \text{if} \quad v_k = 1 \end{cases}$$

$$\text{where } X = \rho(v_1 v_2 ... v_{k-1})$$

The map ρ can be extended to a one-to-one map of $\mathbb{Z}[\mathbb{Z}_2^k]$ to $M^{2^k \times 2^k}$, as follows:

$$\rho(\sum_{v \in \mathbb{Z}_2^k} a_v v) = \sum_{v \in \mathbb{Z}_2^k} a_v \rho(v)$$

It can be verified that if w_1, w_2 are elements of $\mathbb{Z}[\mathbb{Z}_2^k]$ and $\alpha \in \mathbb{Z}$, we have $\rho(w_1 + w_2) = \rho(w_1) + \rho(w_2)$, $\rho(w_1 w_2) = \rho(w_1)\rho(w_2)$ and $\rho(\alpha w_1) = \alpha \rho(w_1)$. In fact, the map ρ defines an isomorphic matrix algebra which we will denote by $\rho(\mathbb{Z}[\mathbb{Z}_2^k])$.

The matrices in the set $\rho(\mathbb{Z}_2^k)$ of matrix representations of the $\mathbb{Z}_2^k$ elements, are simultaneously diagonalizable, i.e. there is a matrix U such that for all $v \in \mathbb{Z}_2^k$, we have $\rho(v) = U^{-1}\Lambda_v U$, where Λ_v are the eigenvalues of $\rho(v)$, also known as the characters of v. If $b(i)$ is the vector containing the k-bit binary form of i, the i^{th} eigenvalue of $\rho(v)$ is given by $(-1)^{v^T b(i-1)}$ [19].

1.3 Parameterized Complexity

Definition 14 ([15]). *A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the parameter of the problem.*

Definition 15 ([15]). *A parameterized problem L is said to be Fixed Parameter Tractable (FPT) if it can be determined in time $f(k).n^{O(1)}$ whether or not $(x, k) \in L$, where f is a computable function depending only on k and $n = |(x, k)|$. The complexity class containing all fixed parameter tractable problems is called FPT.*

2

Pseudo Random Objects

In this chapter we discuss about the (n, k, l)-*family of perfect hash functions* and (n, k)-*universal sets* which are family of functions satisfying some properties. Here we are interested in constructing an (n, k, k)-*family of perfect hash functions* and (n, k)-*universal sets* of size as small as possible, because we want to use these objects to derandomize algorithms whose running time will depend on the size of these objects. FKS Hashing can be used to create (n, k, k)-*family of perfect hash functions* of size $2^{O(k)} \log^2 n$ [9, 17]. Since in parameterized complexity we are concerned about the the base of the exponential function, we describe another construction of (n, k, k)-*family of perfect hash functions* of size $e^k k^{O(\log k)} \log^2 n$ via *splitters* [14]. In section 2.1, we define these objects and splitters. Since construction of (n, k, k)-*family of perfect hash functions* using splitters uses the (n, k, k^2)-*family of perfect hash functions* constructed using FKS Hashing, we describe this construction in section 2.2. In section 2.3, we describe the construction of (n, k, k)-*family of perfect hash functions* and (n, k)-*universal sets* using *splitters*.

2.1 Definitions

Definition 16. *An* (n, k, l)-splitter H *is a family of functions from* $[n]$ *to* $[l]$ *such that for all* $S \in \binom{[n]}{k}$*, there is an* $h \in H$ *that splits* S *perfectly, i.e., into equal sized parts* $h^{-1}(j) \cap S$*,* $j = 1, 2, ..., l$ *(or as equal as possible, if* l *does not divide* k*).*

Definition 17. *Let* H *be a family of functions from* $[n]$ *to* $[l]$*.* H *is an* (n, k, l)-*family of perfect hash functions if for all* $S \in \binom{[n]}{k}$*, there is an* $h \in H$ *which is*

one-to-one on S. Notice that an (n, k, l)- family of perfect hash functions is a (n, k, l)-splitter, where $l \geq k$

Definition 18. *Set of vectors $T \subseteq \{0,1\}^n$ is a (n, k)-universal set, if for any index set $S \subseteq [n]$ with $|S| = k$, the projection of T on S contains all possible 2^k configurations.*

2.2 FKS Hashing

In this section we discuss about an explicit construction of (n, k, k^2)-*family of perfect hash functions* developed in [9]. Consider the case where we want to construct family of hash functions H from $[n]$ to $[l]$ where $l \geq k$ such that for any subset $S \in \binom{[n]}{k}$, there exists $h \in H$ such that h is one-to-one on S.

Lemma 19 ([9])**.** *Fix a prime number p, such that $n < p < 2n$. Let $S \in \binom{[n]}{k}$ and $a \in \mathbb{Z}_p^*$ and $l \geq k$. Let $B(l, S, a, j) = |\{x | x \in S \text{ and } (ax \bmod p) \bmod l = j\}|$ for all $0 \leq j \leq l-1$. In other words, $B(l, S, a, j)$ is the number of times the value j is attained by the function $x \to (ax \bmod p) \bmod l$ when x is restricted to S. Then there exists $a \in \mathbb{Z}_p^*$ such that*

$$\sum_{j=0}^{l-1} \binom{B(l, S, a, j)}{2} < \frac{k^2}{l} \tag{2.2.1}$$

Proof. We show that

$$\sum_{a=1}^{p-1} \sum_{j=0}^{l-1} \binom{B(l, S, a, j)}{2} < \frac{(p-1)k^2}{l} \tag{2.2.2}$$

from which the lemma follows immediately. The sum in (2.2.2) is the number of pairs $(a, \{x, y\})$, with $x, y \in S$, $x \neq y$, $1 \leq a < p$ such that

$$(ax \bmod p) \bmod l = (ay \bmod p) \bmod l.$$

The contribution of $\{x, y\}$, $x \neq y$ to this quantity is at most the number of a such that

$$a(x-y) \bmod p \in \{l, 2l, ..., p-l, p-2l, ...\} \tag{2.2.3}$$

Since p is prime and $\mathbb{Z}_p^*$ is a field, there is a unique solution for a satisfying the equation $a(x-y) \bmod p = c$ for any c. This implies that the number of a satisfying (2.2.3) is at most $\frac{2(p-1)}{l}$. Since the number of choices of $\{x, y\}$, with $x, y \in S$ and $x \neq y$ is $\binom{k}{2}$, we obtain

$$\sum_{a=1}^{p-1} \sum_{j=0}^{l-1} \binom{B(l, S, a, j)}{2} \leq \binom{k}{2} \frac{2(p-1)}{l} < \frac{(p-1)k^2}{l}$$

□

Hence we get the following corollary

Corollary 20 ([9]). *For all $S \in \binom{[n]}{k}$, there exist $a \in \mathbb{Z}_p^*$ such that the mapping $x \to (ax \bmod p) \bmod k^2$ is one-to-one when restricted to S. In other words there exists an (n, k, k^2)-family of perfect hash functions of size $O(n)$*

The following lemma can be proved using lemma 19

Lemma 21 ([9]). *For all $S \in \binom{[n]}{k}$, there exists $a \in \mathbb{Z}_p^*$ such that $\sum_{j=0}^{k-1} B(k, S, a, j)^2 < 3k$*

Proof. By Lemma 19 we know that for all $S \in \binom{[n]}{k}$ there exist $a \in \mathbb{Z}_p^*$ such that

$$\begin{aligned}
\sum_{j=0}^{k-1} \binom{B(k, S, a, j)}{2} &< k \\
\frac{1}{2} \sum_{j=0}^{k-1} B(k, S, a, j)^2 - B(k, S, a, j) &< k \\
\sum_{j=0}^{k-1} B(k, S, a, j)^2 &< 2k + \sum_{j=0}^{k-1} B(k, S, a, j) \\
\sum_{j=0}^{k-1} B(k, S, a, j)^2 &< 3k \qquad \left(\because \sum_{j=0}^{k-1} B(k, S, a, j) = k \right)
\end{aligned}$$

□

Now using prime number theorem we will prove that we can construct (n, k, l)-*family of perfect hash functions* of size $O(k^2 \log n)$ in time $O(poly(k, \log n))$, where $l < k^2 \log n$.

Lemma 22 ([9]). *Let $S \in \binom{[n]}{k}$. Then there exists a prime $p < k^2 \log n$ such that the function $\lambda_p : x \to x \bmod p$ is one-to-one on S, i.e., for all $x, y \in S$ with $x \neq y$, $x \bmod p \neq y \bmod p$.*

Proof. Let $S = \{x_1, x_2, ..., x_k\}$ and $t = \prod_{i<j}(x_i - x_j)$. Then $\log |t| \leq \binom{k}{2} \log n$. By prime number theorem, $\log \left(\prod_{p<x,\ p\ prime} p\right) = x + o(x)$. Therefore $\log \left(\prod_{\substack{p<k^2 \log n \\ p\ prime}} p\right) \geq k^2 \log n$. So we conclude that there is a prime $p < k^2 \log n$ that cannot divide t and this is the prime we are looking for. □

Since primality testing can be done in polynomial time [1] and number of primes less than x is approximately equal to $\frac{x}{\log x}$, we get the following corollary

Corollary 23. *An (n, k, l)-family of perfect hash functions of size $O(\frac{k^2 \log n}{\log(k \log n)})$ can be constructed in time $O(poly(k, \log n))$, where $l < k^2 \log n$.*

Theorem 24. *An (n, k, k^2)-family of perfect hash functions of size $O(\frac{k^4 \log^2 n}{\log(k \log n)})$ can be constructed in time $O(n.poly(k, \log n))$.*

Proof. Let H_1 be $(n, k, k^2 \log n)$-*family of perfect hash functions* of size $O(\frac{k^2 \log n}{\log(k \log n)})$ constructed as mentioned in corollary 23. Let H_2 be $(k^2 \log n, k, k^2)$-*family of perfect hash functions* of size $O(k^2 \log n)$ constructed as mentioned in corollary 20. Now consider the family of functions $H = \{g \circ f | f \in H_1, g \in H_2\}$. we claim that H is (n, k, k^2)-family of perfect hash functions because for any $S \in \binom{[n]}{k}$ there exist a function that maps elements of S to distinct values in $[k^2 \log n]$ and there exist a function in H_2 that maps these distinct values to distinct values in $[k^2]$. It is easy to see that size of H is $O(\frac{k^4 \log^2 n}{\log(k \log n)})$ and it can be constructed in time $O(n.poly(k, \log n))$. □

2.3 Splitters

In this section we describe about the construction of (n, k, k)-*family of perfect hash functions* and (n, k)-*universal sets* using (n, k, l)-*splitters*. In all three combinatorial objects- (n, k, k)-*family of perfect hash functions*, (n, k) -*universal sets* and (n, k, l)-*splitters*, our objective is to find a set of vectors of length n over an alphabet of size b (in case of (n, k, k)-*family of perfect hash functions* $b = k$, in case of (n, k)-*universal sets* $b = 2$ and in case of (n, k, l)-*splitters* $b = l$) such that for any k

out of n indices, we will find some "nice" configurations. This generalized problem is called *k-restriction problem*. We will describe it formally in section 2.3.1 and explain how these problems will fall into *k-restriction problem*. In section 2.3.3 and section 2.3.4 we construct (n, k, k)-*family of perfect hash functions* and (n, k)-*universal sets* respectively using the solution of *k-restriction problem* developed in section 2.3.2 and using FKS hashing.

2.3.1 k-restriction problem

k-restriction problem is formally defined as follows

k-restriction problem

Input: Positive integers b, k, n and a list $\mathcal{C} = C_1, C_2, ..., C_m$ where $C_i \subseteq [b]^k$ and with the collection $\mathcal{C}$ being invariant under permutation of $[k]$

Output: Collection of vectors $\mathcal{V} \subseteq [b]^n$ such that $\forall S \subseteq [n]$ with $|S| = k$ and $\forall j : 1 \leq j \leq m$, $\exists v \in \mathcal{V}$ such that projection of v on S, $v(S) \in C_j$

An important parameter of *k-restriction problem* is $c = \min_{1 \leq j \leq m} |C_j|$. We call c/b^k the *density* of the problem. Now we explain how the combinatorial objects which we defined in section 2.1, fall into the category of *k-restriction problem*.

(i) (n, k, l)-**splitters.** To specify *splitters* as *k-restriction problem*, let $b = l$ and let $\mathcal{C}$ consist of one set C_1 containing all vectors from $[b]^k$ such that each value in $[b]$ appears exactly k/l times (if l does not divide k, then some values in $[b]$ appear $\lceil k/l \rceil$ times and some appear $\lfloor k/l \rfloor$ times)

(ii) (n, k, k)-**family of perfect hash functions.** In this case, $b = k$ and $\mathcal{C}$ consist of only one set C_1 containing all permutations of $[k]$

(iii) (n, k)-**universal sets.** In this case, $b = 2$ and $\mathcal{C}$ consists of $C_x = \{x\}$ for all $x \in \{0, 1\}^k$

2.3.2 Solving k-restriction problem

We introduced *k-restriction problem* to construct *family of perfect hash functions* and *universal sets* of size as small as possible. So first we find the number of vectors that suffices to become solution of *k-restriction problem* using probabilistic

argument. If a vector $v \in [b]^n$ is chosen uniformly at random, then for any $S \in \binom{[n]}{k}$ and C_j the probability that $v(S) \in C_j$ is $\frac{|C_j|}{b^k} \geq \frac{c}{b^k}$, where $c = min_{1 \leq i \leq m}|C_i|$. Therefore, if we choose t random vectors, $V_t = \{v_1, v_2, ..., v_t\}$, we get via union bound that

$$\begin{aligned} \Pr[V_t \text{ is not a solution}] &\leq \sum_{S \in \binom{[n]}{k}} \sum_{j=1}^{m} \Pr[\forall v : v \in V_t \text{ and } v(S) \notin C_j] \\ &\leq \binom{n}{k} \sum_{j=1}^{m} \left(1 - \frac{|C_i|}{b^k}\right)^t \\ &\leq \binom{n}{k} m \left(1 - \frac{c}{b^k}\right)^t \end{aligned} \tag{2.3.1}$$

Restricting equation (2.3.1) to be less than 1 implies that

$$t \geq \lceil \frac{k \ln n + \ln m}{\ln(b^k/(b^k - c))} \rceil \tag{2.3.2}$$

Thus for any given *k-restriction problem*, there exist a solution of at most this size. We will refer to (2.3.2) as the *union bound*. Let $H_{n,k,b}$ be a k-wise independent probability space with n random variables taking values in $[b]$, such as the one mentioned in Proposition 4. Note that the union bound (2.3.2) is applicable even when the vectors are not chosen uniformly at random from $[b]^n$, but chosen uniformly at random from a k-wise independent space $H_{n,k,b}$ because probability calculation only examines k sized sets of $[n]$.

Now we discuss about the construction of a solution of size equaling the union bound. This construction is computationally expensive, i.e, not in polynomial time or even in FPT time in parameter k. But we will use this construction for making a *family of perfect hash functions* and *universal sets* after reducing the size of the universe. Since we are discussing the general *k-restriction problem*, we assume that we have a *membership oracle*: a procedure that, given $v \in [b]^n$, $S \in \binom{[n]}{k}$ and $j \in [m]$, says whether or not $v(S) \in C_j$, within some time bound T. For the examples we are interested in, this oracle computation will be easy, usually taking just $O(k)$ time.

Theorem 25 ([14]). *For any k-restriction problem with $b \leq n$, there is a deterministic algorithm that outputs a collection obeying the k-restrictions, with the size*

of the collection equaling the union bound. The time taken to output the collection is

$$O\left(\frac{b^k}{c}.\binom{n}{k}.m.T.|H_{n,k,b}|\right) \tag{2.3.3}$$

where T is the time complexity of the membership oracle.

Proof. Consider a set-system in which the universe (ground set) is $H_{n,k,b}$. The sets are $T_{S,j}$, indexed by pairs (S, j) such that $S \in \binom{[n]}{k}$ and $1 \leq j \leq m$. $T_{S,j}$ consists of all $h \in H_{n,k,b}$ such that $h(S) \in C_j$. We do not explicitly list out the sets $T_{S,j}$: note that any given h can be tested for membership in $T_{S,j}$ in time T, using the given *membership oracle*. Any subset of $H_{n,k,b}$ that *hits* (intersects) all subsets $T_{S,j}$ is a good collection (i.e., is a collection satisfying the *k-restriction problem*). This is the well-known *hitting set* problem.

We can find such a collection by a greedy algorithm via a simple observation, which follows fairly easily by inspecting (2.3.1) and by using the fact that (2.3.1) holds even if we pick vectors at random from $H_{n,k,b}$; the observation is that there must be an $h \in H_{n,k,b}$ such that h hits at least fraction c/b^k of the sets $T_{S,j}$. The obvious idea then is to find such an h using the *membership oracle* and add it to our current(partial) hitting set, removing the sets hit by h from the set system, and repeating. Finding such an h takes time at most $O\left(\binom{n}{k}.m.T.|H_{n,k,b}|\right)$; also, the number of sets in our set-system is effectively "shrunk" to at most $m\binom{n}{k}(1 - c/b^k)$ after picking h. Therefore the results of a greedy algorithm will produce a solution of size $\lceil \frac{k \ln n + \ln m}{\ln(b^k/(b^k - c))} \rceil$, same as that of (2.3.2). So, the total time taken is at most

$$O\left(\binom{n}{k}.m.T.|H_{n,k,b}|\left(\sum_{i=0}^{\infty}(1 - c/b^k)^i\right)\right) = O\left(\frac{b^k}{c}.\binom{n}{k}.m.T.|H_{n,k,b}|\right) \tag{2.3.4}$$

□

For *family of perfect hash functions* and *universal sets*, we explicitly state the size and time complexity by substituting proper values (Note that $|H_{n,k,b}| \leq n^k$ as mentioned in Proposition 4) in theorem 25 and get theorem 26 as follows

Theorem 26 ([14])**.** *(i) An (n, k, k)-family of perfect hash functions of cardinality $O(e^k\sqrt{k}\log n)$ can be constructed deterministically in time $O(k^{k+1}\binom{n}{k}n^k/k!)$. (ii) An (n, k)-universal set of cardinality $O(k2^k \log n)$ can be constructed deterministically in time $O(\binom{n}{k}k2^{2k}n^k)$.*

2.3.3 (n, k, k)-family of perfect hash functions

First we give a brief overview of construction of (n, k, k)-*family of perfect hash functions*. Starting with the universe size n, we first reduce our problem to one with universe size k^2 by finding a poly-time computable family A of (n, k, k^2)-*family of perfect hash functions* (Theorem 24). A construction of (k^2, k, k)-*family of perfect hash functions* will then be pulled back to (n, k, k)- *family of perfect hash functions* at a $poly(k).\log^2 n$ cost in the size of the family. For that we will find (k^2, k, l)-*splitters* for $l = O(\log k)$. This guarantees us, for each $S \in \binom{[k^2]}{k}$, there exists a function which partitions S equally in l blocks. Then for each block we construct $(k^2, k/l, k/l)$-*family of perfect hash functions* by applying Theorem 26. We need splitters with universe size k^2.

Lemma 27. *For any $k \leq n$ and for all $l \leq n$, there is an explicit family $B(n, k, l)$ of (n, k, l)-splitters of size $\binom{n}{l-1}$*

Proof. For every choice of $1 \leq i_1 < i_2 < ... < i_{l-1} \leq n$, define a function $h : [n] \to [l]$ by $h(s) = j$ iff $i_{j-1} < s \leq i_j$, for all $s \in [n]$ (taking i_0=0 and i_l=n). □

Construction. Let $l = c \log k$ for some constant c (we will fix c later). Let $A = A(n, k, k^2)$, $B = B(k^2, k, l)$ and $C = C(k^2, k/l, k/l)$ be respective function families presented by Theorem 24, Lemma 27 and (i) of Theorem 26. Then our required *perfect hash function family* H can be defined as

$$H = \{(a, b, c_1, c_2, ...c_l) | a \in A, b \in B, \forall i \in [l] : c_i \in C\}$$

where each $(a, b, c_1, c_2, ..., c_l) \in H$ is defined by

$$(a, b, c_1, c_2, ..., c_l)(x) = c_{b(a(x))}\left(a(x)\right) + \frac{k}{l}\left(b\left(a(x)\right) - 1\right)$$

Correctness. It can be easily verified that each $h \in H$ maps $[n]$ to $[k]$. Let $S \in \binom{[n]}{k}$. We need to show that there exist a function in H which is *one-to-one* on S. By the property of A there exist a function $a \in A$ which is *one-to-one* on S. Let $S' = \{i | \exists j \in S : a(j) = i\}$. Since a is *one-to-one* on S, $|S'| = k$. By the property of B, there exist $b \in B$ such that b splits S' equally into l blocks. Let $S'_i = \{j | j \in S' \text{ and } b(j) = i\}$ for all $i \in [l]$. Now by the property of C, we have $c_i \in C$ for all i such that c_i is *one-to-one* on S'_i

Size and Time We know that $|A| = O\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right)$ and A can be constructed in time $O(n.poly(k, \log n))$. By Lemma 27, $|B| = \binom{k^2}{l-1} = k^{O(\log k)}$ and B can be constructed in time $k^{O(\log k)}$. By Theorem 26, $|C| = O(e^{k/l}\sqrt{k/l} \log k)$ and can be constructed in time $k^{O(k/l)}$, which is equal to 2^k for a suitable choice of c. Hence size of H is,

$$\begin{aligned} |H| &= |A|.|B|.|C|^l \\ &= O\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right).k^{O(\log k)}.(e^k k^{O(\log k)}) \\ &= e^k k^{O(\log k)} \log^2 n \end{aligned}$$

Theorem 28. *An (n, k, k)-family of perfect hash functions of size $e^k k^{O(\log k)} \log^2 n$ can be constructed in time linear in the output size.*

2.3.4 (n, k)-universal sets

The idea for (n, k)-*universal sets* is similar to that behind Theorem 28, with the only modification being that we now need the universal sets guaranteed by (ii) of Theorem 26. Thus we get

Theorem 29. *An (n, k)-universal sets of size $2^k k^{O(\log k)} \log^2 n$ can construed in time linear in the output size*

3

Color Coding

Color Coding is a randomized technique introduced by Alon et al. [3] to handle constant treewidth subgraph isomorphism problem (whether the given constant treewidth k-sized graph H exist in the given graph G as a subgraph) in $2^{O(k)}n^{O(1)}$ time. The idea behind this technique is to randomly color the vertices of graph G with k colors such that if the graph H does exists in the graph G as a subgraph, then with good probability the copy of H in G will become colorful, that is, each of the vertices in the copy of H gets distinct colors. If it happens that given a colored graph one can efficiently test whether there exists a colorful subgraph H in it, then we will get a randomized algorithm for the subgraph isomorphism problem. This randomized algorithm can be derandomized using family of perfect hash functions. In this chapter we study *Color Coding* technique by applying it to k-PATH

Let $G = (V, E)$ be a directed or undirected graph. Recall that k-PATH problem is to test whether a path of length $k-1$ exists in G. Choose a random coloring of the vertices of G with k colors. A path in G is said to be *colorful* if each vertex on it is colored with distinct colors. Each simple path of length $k-1$ becomes colorful in a random coloring with probability $k!/k^k > e^{-k}$ because number of ways we can color a fixed k-path using k colors is k^k and out of which exactly in $k!$ colorings the path becomes colorful. Next we present an algorithm to check whether there exists a colorful k-path in a colored graph.

Lemma 30. *Let $G = (V, E)$ be a directed or undirected graph and let $c : V \to [k]$ be a coloring of its vertices with k colors. There exists a deterministic algorithm that counts the number of colorful paths of length $k-1$ in G in $2^k n^{O(1)}$ worst-case time.*

Proof. We give an algorithm that exploits the inclusion exclusion principle. For any $S \subseteq [k]$, let W_S be the set of walks of length $k-1$ in the induced subgraph $G[V \setminus c^{-1}(S)]$ where $c^{-1}(S)$ is the set of vertices in G colored with S. Note that $W_\emptyset$ is the set of walks of length $k-1$ in G and for all $S \subseteq [k]$, $W_S \subseteq W_\emptyset$. Let $W_j = W_{\{j\}}$ for all $j \in [k]$. It is clear that $W_j \subseteq W_\emptyset$ for all $j \in [k]$ and $W_{i_1} \cap W_{i_2} \cap \ldots \cap W_{i_l} = W_{\{i_1,i_2,\ldots,i_l\}}$. It is easy to see that any colorful walk of length $k-1$ has to be a colorful k-path. Then by inclusion exclusion principle (section 1.2.2 of chapter 1), we get

$$\begin{aligned} \text{No. of colorful } k\text{-paths} &= |W_\emptyset| + \sum_{j=1}^{k} (-1)^j \sum_{i_1<i_2<\ldots<i_j} \left|W_{i_1} \cap W_{i_2} \cap \ldots \cap W_{i_j}\right| \\ &= \sum_{S \subseteq [k]} (-1)^{|S|} |W_S| \end{aligned}$$

Each $|W_S|$ can be computed easily. Let A_S be the adjacency matrix of $G[V \setminus c^{-1}(S)]$. Then sum of entries in the matrix A_S^{k-1} is equal to $|W_S|$ and it can be computed using $O(\log k)$ matrix multiplication. Hence the number of colorful k-paths in G can be computed in time $O(2^k V^w \log k)$, where $w < 2.376$ is the exponent of matrix multiplication. □

Theorem 31. *There is a randomized algorithm for* k-PATH *in a graph* G *in* $O((2e)^k V^w \log k)$ *time, with no errors for* NO *instances and errors with constant probability for* YES *instances.*

Proof. If we randomly color G with k colors, then a fixed k-path becomes colorful with probability $k!/k^k > e^{-k}$. We know that we can test whether a colorful k-path exists in a colored graph G in $O(2^k V^w \log k)$ time (Lemma 30). If a random coloring makes any of the k-path colorful, then algorithm explained in Lemma 30 outputs a non-zero value. Since the probability that a fixed k-path becomes colorful in a random coloring is at least e^{-k}, we can repeat this process e^k times and get a constant success probability. So if we repeat the process of randomly coloring G with k colors and counting colorful k-paths (as mentioned in Lemma 30) e^k times, then with probability $(1-e^{-k})^{e^k} \leq e^{-1}$ ($\because 1 - e^{-k} \leq e^{-e^{-k}}$) a fixed k-path will not be counted in all e^k execution of the algorithm mentioned in Lemma 30. Hence by this procedure a k-path, if one exists, will not be detected with probability at most e^{-1}. Running time of this procedure is $O((2e)^k V^w \log k)$. □

We can derandomize this algorithm using (n, k, k)-*family of perfect hash functions*. If we want to give every simple path of length $k-1$ in a graph $G = (V, E)$ a chance of being discovered, we need a list of k colorings of V such that for every subset $V' \subseteq V$ with $|V'| = k$ there exists a coloring in the list that gives each vertex in V' a distinct color. In other words we need an $(|V|, k, k)$-*family of perfect hash functions* that maps $\{1, 2, ..., |V|\}$ to $\{1, 2, ..., k\}$. Recall that there exist an (n, k, k)- *family of perfect has functions* of size $e^k k^{O(\log k)} \log^2 n$ and can be constructed in time linear in the output size (Theorem 28). We can construct an (n, k, k)-*family of hash functions* of size $2^{O(k)} n^{O(1)}$ using FKS hashing. But the constant in the exponential function $2^{O(k)}$ is very large and that makes the derandomization using FKS hashing practically infeasible. Hence we derandomize the above algorithm using (n, k, k)-*family of hash functions* constructed via spitters (refer Chapter 2) and we get the following theorem

Theorem 32. k-PATH *can be solved deterministically in* $O((2e)^k k^{O(\log k)} V^w)$ *time, where* $w < 2.376$ *is the exponent of matrix multiplication.*

4

Divide and Color

Divide and Color is a randomized technique introduced by Kneis et al. [11] and Chen et al. [5] independently, for solving hard graph problems. It is a combination of well known *divide and conquer* paradigm and *color coding*. The idea behind this technique is to randomly color all vertices (or edges) of a graph with black and white, and then solve the problem recursively on the two induced parts. In this chapter we demonstrate this technique by giving a randomized algorithm for k-PATH. Finally we obtain a deterministic algorithm by derandomization using (n, k)-*universal sets*.

4.1 Randomized Algorithm for k-PATH

The basic idea of *Divide and Color* technique is to use only two colors and solve reduced instance of the problem on each of the induced subgraphs recursively. These two solutions must be combined into the solution of original instance. Unlike the *Color Coding* technique, there is no need for solving the problem for the colored instance, which we addressed by principle of inclusion exclusion. This is because the recursive approach eventually reduces the problem to a simple instance. However we need to be careful when combining solutions of the reduced instances to get a solution on the original instance. Towards this we define the problem EXTENDED k-PATHS as follows

EXTENDED k-PATHS	
Input:	A graph $G = (V, E)$ and a positive integer k
Parameter:	k
Output:	The set $\{(u, v) \in V \times V \mid u \xrightarrow{k} v\}$

Consider the problem EXTENDED k-PATHS. If we randomly color the vertices of a given graph G with two colors (say 0 and 1), probability that for any fixed k-path P, the first half of the vertices in P gets color 0 and the second half of the vertices in P gets color 1 is $\frac{1}{2^k}$. So in an algorithm that randomly colors with two colors and recursively solves the two induced parts, the probability that a fixed k-path will be detected is less than or equal to 2^{-k}. To get an algorithm with constant success probability we need to repeat this procedure at least exponential in k many times. Since the procedure explained is recursive we can repeat the process in two ways. One way is to repeat the entire procedure exponentially many times and the other is to repeat each recursive call exponentially many times (exponential in parameter). Here we prefer the latter option because we can easily derandomize the latter option using universal sets.

Algorithm 1 RANDOMIZED PATHS(G, k)

Input: A graph $G = (V, E)$ and a positive integer k
Output: The set $\{(u, v) \in V \times V \mid u \xrightarrow{k} v\}$

1. If $k = 1$ then return $\{(v, v) \in V \times V \mid v \in V\}$
2. $R := \emptyset$
3. Repeat 3.2^k times
   (a) Randomly color vertices in V using colors 0 and 1 with uniform probability. Let $V' \subseteq V$ be set the vertices colored with 0
   (b) $R_1 :=$RANDOMIZED PATHS$(G[V'], \lceil k/2 \rceil)$
   (c) $R_2 :=$RANDOMIZED PATHS$(G[V \setminus V'], \lfloor k/2 \rfloor)$
   (d) For all $u, v, w, x \in V(G)$, if $(u, v) \in R_1 \ \wedge (v, w) \in E \wedge \ (w, x) \in R_2$, then add (u, x) to R
5. Return R

Theorem 33. *Algorithm 2 solves* Extended k-path *with error probability at most* $1/4$ *in time* $O^*(4^k)$.

Proof. It is easy to see that if Algorithm 2 returns a pair (u, v), then indeed $u \xrightarrow{k} v$. On the other hand, if G contains such a path, Algorithm 2 may not return the corresponding pair (u, v) with some probability p_k.

Assume there is a simple k-path P from u to v in G. The probability that the first $k/2$ nodes of P are colored 0 and the other $k/2$ nodes of P are colored 1 in a random coloring is 2^{-k}. In that case, randomized paths$(G[V'], k/2)$ and randomized paths$(G[V \setminus V'], k/2)$ do not contain pairs that allow the algorithm to insert (u, v) into R with probability at most $2p_{k/2}$. After 3.2^k iterations, the probability that $(u, v) \notin R$ is at most

$$\left(1 - 2^{-k} + 2^{-k}.2.p_{k/2}\right)^{3.2^k},$$

because with probability $1 - 2^{-k}$ the initial coloring is bad and with probability at most $2^{-k}.2.p_{k/2}$ the initial coloring is good, but recursive detection of the monochromatic subpaths fails. We show by induction that $p_k \leq \frac{1}{4}$ for every k: Obviously $p_1 = 0$, and for $k \geq 2$ we get the inequality

$$\begin{aligned} p_k &\leq \left(1 - 2^{-k} + 2^{-k}.2.\frac{1}{4}\right)^{3.2^k} && \text{(By induction hypothesis)} \\ &= \left(1 - 2^{-k}\frac{1}{2}\right)^{3.2^k} \\ &\leq e^{-3/2} && (\because 1 - x \leq e^{-x}) \\ &< \frac{1}{4} \end{aligned}$$

Let $T(k)$ denote the number of recursive calls issued by Algorithm 2. Then we get the recurrence

$$T(k) \leq 3.2^k \left(T(\lceil k/2 \rceil) + T(\lfloor k/2 \rfloor)\right) \leq 3.2^{k+1} T(\lceil k/2 \rceil)$$

Using the fact that $k + \lceil k/2 \rceil + \lceil k/2 \rceil/2 \rceil + \ldots + 1 \leq 2k + \log k$, we get $T(k) = O(4^k k^2 3^{\log k})$. All other operations performed during a call to the algorithm only take polynomial time. Thus Algorithm 2 finds a path of length k with probability

at least 3/4, if one exists, in time $O^*(4^k)$. □

4.2 Derandomization

In this section we discuss about derandomization of the above algorithm. Any simple k-path, $v_1, v_2, \ldots, v_k$ in a graph $G = (V, E)$ will be discovered by the above algorithm only if in the initial coloring $v_1, v_2, ..., v_{\lceil \frac{k}{2} \rceil}$ gets one color and $v_{\lfloor \frac{k}{2} \rfloor}, v_{\lfloor \frac{k}{2} \rfloor + 1}, ..., v_k$ gets another color and this property hold in the recursive steps as well. So if we have a list of vertex coloring with 2 colors such that for any k vertices the restriction of colorings on these vertices gives all possible colorings with 2 colors on these vertices, then for any k-path there exist one good coloring, i.e, first half of vertices will get one color and the second half of the vertices will get the other color. In other words we can use (n, k)-*universal sets* to derandomize above algorithm. Let $U_{n,k}$ be an (n, k)-*universal sets* of size $2^k k^{O(\log k)} \log^2 n$ as described in Theorem 29.

Algorithm 2 PATHS(G, k)

Input: A graph $G = (V, E)$ and a positive integer k
Output: The set $\{(u, v) \in V \times V \mid u \xrightarrow{k} v\}$

1. If $k = 1$ then return $\{(v, v) \in V \times V \mid v \in V\}$
2. $R := \emptyset$
3. For all $f \in U_{|V(G)|,k}$
   (a) Let $V' \subseteq V$ be set the vertices that are mapped to 0 in f
   (b) $R_1 :=$PATHS$(G[V'], \lceil k/2 \rceil)$
   (c) $R_2 :=$PATHS$(G[V \setminus V'], \lfloor k/2 \rfloor)$
   (d) For all $u, v, w, x \in V(G)$, if $(u, v) \in R_1 \ \wedge (v, w) \in E \wedge \ (w, x) \in R_2$, then add (u, x) to R
5. Return R

Theorem 34. *Algorithm 2 solves* EXTENDED k-PATH *deterministically in time* $4^k k^{O(\log^2 k)} n^{O(1)}$ *where* $n = |V(G)|$

Proof. It is easy to see that if Algorithm 2 returns a pair (u, v), then indeed $u \xrightarrow{k} v$. Let $v_1, v_2, ..., v_k$ be a path P in G. We can show via induction on k, that Algorithm 2 will output a set that contains (v_1, v_k). There exists $f \in U_{|V(G)|,k}$ such that f maps the first half of vertices in P to 0 and the second half of vertices in P to 1. Inductively PATHS$(G_1, \lceil k/2 \rceil)$ and PATHS$(G_2, \lfloor k/2 \rfloor)$ will return $(v_1, v_{\lceil k/2 \rceil})$ and $(v_{\lceil k/2 \rceil+1}, v_k)$ respectively. Hence Algorithm 2 will return an output that will contain (v_1, v_k).

Let $T(k)$ be the number of recursive calls of the Algorithm 2.

$$\begin{aligned} T(k) &\leq (2^k k^{c \log k} \log^2 n)\,(T(\lceil k/2 \rceil) + T(\lfloor k/2 \rfloor)) \qquad \text{where } c \text{ is a constant} \\ &\leq (2^{k+1} k^{c \log k} \log^2 n) T(\lceil k/2 \rceil) \end{aligned}$$

Using the fact that $k + \lceil k/2 \rceil + \lceil k/2 \rceil/2 \rceil + ... + 1 \leq 2k + \log k$, and $(\log n)^{\log k} \leq k^{\log k} n$ for $n \geq 6$, we get $T(k) = 4^k k^{O(\log^2 k)} n$. All other operations performed during a call to the algorithm only take polynomial time. Thus Algorithm 2 finds a path of length k, if one exists, in time $T(k) = 4^k k^{O(\log^2 k)} n^{O(1)}$. □

5

Cut and Count

Cut and Count is a randomized technique introduced by Cygan et al. [7] to deal with connectivity-type graph problems in bounded treewidth graphs. For most problems involving a global constraint like connectivity, this technique gives a randomized algorithm with runtime $c^{tw}|V(G)|^{O(1)}$ where tw is the treewidth of the given graph G and c is some constant, and one sided error with constant probability. In a decision problem our objective is to test whether the solution set is empty or not. The idea behind this technique is to relax the solution set (usually we relax the connectivity constraint) and construct a set $\mathcal{C}$ in such way that all the elements from the relaxed solution set which are not in the solution set will contribute even number of elements to $\mathcal{C}$ and elements from the solution set will contribute odd number of elements to $\mathcal{C}$ with good probability. Now if we have an efficient way to count $|\mathcal{C}|$ mod 2, we get a randomized algorithm for the problem. We relax the solution set and construct a set $\mathcal{C}$ in such a way that counting C is a local problem and can be done efficiently by standard dynamic programming. Randomization in this technique is used via *isolation lemma* which we define and prove in section 5.1. In section 5.2 we describe the general framework of *Cut and Count* technique and in section 5.3 we apply this technique to solve the problem of testing whether the given bounded treewidth graph has a k-cycle or not. In section 5.4 we describe how to solve k-PATH using depth first search(DFS) and solution of the problem of testing whether there exists k-cycle in a bounded treewidth graph.

5.1 Isolation Lemma

As mentioned in the initial paragraph we need a way to construct $\mathcal{C}$ such that elements in the solution set will contribute odd number of elements to $\mathcal{C}$. This can be achieved using isolation lemma. In a graph problem solution set is usually a set of vertices (or edges). According to isolation lemma if we assign positive integer weights to vertices (or edges) then with a good probability there exists a unique minimum weight element in the solution set. So with an additional weight constraint we can convert solution set to another set containing only one element. We formally define and prove isolation lemma in this section.

Definition 35 ([7]). *A function $w : U \to \mathbb{Z}$ isolates a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $w(S') = \min_{S \in \mathcal{F}} w(S)$ where $w(S) = \sum_{e \in S} w(e)$.*

Lemma 36 (Isolation Lemma,[13]). *Let $\mathcal{F} \subseteq 2^U$ be a set family over a universe U with $|\mathcal{F}| > 0$. Suppose each element u of the set U is independently assigned a weight $w(u)$ uniformly from $\{1, 2, ..., N\}$, Then*

$$\Pr[w \text{ isolates } \mathcal{F}] \geq 1 - \frac{|U|}{N}$$

Proof. For any element $x \in U$, define

$$\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} w(S) - \min_{S \in \mathcal{F}, x \in S} w(S \setminus \{x\})$$

Observe that $\alpha(x)$ depends only on the weights of elements other than x, and not on $w(x)$ itself. So whatever the value of $\alpha(x)$, as $w(x)$ is chosen uniformly from $\{1, 2, ..., N\}$, the probability that it is equal to $\alpha(x)$ is at most $\frac{1}{N}$. Thus the probability that $w(x) = \alpha(x)$ for some x is at most $\frac{|U|}{N}$. Now if there are two sets A and B in $\mathcal{F}$ with minimum weight, then, taking any x in $A \setminus B$, we have

$$\begin{aligned}
\alpha(x) &= \min_{S \in \mathcal{F}, x \notin S} w(S) - \min_{S \in \mathcal{F}, x \in S} w(S \setminus \{x\}) \\
&= w(B) - (w(A) - w(x)) \\
&= w(x)
\end{aligned}$$

and as we have seen, this event happens with probability at most $\frac{|U|}{N}$. □

5.2 General Framework

The *Cut and Count* technique applies to problems with certain connectivity requirements. Let $S \subseteq 2^U$ be a set of solutions; we aim to decide whether it is empty. Conceptually, *Cut and Count* can naturally be split in two parts:

- **The Cut part:** Relax the connectivity requirement by considering the set $R \supseteq S$ of possibly disconnected candidate solutions. Furthermore, consider the subset $\mathcal{C}$ of set of pairs (X, C) where $X \in R$ and C is a consistent cut (to be defined later) of X such that for any $X \in R \setminus S$ there are even number of consistent cuts and for any $X \in S$ there is only one consistent cut.
- **The Count part:** Compute $|\mathcal{C}|$ modulo 2 using a sub-procedure. Non-connected candidate solutions $X \in R \setminus S$ cancel since they are consistent with an even number of cuts. Connected candidates $X \in S$ remain.

Note that we need the number of solutions to be odd in order to make the counting part work. For this we use the Isolation Lemma (Lemma 36): We introduce uniformly and independently chosen weights $w(v)$ for every $v \in U$ and compute $|\mathcal{C}_W|$ modulo 2 for every W , where $\mathcal{C}_W = \{(X, C) \in \mathcal{C} | w(X) = W\}$. The general setup can thus be summarized as in Algorithm 3

Algorithm 3 CutandCount($U, \mathbb{T}$, CountC)

Input: Set U; tree decomposition $\mathbb{T}$; Procedure *CountC* accepting a $w : U \to \{1, 2, ..., N\}$, $W \in \mathbb{Z}$ and $\mathbb{T}$

1: **for all** $v \in U$ **do**
2: Choose $w(v) \in \{1, 2, ..., 2|U|\}$ uniformly at random
3: **end for**
4: **for all** $0 \leq W \leq 2|U|^2$ **do**
5: **if** CountC$(w, W, \mathbb{T}) = 1 \mod 2$ **then**
6: **return** *yes*
7: **end if**
8: **end for**
9: **return** *no*

The following theorem can be proved from Lemma 36 by setting $\mathcal{F} = S$ and $N = 2|U|$:

Theorem 37. *Let $S \subseteq 2^U$ and $\mathcal{C} \subseteq 2^U \times (2^V \times 2^V)$. Suppose that for every $W \in Z$*

(1) $|\{(X,C) \in \mathcal{C} | w(X) = W\}| = |\{X \in S | w(X) = W\}| \mod 2$

(2) $\mathit{CountC}(w, W, \mathbb{T}) = |\{(X,C) \in \mathcal{C} | w(X) = W\}| \mod 2$

Then Algorithm 3 returns no if S is empty and yes with probability at least $\frac{1}{2}$ otherwise.

Proof. Suppose S is empty. Then because of condition (1) of the theorem, $|\{(X,C) \in \mathcal{C} | w(X) = W\}| = 0 \mod 2$ for all $W \in Z$ and hence because of condition (2) of the theorem $\mathrm{CountC}(w, W, \mathbb{T}) = 0 \mod 2$ for all $W \in Z$. So Algorithm 3 will return *no*. Suppose S is not empty. Now apply Lemma 36 by setting $\mathcal{F} = S$ and $N = 2|U|$. There exist a unique minimum weight (say it is W^*) element $X \in S$ with probability at least $1/2$ and so $|\{X \in S | w(X) = W^*\}| = 1$ with probability $1/2$. Hence Algorithm 3 will return *yes* with probability at least $1/2$ because of the conditions (1) and (2) of the theorem. □

When applying the technique, both the Cut and the Count part are non-trivial: In the Cut part one has to find the proper relaxation of the solution set, and in the Count part one has to show that the number of non-solutions is even for each W and provide an algorithm CountC. In the next section, we illustrate both parts by applying to the problem of testing whether a given bounded treewidth graph contains a k-cycle.

5.3 BOUNDED TREEWIDTH k-CYCLE

In this section we explain how *Cut and Count* technique can be used to solve k-cycle in bounded treewidth graph. BOUNDED TREEWIDTH k-CYCLE can be formally defined as follows

BOUNDED TREEWIDTH k-CYCLE	
Input:	Graph $G = (V, E)$, a nice tree decomposition $\mathbb{T}$ of width t and a positive integer k
Parameter:	t
Question:	Does there exists a simple cycle of length k in G

Let set of solutions, S contain pairs (X, M) where $X \subseteq E$ is a k-cycle and $M \subseteq X$ with $|M| = 1$. We defined S as set of *marked* k-cycles instead of k-cycles to make sure that we can create a set $\mathcal{C}$ such that for each element in S, there is only one consistent cut in $\mathcal{C}$. Note that any k-cycle with different markers are considered to be different solutions. For this reason we assign two random weights to each edge, one to represent it as part of a k-cycle and the other to represent it as a marked edge in a k-cycle. When we relax the requirement that X is a k-cycle to X is a cycle cover (i.e, X form a set of cycles) with $|X| = k$, we get candidate solutions.
The Cut part. Since the solution set contains pairs (X, M), we define weight function $w : E \times \{\mathbb{X}, \mathbb{M}\} \to \{1, 2, ..., N\}$ where $N = 4|E|$.

Definition 38. *A cut (V_1, V_2) of an undirected graph $G = (V, E)$ is consistent if $u \in V_1$ and $v \in V_2$ implies $(u, v) \notin E$ where $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. A consistently cut subgraph of G is a pair $(X, (X1, X2))$ such that $(X1, X2)$ is a consistent cut of $G[X]$ where X is a set of vertices or edges.*

Definition 39. *For an integer W we define:*

1. *R_W to be the set of candidate solutions, that is R_W is the set of all pairs (X, M), such that $X \subseteq E$ is a cycle cover, i.e., $deg_{G[X]}(u) = 2$ for every vertex $u \in V(X)$; $|X| = k$; $M \subseteq X$; $|M| = 1$ and $w(X \times \mathbb{X} \cup M \times \mathbb{M}) = W$*

2. *S_W to be the set of solutions, that is S_W is the set of all pairs (X, M), where $(X, M) \in R_W$ and X forms a cycle of length k.*

3. *$\mathcal{C}_W$ to be set of all pairs $((X, M), (X_1, X_2))$ such that: $(X, M) \in R_W$, $(X, (X_1, X_2))$ is a consistent cut subgraph of $G[X]$, and $V(M) \subseteq X_1$*

Observe that G contains a k-cycle if and only if there exists a W such that S_W is nonempty.
The Count part. We proceed to the count part by showing that candidate solutions that contain more than one cycle cancel modulo 2.

Lemma 40. *Let $G = (V, E)$ be a graph and $(X, M) \in R_W$. The number of consistently cut subgraphs $(X, (X_1, X_2))$ with $M \subseteq X_1$ is equal to $2^{cc(G[X])-1}$*

Proof. By definition, we know that for every consistently cut subgraph $(X, (X_1, X_2))$ and connected component C of $G[X]$, either $C \subseteq X_1$ or $C \subseteq X_2$. For the connected component containing M, the choice is fixed, and for all $cc(G[X]) - 1$ other

connected components we are free to choose a side of a cut, which gives $2^{cc(G[X])-1}$ possibilities leading to different consistently cut subgraphs. □

The following lemma shows that the first condition of Theorem 37 is indeed met:

Lemma 41. *Let $G, w, \mathcal{C}_W, S_W$ and R_W be as defined above. Then for every W, $|S_W| = |\mathcal{C}_W| \mod 2$*

Proof. By Lemma 40, we know that $|\mathcal{C}_W| = \sum_{X \in R_W} 2^{cc(G[X])-1}$. Thus $|\mathcal{C}_W| = |\{X \in R_W | cc(G[X]) = 1\}| \mod 2 = |S_W| \mod 2$ □

Now the only missing ingredient left is the sub-procedure CountC. This sub-procedure, which counts the cardinality of $\mathcal{C}_W$ modulo 2, is a standard application of dynamic programming:

Lemma 42. *Given $G = (V, E)$, an integer k, $w : E \times \{\mathbb{X}, \mathbb{M}\} \to \{1, 2, ..., N\}$ and a nice tree decomposition $\mathbb{T}$ of width t, there exists an algorithm that can determine $|\mathcal{C}_W|$ modulo 2 for every $0 \leq W \leq (k+1)N$ in $O^*(4^t)$ time.*

Proof. We use dynamic programming, but we first need some preliminary definitions. Recall that for a bag $x \in \mathbb{T}$ we denoted by V_x the set of vertices of all descendants of x, while by G_x we denoted the graph composed of vertices V_x and the edges E_x introduced by the descendants of x. We now define "partial solutions": For every bag $x \in \mathbb{T}$, integers $0 \leq i \leq k$, $0 \leq \mathsf{w} \leq (k+1)N$, $b \in \{0, 1\}$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}^{B_x}$ define

$$\begin{aligned}
R_x(i, b, \mathsf{w}) = \{(X, M) \mid M \subseteq X \subseteq E_x \ \wedge \ |X| = i \\
\wedge |M| = b \ \wedge \ w(X \times \mathbb{X} \cup M \times \mathbb{M}) = \mathsf{w}\} \\
\mathcal{C}_x(i, b, \mathsf{w}) = \{((X, M), (X_1, X_2)) \mid (X, M) \in R_x(i, b, \mathsf{w}) \ \wedge \ V(M) \subseteq X_1 \\
\wedge \ (X, (X_1, X_2)) \text{ is a consistently cut subgraph of } G[X]\} \\
A_x(i, b, \mathsf{w}, s) = |\{((X, M)(X_1, X_2)) \in \mathcal{C}_x(i, b, \mathsf{w}) \mid (s(v) = \mathbf{0} \Rightarrow degree_{G[X]}(v) = 0) \\
\wedge \ (s(v) = \mathbf{1}_j \Rightarrow (degree_{G[X]}(v) = 1 \ \wedge \ v \in X_j)) \\
\wedge \ (s(v) = \mathbf{2} \Rightarrow degree_{G[X]}(v) = 2)\}|
\end{aligned}$$

The intuition behind these definitions is as follows: the set $R_x(i, b, \mathsf{w})$ contains all sets $X \subseteq E_x$ that could potentially be extended to a candidate solution, subject

to an additional restriction that the cardinality and weight of the partial solution are equal to i and w, respectively. Similarly, $\mathcal{C}_x(i, b, \mathsf{w})$ contains consistently cut subgraphs, which could potentially be extended to elements of $\mathcal{C}$, again with the cardinality and weight restrictions. The number $A_x(i, b, \mathsf{w}, s)$ counts those elements of $\mathcal{C}_x(i, b, \mathsf{w})$ which additionally behave on vertices of B_x in a fashion prescribed by the sequence s. The value of $s(v)$ denotes the degree of v in $G[X]$ and, in case of degree one, $s(v)$ also stores information about the side of the cut v belongs to. We note that we do not need to store the side of the cut for v if its degree is 0 and 2, since it is not yet or no more needed. For vertices of degree 0 they are not part of any connected component in $G[X]$. For vertices of degree 2 the situation is more tricky. They are part of cut (that is, each such vertex is on some side of the cut in each counted object in $\mathcal{C}_x(i, b, \mathsf{w})$), but the information about the side of the cut will not be needed – we have a guarantee that no new edges will be added to that vertex (as 2 is the maximum degree). We need to remember the side of the cut to ensures that when we have a path in the currently constructed solution, both endpoints of the path are remembered to be on the same side of the cut, even though we no more needed to remember sides for the internal vertices of the path. The accumulators i, b and w keep track of the size of X, the size of M and the weight of (X, M), respectively.

To obtain all values $|\mathcal{C}_W| \mod 2$ it is enough to compute $A_r(k, b, W, \emptyset)$ modulo 2 for all values of W, since $|\mathcal{C}_W| = A_r(k, b, W, \emptyset)$ where r is the root of nice tree decomposition $\mathbb{T}$.

We now give the recurrence for $A_x(i, b, \mathsf{w}, s)$ that is used by the dynamic programming algorithm. In order to simplify notation denote by v the vertex introduced and contained in an introduce bag, by (u, v) the edge introduced in an introduce edge bag, and by y, z the left and right child of x in $\mathbb{T}$ if present.

- **Leaf bag:**

$$A_x(0, 0, 0, \emptyset) = 1$$

- **Introduce vertex bag:**

$$A_x(i, b, \mathsf{w}, s[v \to 0]) = A_y(i, b, \mathsf{w}, s) \tag{5.3.1}$$

The new vertex has degree zero and we do not impose any other constraints.

- **Introduce edge bag** For the sake of simplicity of the recurrence formula let us define a function $subs : \Sigma \to 2^{\Sigma}$ where $\Sigma = \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}$

Σ	$\mathbf{0}$	$\mathbf{1}_1$	$\mathbf{1}_2$	$\mathbf{2}$
$subs$	$\emptyset$	$\{\mathbf{0}\}$	$\{\mathbf{0}\}$	$\{\mathbf{1}_1, \mathbf{1}_2\}$

  Intuitively, for a given state $\alpha \in \Sigma$ the value $subs(\alpha)$ is the set of possible states a vertex can have before adding an incident edge.
  We can now write the recurrence for the introduce edge bag.

$$A_x(i, b, \mathsf{w}, s) = A_y(i, b, \mathsf{w}, s) \; + \sum_{\alpha_u \in subs(s(u))} \sum_{\alpha_v \in subs(s(v))} \sum_{i \in \{1,2\}} [(\alpha_u = \mathbf{1}_j \;\vee\; s(u) = \mathbf{1}_j) \;\wedge\; (\alpha_v = \mathbf{1}_j \;\vee\; s(v) = \mathbf{1}_j)] \Big(A_y\Big(i-1, b, \mathsf{w} - w((u,v) \times \mathbb{X}), s[u \to \alpha_u,\; v \to \alpha_v]\Big) + [j = 1] \; A_y\Big(i-1, b-1, \mathsf{w} - w(\{(u,v)\} \times \{\mathbb{X}, \mathbb{M}\}), s[u \to \alpha_u,\; v \to \alpha_v]\Big) \Big)$$

  To work the above recursive formula correctly we set a base case $A_x(i, -1, \mathsf{w}, s) = 0$ for all x, i, w, s. While introducing an edge either we can choose not to use the introduced edge or we can choose the edge with or without considering one of it as a marked edge.

- **Forget bag:**

$$A_x(i, b, \mathsf{w}, s) = A_y(i, b, w, s[v \to \mathbf{2}]) + A_y(i, b, \mathsf{w}, s[v \to 0])$$

  The forgotten vertex must have degree two or zero in $G[X]$.

- **Join bag:** For colorings $s_1, s_2, s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}^{B_x}$ we say that $s_1 + s_2 = s$ if for each $v \in B_x$ at least one of the following holds:

$$\begin{aligned} s_1(v) = \mathbf{0} \quad &\wedge \quad s(v) = s_2(v) \\ s_2(v) = \mathbf{0} \quad &\wedge \quad s(v) = s_1(v) \\ s_1(v) = s_2(v) = \mathbf{1}_j \quad &\wedge \quad s(v) = \mathbf{2} \end{aligned}$$

We can now write the recurrence for the join bags.

$$A_x(i,b,\mathsf{w},s) = \sum_{i_1+i_2=i} \sum_{b_1+b_2=b} \sum_{\mathsf{w}_1+\mathsf{w}_2=\mathsf{w}} \sum_{s_1+s_2=s} A_y(i_1,b_1,\mathsf{w}_1,s_1)A_z(i_2,b_2,\mathsf{w}_2,s_2)$$

It is easy to see that the Lemma can now be obtained by combining the above recurrence with dynamic programming. Note that as we perform all calculations modulo 2, we take only constant time to perform any arithmetic operation. Since each vertex in a bag can be colored with any of the four colors from $\{\mathbf{0},\mathbf{1}_1,\mathbf{1}_2,\mathbf{2}\}$ dynamic programming to calculate $|\mathcal{C}_W|$ for all $0 \leq W \leq (k+1)|E|$ will take $O^*(4^t)$ time. □

Theorem 43. *There exists a randomized algorithm that given a graph $G = (V, E)$ and a nice tree decomposition of width t solves* Bounded treewidth k-cycle *in $O^*(4^t)$ time. The algorithm cannot give false positives and may give false negatives with probability at most $\frac{1}{2}$.*

Proof. Run Algorithm 3 by setting $U = E \times \{\mathbb{X}, \mathbb{M}\}$ and CountC to be the algorithm implied by Lemma 42. The correctness follows from Theorem 37 by setting $S = \bigcup_W S_W$ and $\mathcal{C} = \bigcup_W \mathcal{C}_W$ and Lemma 41. It is easy to see that the time bound follows from Lemma 42. □

5.4 Solving k-PATH

In this section we show that there exists a randomized algorithm for k-PATH using DFS(Depth First Search) and algorithm for BOUNDED TREEWIDTH k-CYCLE that takes time $O^*(4^k)$

Theorem 44. *Given a graph $G = (V, E)$. There exists a polynomial time deterministic algorithm that outputs either a path of length k, or a tree decomposition of G of width at most $k - 1$.*

Proof. Without loss of generality we assume that G is connected. We first do a depth-first search and find a DFS tree T. If the depth of the tree is k, then output a path from root to a leaf of length k. Otherwise depth of the tree T is at most $k - 1$. Now we can construct a tree decomposition $\mathbb{T}$ as follows. Let

there are l leaves in the DFS tree T. Let $v_1, v_2, \ldots, v_l$ be the leaves in the order in which it is visited in the depth-first search. Now the tree decomposition $\mathbb{T} = (\{x_1, x_2, \ldots, x_l\}, \{(x_i, x_{i+1}) \mid 1 \leq i \leq l-1\})$ and B_{x_i} contains all the vertices in the path from root of the tree T to the leaf v_i. Clearly every bag will contain at most k vertices and hence the width of the tree decomposition is at most $k-1$. Now we need to show that $\mathbb{T}$ is indeed a tree decomposition. It is easy to see that $\cup_{x \in \mathbb{T}} B_x = V$. We can classify the edges in E as tree edges which are part of T, and non tree edges. For any tree edge, there exists bag that contains both end points of that edge because any tree edge is part of at least one path from root to a leaf node. Any non tree edge (u, v) will be from a vertex u to an ancestor v of u. Hence for any non tree edge there exist a bag that contains both of its end points. Now we need to show that for any vertex v, if $v \in B_{x_i}$ and $v \in B_{x_j}$ for $i < j$, then $v \in B_k$ for $i < k < j$. if $v \in B_{x_i}$ and $v \in B_{x_j}$ then v should be a common ancestor of v_i and v_j. So v should be an ancestor of v_k for any $i < k < j$. Hence $v \in B_k$ for any $i < k < j$. □

Theorem 45. *There exists a randomized algorithm that given graph $G = (V, E)$ solves k-PATH in $O^*(4^k)$ time. The algorithm cannot give false positives and may give false negatives with probability at most $\frac{1}{2}$*

Proof. We first do the algorithm implied by Theorem 44. If the output is a k-path, then we are done. Otherwise output of the algorithm is a tree decomposition $\mathbb{T}$ of width at most $k-2$. We know that any tree decomposition can be transformed into a nice tree decomposition of same width in polynomial time (refer Section 1.2.4 of Chapter 1). So we can transform tree decomposition $\mathbb{T}$ into a nice tree decomposition $\mathbb{T}'$ of width at most $k-2$. Then guess the end points (say s and t) of k-path in G and then solve BOUNDED TREEWIDTH k-CYCLE with input as graph $G' = (V(G), E(G) \cup (s, t))$ and $\mathbb{T}'$, as mentioned in Theorem 43. The correctness is clear from the correctness of Theorem 44 and Theorem 43. Also it is clear that the running time is equal to $O^*(4^k)$. □

6

Algebraic techniques

In this chapter we study two randomized algorithms for k-PATH – Koutis Williams approach [12, 20] and Narrow Sieve [4] for k-PATH. In both cases k-PATH is reduced to algebraic problems. In Koutis Williams approach, k-PATH is reduced to the problem of detecting a square free term of degree k in a multivariate polynomial which is given in the form of an arithmetic circuit. In Narrow Sieve for k-PATH, walks of length $k-1$ are associated with monomials in such a way that monomials corresponding to non-paths are cancelled out in a characteristic two field. Ultimately we arrive at the polynomial identity testing problem.

6.1 Koutis Williams approach

A randomized algorithm of running time $O^*(2^k)$ for ODD MULTILINEAR k-TERM (detecting a square free term with odd coefficient in a multivariate polynomial) was developed by Koutis [12]. In paper [12] a reduction from k-PATH to ODD MULTILINEAR $3k/2$-TERM is described. A simple reduction from k-PATH to ODD MULTILINEAR k-TERM was developed later by Ryan Williams [20]. Combining these two we get $O^*(2^k)$ randomized algorithm for k-PATH with constant one sided error.

6.1.1 Detecting square-free terms with odd coefficients

Let $X = \{x_1, ..., x_n\}$ and let $K[X]$ be the commutative ring of polynomials with coefficients from the field K. Any non-zero polynomial in $\mathbb{Z}_2[X]$ is by definition

a sum (or equivalently a set) of monomials. A monomial is called square-free or multilinear if it is linear in all its variables. The total degree of a monomial is the sum of the degrees of its variables. In general, any polynomial $P \in \mathbb{Z}_2[X]$ can be represented as an arithmetic circuit which is a directed acyclic graph with addition and multiplication gates, and terminals corresponding to the variables. The ODD MULTILINEAR k-TERM problem is formally defined as follows

ODD MULTILINEAR k-TERM	
Input:	An arithmetic circuit C that represents a polynomial $P \in \mathbb{Z}_2[X]$
Parameter:	k
Output:	Does there exist a multilinear term of degree less than or equal to k in P?

The main idea of the algorithm for this problem is the evaluation of the given polynomial over a suitably selected *commutative algebra*. This enables looking at the polynomial in two equivalent ways; its circuit representation allows for fast evaluation, whereas its expanded form as a sum of monomials allows us reasoning about its value in terms of the individual evaluations of its monomials. The intuition is that a commutative algebra may contain elements whose square is 0. This idea will be exploited to annihilate non-multilinear terms in the evaluation of P. It turns out that this can be done using commutative *group algebras* of $\mathbb{Z}_2^k$ (refer section 1.2.5 for basic definitions and facts). We now give an algorithm for the ODD MULTILINEAR k-TERM, that works with the assumption that P contains only multilinear terms of degree exactly k. We will then see how this restriction can be easily removed.

It may appear that given the two options for step 2, Algorithm 4 gives two algorithms. However, in Theorem 49 we will prove that option 2 is equivalent to option 1 and thus it just provides an alternative implementation; from this we will derive complexity claims. Soundness of the algorithm will be proved in Theorem 48, using option 1. If S $\subseteq \mathbb{Z}_2^k$ is a set of vectors, we denote by $\pi(S)$ their product in $\mathbb{Z}_2^k$ and let $\pi(\emptyset) = \vec{0}$. Note that $\pi(A)\pi(B) = \vec{0}$ if and only if $\pi(A) = \pi(B)$. We let J denote the element of $\mathbb{Z}_2[\mathbb{Z}_2^k]$ which is the sum of all vectors in $\mathbb{Z}_2^k$, that is $J = \sum_{v \in \mathbb{Z}_2^k} v$. We also say that an element w of $\mathbb{Z}_2[\mathbb{Z}_2^k]$ is *split* if it is the sum of exactly 2^{k-1} distinct vectors.

Algorithm 4 MULTILINEAR(C)

Input: An arithmetic circuit C that represents a polynomial $P \in \mathbb{Z}_2[X]$
Output: if there exists a multilinear term of degree k in P, output *yes* otherwise output *no*.

1. For each $x_i \in X$, independently pick a random vector $v_i \in \mathbb{Z}_2^k$
2. **[Option 1:]** Let $\bar{X}$ denote assignment $x_i \leftarrow \vec{0} + v_i$ for all i where $\vec{0}$ is the k-dimensional zero vector and $(\vec{0} + v_i) \in \mathbb{Z}_2[\mathbb{Z}_2^k]$. Evaluate $P(\bar{X})$ and if coefficient of $\vec{0}$ in $P(\bar{X})$ is equal to 1, then return *yes* otherwise return *no*.
2. **[Option 2:]** Let $b(t)$ denote the k dimensional vector containing the binary form of t, Λ_t denote the assignment $x_i \leftarrow 1+(-1)^{v_i^T b(t)}$ and $Z = \sum_{t=0}^{2^k-1} P(\Lambda_t)$. If Z is equal to $2^k \mod 2^{k+1}$ then return *yes*, otherwise return *no*.

By construction, each monomial evaluates to an element of the form $\Pi(V) = \Pi_{v \in V}(\vec{0} + v)$, where $V \subseteq \mathbb{Z}_2^k$. Using the fact that for all $v \in \mathbb{Z}_2^k$ we have $\vec{0}v = v$, we can expand $\Pi(V)$ into a sum, to get

$$\Pi(V) = \prod_{v \in V} (\vec{0} + v) = \sum_{S \subseteq V} \pi(S) \tag{6.1.1}$$

Lemma 46 ([12]). *If the vectors in $V \subseteq \mathbb{Z}_2^k$ are linearly dependent over $\mathbb{Z}_2$, $\Pi(V)$ evaluates to 0. If the vectors in V are linearly independent, $\Pi(V)$ is a sum of $2^{|V|}$ distinct vectors (including $\vec{0}$).*

Proof. By definition, when the vectors in V are linearly dependent, there is $V' \subseteq V$ such that $\pi(V') = \vec{0}$. Then for all $S \subseteq V'$ we have $\pi(S)\pi(V' \setminus S) = \vec{0}$, which implies that $\pi(S) = \pi(V' \setminus S)$. Hence, every term in the sum expansion (equality (6.1.1)) of $\Pi(V')$ is generated an even number of times, which gives us $\Pi(V') = 0$. This in turn implies $\Pi(V) = 0$, because $\Pi(V) = \Pi(V')\Pi(V \setminus V')$. This shows the first part of the Lemma. For the second part of the Lemma we observe that for all $S_a \neq S_b \subseteq V$ we have $\pi(S_a) \neq \pi(S_b)$. To see why, note that if $\pi(S_a) = \pi(S_b)$, then $\pi(S_a)\pi(S_b) = \pi(S_a S_b) = \vec{0}$, which implies that the vectors in $S_a \cup S_b \setminus (S_a \cap S_b)$ are linearly dependent, a contradiction. Therefore, since there are $2^{|V|}$ possible subsets of V (including $\emptyset$), $\Pi(V)$ is a sum of $2^{|V|}$ distinct vectors (including $\vec{0}$). □

Lemma 47 ([12]). *Let $P_{k-1} \in \mathbb{Z}_2[X]$ be a sum of multilinear monomials of degree*

exactly $k-1$. *(i) For all assignments* $\bar{X}$ *of the form* $x_i \leftarrow (\vec{0}+v_i)$, $P_{k-1}(\bar{X})$ *is either split, or equal to* 0, *or equal to* J. *(ii) In the case* $P_{k-1}(\bar{X})$ *is split, we have* $\Pr_{v\in\mathbb{Z}_2^k}\left[(\vec{0}+v)P_{k-1}(\bar{X})=J\right] = \Pr_{v\in\mathbb{Z}_2^k}\left[(\vec{0}+v)P_{k-1}(\bar{X})=0\right] = 1/2$

Proof. Let $P_{k-1}=\sum_j M_j$ where each M_j is a monomial of degree $k-1$. We will derive the Lemma by looking at $I=(\vec{0}+v)P_{k-1}(\bar{X})$ for a proper vector v. Note that $\mathbb{Z}_2^k$ contains 2^k vectors. Lemma 46 then implies that for all $v\in\mathbb{Z}_2^k$ and all M_j, we have $(\vec{0}+v)M_j(\bar{X})=0$ or $(\vec{0}+v)M_j(\bar{X})=J$. Therefore, we have $I=0$ or $I=J$, so the coefficient of any vector in I completely determines the value of I.

Suppose $P_{k-1}(\bar{X})\neq 0$ and $P_{k-1}(\bar{X})\neq J$. Now assume that $P_{k-1}(\bar{X})$ is a sum of t distinct vectors, where $1\leq t<2^k$. We have

$$I=(\vec{0}+v)P_{k-1}(\bar{X})=P_{k-1}(\bar{X})+vP_{k-1}(\bar{X}).$$

Since $vv_1=vv_2$ implies $v_1=v_2$, it must be that $vP_{k-1}(\bar{X})$ is a sum of t distinct vectors in $\mathbb{Z}_2^k$. Hence, every vector in the expansion of I is generated $0,1$ or 2 times. If some vector w is generated two times, its coefficient in I will be 0, and hence $I=0$.

Now pick a vector v such that $vP_{k-1}(\bar{X})$ contains a vector w which is not in $P_{k-1}(\bar{X})$; this is clearly always possible. In that case the coefficient of w in I is 1, thus $I=J$. In addition I is a sum of at most $2t$ vectors, and since J is the sum of 2^k vectors, we must have $t\geq 2^{k-1}$. If $t>2^{k-1}$, a simple pigeon hole argument shows that there must be a vector w' which is generated two times in I, implying that $I=0$. This is a contradiction, so we must have $t=2^{k-1}$. The claim (ii) follows from the fact that $t=2^{k-1}$ and the observation that I contains the vector $\vec{0}$ with probability $1/2$, with respect to the choice of v. □

Now we prove the soundness of the algorithm.

Theorem 48 ([12])**.** *If* P *does not contain a multilinear term, the Algorithm 4 returns no. Otherwise it returns yes with probability greater than* $1/4$.

Proof. For the first claim, note that every monomial M which is not multilinear can be written as x_i^2M' for some variable x_i and monomial M'. Now observe that $\bar{x}_i^2=(\vec{0}+v_i)^2=0$. Hence $\bar{x}_i^2M'=0$. So, if P does not contain multilinear terms, all its terms evaluate to 0. Thus, $P(\bar{X})=0$, and the algorithm returns *no*. We

will show the other direction using induction on the number of multilinear terms in P. Recall our assumption that all these terms have degree exactly k.

Base case: Assume P contain only one multilinear monomial of degree k. Let $V = \{v_1, ..., v_k\}$ be k random vectors drawn independently from $\mathbb{Z}_2^k$. The probability that a multilinear monomial of degree k evaluates to J is by construction equal to $\Pr[\Pi_{i=1}^{k}(\vec{0} + v_i) = J]$. By Lemma 46, this is equal to the probability that the vectors in V are linearly independent. By standard linear algebra facts, given that $\{v_1, ..., v_{j-1}\}$ are linearly independent, the vectors $\{v_1, ..., v_j\}$ are linearly independent if and only if v_j is not in the vector space S generated by $\{v_1, ..., v_{j-1}\}$. There are exactly 2^{j-1} distinct linear combinations of the $j-1$ vectors, so there are $2^k - 2^{j-1}$ vectors that are not in S. Hence,

$$\Pr\left[\prod_{i=1}^{j}(\vec{0} + v_i) \neq 0\right] = \left(1 - \frac{2^{j-1}}{2^k}\right) \Pr\left[\prod_{i=1}^{j-1}(\vec{0} + v_i) \neq 0\right] \geq \prod_{i=1}^{k}\left(1 - \frac{1}{2^i}\right) > \frac{1}{4}$$

Inductive argument: If P is not a single monomial, then there is a variable x such that $P = xP_{k-1} + P'$ where x does not appear in $P' \neq 0$, and P_{k-1} is a sum of multilinear terms of degree $k-1$. Using Lemma 47 we can consider all possible cases under which P evaluates to J, to get

$$\begin{aligned}
\Pr[P = J] &= \Pr[xP_{k-1} + P' = J] \\
&\geq \Pr[P' = J] \Pr[P_{k-1} \textit{ is split} / P' = J] \Pr[xP_{k-1} = 0 / P_{k-1} \textit{ is split}] \\
&\quad + \Pr[P' = 0] \Pr[P_{k-1} \textit{ is split} / P' = 0] \Pr[xP_{k-1} = J / P_{k-1} \textit{ is split}] \\
&\geq \frac{1}{2} \Pr[P_{k-1} \textit{ is split}] \\
&\geq \Pr[xP_{k-1} = J]
\end{aligned}$$

We derived above probability using conditional probability and the fact that $\Pr[xP_{k-1} = J / P_{k-1} \textit{ is split}] = \Pr[xP_{k-1} = 0 / P_{k-1} \textit{ is split}] = 1/2$ (Lemma 47). The probabilities are taken with respect to the random assignment $\bar{X}$. The polynomial xP_{k-1} contains less monomials than P, hence the inductive hypothesis applies and we get $\Pr[P = J] \geq 1/4$ □

Let A be a commutative algebra and $\bar{Y}$ be an assignment $x_i \leftarrow y_i \in A$, for $i = 1, ..., n$. We denote by $P_A(\bar{Y})$ the evaluation of P at $\bar{Y}$ over A.

Theorem 49 ([12]). *Options* 1 *and* 2 *for step* 2 *of Algorithm 4 are equivalent. Furthermore, if the input circuit* C *can be evaluated over the integers modulo* 2^{k+1} *in time* t *and space* s*, option* 2 *can be performed in* $O((nk+t)2^k)$ *time and* $O(nk+s)$ *space.*

Proof. Let C be an arithmetic circuit with n variables X and $P \in \mathbb{Z}[X]$ be the polynomial represented by C. Also, let $\bar{X}$ be the assignment $x_i \leftarrow (\vec{0} + v_i)$, as defined in Algorithm 4. Let ρ be the representation of $\mathbb{Z}[\mathbb{Z}_2^k]$ as mentioned in the Section 1.2.5 of Chapter 1. Observe that

$$P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X}) = \sum_{g \in \mathbb{Z}_2^k} a_g g \quad \Rightarrow \quad P_{\mathbb{Z}_2[\mathbb{Z}_2^k]}(\bar{X}) = \sum_{g \in \mathbb{Z}_2^k} (a_g \bmod 2) g \tag{6.1.2}$$

Thus, it is enough to consider the parity of the coefficient of $\vec{0}$ in $P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})$. Moving to $\mathbb{Z}[\mathbb{Z}_2^k]$ allows us working with the matrix representation of $P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})$, which is given by

$$\rho\left(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})\right) = \sum_{g \in \mathbb{Z}_2^k} a_g \rho(g) \tag{6.1.3}$$

For each $g \in \mathbb{Z}_2^k$, $\rho(g)$ is a permutation matrix of dimension 2^k with zeros in the diagonal, with the exception of the identity $\vec{0}$ of $\mathbb{Z}_2^k$, for which $\rho(\vec{0})$ is the identity matrix. Hence all the diagonal entries of $\rho(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X}))$ are equal. This, in combination with equality (6.1.2) implies that

$$\begin{aligned} P_{\mathbb{Z}_2[\mathbb{Z}_2^k]}(\bar{X}) = 0 \quad &\Rightarrow \quad trace\left(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})\right) = 0 \bmod 2^{k+1} \\ P_{\mathbb{Z}_2[\mathbb{Z}_2^k]}(\bar{X}) = J \quad &\Rightarrow \quad trace\left(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})\right) = 2^k \bmod 2^{k+1} \end{aligned}$$

Now instead of evaluating P at $x_i \leftarrow (\vec{0} + v_i)$ over $\mathbb{Z}[\mathbb{Z}_2^k]$, we can equivalently evaluate it at $x_i \leftarrow \rho(\vec{0} + v_i)$ over the isomorphic matrix algebra $\mathcal{M} = \rho(\mathbb{Z}[\mathbb{Z}_2^k])$, and then compute (modulo 2^{k+1})

$$trace\left(P_{\mathcal{M}}(\bar{X})\right) = trace\left(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})\right) \tag{6.1.4}$$

By the representation theory of $\mathbb{Z}_2^k$ (Section 1.2.5 of Chapter 1), there is a matrix U of dimension 2^k such that for all $v \in \mathbb{Z}_2^k$, $\rho(v) = U\Lambda_v U^{-1}$, where Λ_v is a diagonal matrix with the eigenvalues of $\rho(v)$ and i^{th} eigenvalue of $\rho(v)$ (i^{th}

diagonal entry in Λ_v) will be $(-1)^{v^T b(i-1)}$ where $b(i)$ is the k-bit binary form of i. Let Λ_i denote the diagonal matrix containing the eigenvalues of $\rho(\vec{0} + v_i)$ and $\bar{\Lambda}$ denote the assignment $x_i \leftarrow \Lambda_i$. Let $\Lambda_{i,j}$ denote the j^{th} diagonal entry of Λ_i and let $\bar{\Lambda_j}$ denote the assignment $x_i \leftarrow \Lambda_{i,j}$ (note that $\Lambda_{i,j} = 1 + (-1)^{v_i^T b(j-1)}$). Then using the fact that matrices in $\mathcal{M}$ are simultaneously diagonalizable and simple facts about the trace of matrices we get

$$trace\left(P_{\mathcal{M}}(\rho(\bar{X}))\right) = trace\left(UP_{\mathcal{M}}(\bar{\Lambda})U^{-1}\right) = trace\left(P_{\mathcal{M}}(\bar{\Lambda})\right) = \sum_{j=1}^{2^k} P_{\mathbb{Z}_{2^k+1}}(\bar{\Lambda_j})$$

This completes the proof for the equivalence of options 1 and 2.

We have reduced the original problem to 2^k evaluations of P and the summation of the outputs over $\mathbb{Z}_{2^k+1}$. The 2^k evaluations can be performed sequentially, re-using the space, while the output sum is updated. The algorithm needs to maintain in the memory the assignment $\bar{X}$ which takes space $O(kn)$. For each j, the algorithm computes the input $\bar{\Lambda_j}$ in $O(nk)$ time. The evaluation of $P(\bar{\Lambda_j})$ can be done in time $O(t)$, and space $O(nk + s)$, by assumption. Hence the total time is $O((nk + t)2^k)$ and the space requirement is $O(nk + s)$. □

Remark 50. *If the smallest multilinear term in P has degree $k-j$, we can consider $P_j = (y_1 \dots y_j)P$. By Lemma 46, any term of degree greater than k always evaluates to 0. Hence we can run Algorithm 4 with P_j for all $1 \leq j < k$ with the assumed restriction (i.e, input contain multilinear terms of degree exactly k, if exists) and gives output as yes if at least one execution gives yes answer.*

6.1.2 Reducing k-PATH to ODD MULTILINEAR k-TERM

A simple reduction from k-PATH to ODD MULTILINEAR k-TERM is described in [20]. Let $G = (V, E)$ be a the n vertex input graph of k-PATH problem. Let A be the $n \times n$ adjacency matrix of G. Define a matrix $B[i, j] = A[i, j]x_i$. Let $\vec{1}$ be the row n-vector of all 1's, and $\vec{x}$ is the row n-vector defined by $\vec{x}[i] = x_i$. Define the k-walk polynomial to be $P(X) = \vec{1}.B^{k-1}.\vec{x}^T$. Following proposition is an easy observation.

Proposition 51 ([20]). $P(X) = \sum_{v_{i_1},\dots,v_{i_k} \text{ is a walk in } G} x_{i_1} \dots x_{i_k}$

$P(X)$ can be implemented using a circuit C where the number of multiplication and addition gates will be exactly equal to the number of multiplication and addition required to compute the product $\vec{1}.B^{k-1}.\vec{x}^T$. So the number of gates in C is bounded by $O(n^2 \log k)$. Since the size of the circuit C is polynomial in n, the time and space required to evaluate $P(\bar{X})$ over $\mathbb{Z}_{2^{k+1}}$ for any assignment $\bar{X}$ will be bounded by polynomial in n. Combining this with Theorem 49 we get the result that k-PATH can be solved in $O^*(2^k)$ time with no false positives and false negatives with probability at most $1/4$.

6.2 Narrow Sieve for k-PATH

The idea behind this algorithm is to create a multivariate polynomial from an instance of k-PATH such that each monomial corresponds to a k-walk. There exist a unique monomial for any fixed path of length $k-1$ with some probability (exponentially small in k). Any monomial in the polynomial corresponding to a k-walk which is not a k-path, will appear even number of times. Then the problem reduces to polynomial identity testing where polynomial is considered over a field of characteristic two. In this section we consider the following variant of k-PATH.

k-PATH	
Input:	An undirected graph $G = (V, E)$, a vertex $s \in V$ and a positive integer k
Parameter:	k
Output:	Does there exists a path of length $k-1$ starting at s

For convenience we denote k-PATH for the above mentioned problem in this section and it is easy to see that solving k-path starting at a special vertex in time $O^*(c^k)$ leads to the solution of general k-PATH in time $O^*(c^k)$.

6.2.1 Overview

If the input graph $G = (V, E)$ is randomly partitioned into two sets V_1 and V_2, then with good probability any path of length $k-1$ in G has roughly $k/2$ vertices in $G[V_1]$ and $k/4$ edges in $G[V_2]$. We label the vertices in V_1 and edges in V_2 and each *labeled k-walk* will be associated with a monomial. We say a k-walk is *bijective* if

each occurrence on the walk of a vertex in $G[V_1]$ and of an edge in $G[V_2]$ receives a unique label. Our objective is to detect *bijective labeled k-path*. We will use an inclusion exclusion sieve to cancel all walks that are not bijectively labeled. To cancel *bijectively labeled k-walks* which are not paths, we associate monomials such that a monomial corresponding to such a walk will appear even number of times.

6.2.2 Labeled Admissible Walks and Monomials

A *k-walk* in a graph G can be thought of as a string of length $2k-1$ such that

(a) each odd position contains a vertex of G

(b) each even position contains an edge of G

(c) for every $i = 1, 2, ..., k-1$, the edge at position $2i$ joins in G the vertices at positions $2i-1$ and $2i+1$.

For a subset $B \subseteq A$ and a string $a_1a_2...a_l$, introduce the notation

$$B\{a_1a_2...a_l\} = \{(a_i, i) \mid a_i \in B\}$$

Definition 52 ([4]). *Let $G = (V, E)$ be a graph and let s be a fixed vertex of G. Partition the vertex set into two disjoint sets $V = V_1 \cup V_2$. Denote by E_1 and E_2 the set of edges of G with both ends in V_1 and V_2 respectively. Let k, k_1, l_2 be non negative integers. we say that a* k-walk *W in G is admissible if*

(a) W starts at s

(b) $|V_1\{W\}| = k_1$

(c) $|E_2\{W\}| = l_2$ and

(d) W is $V_2EV_1EV_2$-palindromeless (i.e, W does not contain a palindrome substring which is $V_2EV_1EV_2$-string)

Lemma 53 (Admissibility [4]). *Let k_1, l_2 be non negative integers and let P be a k-path in $G = (V, E)$. For a partition (V_1, V_2) selected uniformly at random, we have*

$$\Pr\Big(|V_1\{P\}| = k_1 \text{ and } |E_2\{P\}| = l_2\Big) = 2^{-k}\binom{k_1+1}{k-k_1-l_2}\binom{k-k_1-1}{l_2}.$$

Proof. There are 2^k strings of length k over the alphabet $\{1,2\}$. The probability in question is exactly the fraction of such strings that have exactly k_1 1-positions and exactly l_2 22-substrings. There are exactly k_1+1 positions where to interleave the k_1 1s with substrings of 2s. Each such substring of length j contributes exactly $j-1$ 22-substrings. The total number of 2s is $k-k_1$, so there must be $k-k_1-l_2$ substrings of 2s. The positions where the substrings interleave the 1s are allocated by the first binomial coefficient. It remains to allocate the lengths of the strings. The total length is $k-k_1$, and each of the $k-k_1-l_2$ strings must have length at least 1. Thus we have to allocate $k-k_1$ 2s to $k-k_1-l_2$ distinct bins such that each bin get at least one. The second binomial coefficient carries out this allocation. □

A fixed k-path starting at s is admissible with positive probability if $k_1+l_2 \leq k-1 \leq 2k_1+l_2$. Let us now derive an asymptotic approximation for the probability in Lemma 53. According to Stirling's approximation,

$$j! \approx \sqrt{2\pi j}\left(\frac{j}{e}\right)^j \tag{6.2.1}$$

Let us abbreviate

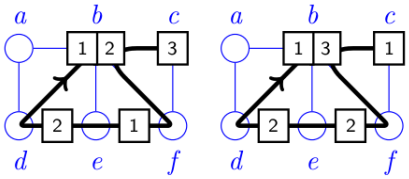
$$\left\langle {a \atop b} \right\rangle = \left(\frac{b}{a}\right)^{-b}\left(1-\frac{b}{a}\right)^{-a+b}$$

From Stirling's formula (6.2.1) it follows that $\binom{a}{b} = \theta^*(\langle {a \atop b} \rangle)$. We can thus approximate the probability in Lemma 53

$$\Pr\left(|V_1\{P\}| = k_1 \text{ and } |E_2\{P\}| = l_2\right) = \theta^*\left(2^{-k}\left\langle {k_1 \atop k-k_1-l_2}\right\rangle\left\langle {k-k_1 \atop l_2}\right\rangle\right).$$

Definition 54 (Labeled Admissible walks)**.** *Let $K_1 = \{1,2,...,k_1\}$ be a set of k_1 labels and $L_2 = \{1,2,...,l_2\}$ be a set of l_2 labels. Let W be an admissible walk. Let $\kappa : V_1\{W\} \to K_1$ and $\lambda : E_2\{W\} \to L_2$ be arbitrary functions. The three-tuple (W,κ,λ) is a labeled admissible walk, i.e, in a labeled walk each position in W that contains a vertex in V_1 gets assigned a label in K_1 by κ and each position in W that contains an edge in E_2 gets assigned a label in L_2 by λ. We say that the labeling is bijective if both κ and λ are bijections.*

Example. Consider two labelings of the same walk W,



The walk is admissible with parameters $s = c, k = 6, k_1 = 3$, and $l_2 = 2$. Both labelings associate a label with each position of W that contains a symbol from $V_1 = \{a, b, c\}$ or $E_2 = \{de, ef\}$. We have $V_1\{W\} = \{(b, 3), (b, 11), (c, 1)\}$, that is, there are three occurrences of symbols from V_1 in W; in particular the symbol b occurs at the 3^{rd} and the 11^{th} position. Similarly, we have $E_2\{W\} = \{(ef, 6), (de, 8)\}$. The labeling on the left is

$$\lambda(b, 3) = 2,\ \lambda(b, 11) = 1,\ \lambda(b, 1) = 3,\ \kappa(ef, 6) = 2,\ \kappa(de, 8) = 1.$$

We observe that this labeling is bijective since λ and κ are bijections. The labeling on the right is

$$\lambda(b, 3) = 3,\ \lambda(b, 11) = \lambda(c, 1) = 1,\ \kappa(ef, 6) = \kappa(de, 8) = 2,$$

and not bijective. In fact, λ avoids the label 2, and κ avoids the label 1.

Fingerprinting and identifiability: We associate with each labeled admissible walk a *monomial* (or "*fingerprint*") that we use to represent the labeled admissible walk in sieving. The sieve operates over a multivariate polynomial ring with the coefficient field $\mathbb{F}_{2^b}$ (the finite field of order 2^b) and the following indeterminate. Introduce one indeterminate x_e for each edge $e \in E$. Introduce one indeterminate $y_{v,i}$ for each pair $(v, i) \in V_1 \times K_1$. Introduce one indeterminate $z_{e,i}$ for each pair $(e, i) \in E_2 \times L_2$. Let (W, κ, λ) be a labeled admissible walk. Associate with (W, κ, λ) the *monomial* (*fingerprint*)

$$m(W, \kappa, \lambda) = \prod_{(e,j)\in E\{W\}} x_e \prod_{(v,i)\in V_1\{W\}} y_{v,\kappa(v,i)} \prod_{(z,i)\in E_2\{W\}} z_{e,\lambda(e,i)}$$

The following lemma is immediate.

Lemma 55 (Identifiability [4])**.** *The monomial $m(W, \kappa, \lambda)$ of a labeled admissible*

walk (W, κ, λ) *uniquely determines the edges and their multiplicities of occurrence in* W*. In particular, any path is uniquely identified. Furthermore, if* W *is a path and* κ, λ *are bijections, then* $m(W, \kappa, \lambda)$ *uniquely identifies* $m(W, \kappa, \lambda)$*.*

6.2.3 Sieving for bijectively labeled admissible paths

Now our objective is to define a polynomial such that it will contain monomials corresponding to bijectively labeled admissible walks. Furthermore any bijectively labeled admissible walk W which is not a path can be paired with another bijectively labeled admissible walk W' such that monomials corresponding to W and W' will be same. To get rid of non bijectively labeled k-walks we can use inclusion exclusion principle. Let $\mathcal{L}$ denote the set of all labeled admissible walks and $\mathcal{B}$ denote the set of all bijectively labeled admissible walks. For $I_1 \subseteq K_1$ and $J_2 \subseteq L_2$, denote by $\mathcal{L}[I_1, J_2]$ the set of all labelled admissible walks that avoids the labels in I_1 and J_2. By the principle of inclusion exclusion, we have

$$\sum_{(W,\kappa,\lambda)\in\mathcal{B}} m(W,\kappa,\lambda) = \sum_{I_1\subseteq K_1}\sum_{J_2\subseteq L_2}(-1)^{|I_1|+|J_2|}\sum_{(W,\kappa,\lambda)\in\mathcal{L}[I_1,J_2]} m(W,\kappa,\lambda) \tag{6.2.2}$$

Now we show that monomials corresponding to bijectively labeled non-paths cancels in a field of characteristic 2. Let partition $\mathcal{B}$ into $\mathcal{B} = \mathcal{P} \cup \mathcal{R}$, where $\mathcal{P}$ consists of bijectively labeled admissible paths, and $\mathcal{R}$ consists of bijectively labeled admissible non-paths. Accordingly, the left-hand side of (6.2.2) splits into

$$\sum_{(W,\kappa,\lambda)\in\mathcal{B}} m(W,\kappa,\lambda) = \sum_{(W,\kappa,\lambda)\in\mathcal{P}} m(W,\kappa,\lambda) + \sum_{(W,\kappa,\lambda)\in\mathcal{R}} m(W,\kappa,\lambda)$$

We show that the rightmost sum vanishes. To this end, let us first define that an *involution* is a permutation that is its own inverse. We claim that it suffices to construct a fixed-point-free involution $\phi : \mathcal{R} \to \mathcal{R}$ with $m(W, \kappa, \lambda) = m(\phi(W, \kappa, \lambda))$ for all $(W, \kappa, \lambda) \in R$. To construct a such a fixed-point-free involution ϕ, we observe that every walk W that is not a path contains at least one closed subwalk. In particular, W contains a first closed subwalk, that is, the closed subwalk C with the property that C is the unique closed subwalk in the prefix SC of $W = SCT$. We denote the first closed subwalk of W by $C(W)$ and by $c(W)$ the first (and hence also the last) vertex of $C(W)$.

Let us partition $\mathcal{R}$ into two disjoint sets, $\mathcal{R}_1$ and $\mathcal{R}_2$, where

$$\begin{aligned}\mathcal{R}_1 &= \{(W,\kappa,\lambda) \in \mathcal{R} \mid c(W) \in V_1\} \\ \mathcal{R}_2 &= \{(W,\kappa,\lambda) \in \mathcal{R} \mid c(W) \in V_2\}\end{aligned}$$

We proceed to construct the pairing ϕ on these two sets.

The pairing on $\mathcal{R}_1$: Select an arbitrary $(W,\kappa,\lambda) \in R_1$. Let j and l be the positions of W that contain the symbol $c(W)$ and constitute the ends of $C(W)$. For brevity, let us write c for $c(W)$. Because $c \in V_1$, we have $(c,j),\ (c,l) \in V_1\{W\}$. Define κ' to be identical to κ except that

$$\kappa'((c,j)) = \kappa((c,l)), \qquad \kappa'((c,l)) = \kappa((c,j)).$$

Observe that $\kappa((c,j)) \neq \kappa((c,l))$ because (W,κ,λ) is bijectively labeled. Thus $\kappa' \neq \kappa$. Furthermore, we have $m(W,\kappa,\lambda) = m(W,\kappa',\lambda)$. Thus, we can set $\phi(W,\kappa,\lambda) = (W,\kappa',\lambda)$ to obtain the desired fixed-point-free involution on $\mathcal{R}_1$. Indeed, $\phi(W,\kappa,\lambda) = (W,\kappa',\lambda) \neq (W,\kappa,\lambda)$ and $\phi^2(W,\kappa,\lambda) = (W,\kappa,\lambda)$.

The pairing on $\mathcal{R}_2$: Select an arbitrary $(W,\kappa,\lambda) \in \mathcal{R}_2$. Let $C = C(W)$ and let $S,\ T$ be strings such that

$$W = SCT. \tag{6.2.3}$$

Let us define the string W' by reversing C in W, that is, $W' = S\overleftarrow{C}T$. We observe that W' is an admissible walk in G and $c(W) = c(W')$. Let j and l be the positions of W that contain the symbol $c(W)$ and constitute the ends of $C(W)$. Define the permutation $\rho : \{1,2,...,k\} \to \{1,2,...,k\}$ by

$$\rho(i) = \begin{cases} i & \text{if } i < j \text{ or } i > l \\ l - i + j & \text{if } j \leq i \leq l \end{cases}$$

Let denote the symbol at the i^{th} position of W by w_i. Observe that $w'_{\rho(i)} = w_i$ for

all i. We now introduce a labeling κ', λ' of W' using the labeling κ, λ of W.

$$\kappa'\left(\left(w'_{\rho(i)}, \rho(i)\right)\right) = \kappa\left((w_i, i)\right)$$
$$\kappa'\left(\left(w'_{\rho(i)}, \rho(i)\right)\right) = \kappa\left((w_i, i)\right)$$

Now set $\phi(W, \kappa, \lambda) = (W', \kappa', \lambda')$ and observe that $\phi(W, \kappa, \lambda) \in \mathcal{R}_2$, $\phi^2(W, \kappa, \lambda) = (W, \kappa, \lambda)$, and $m(W, \kappa, \lambda) = m(W', \kappa', \lambda')$.

Now we need to prove that $\phi(W, \kappa, \lambda) \neq (W', \kappa', \lambda')$. There are two cases to consider, depending on C. In the first case, C is not a palindrome, that is, $C \neq \overleftarrow{C}$. Thus, $W' \neq W$ and hence $(W', \kappa', \lambda') \neq (W, \kappa, \lambda)$. In the second case, C is a palindrome. Since C is a closed walk, the string C has odd length at least 3. In particular, the length 3 (that is, a palindrome of the form ueu with $u \in V_2$ and $e \in E$) cannot occur because G has no loop edges. For palindromes of length 5, the only possibility is that C is a $V_2E_2V_2E_2V_2$-palindrome. Indeed, C can neither be a $V_1EV_1EV_1$-palindrome nor a $V_1EV_2EV_1$-palindrome because $c(W) \in V_2$. Furthermore, C cannot be a $V_2EV_1EV_2$-palindrome because such palindromes by definition do not occur in the admissible walk W. Thus, for length 5 the only possibility is a $V_2EV_2EV_2$-palindrome, that is, a $V_2E_2V_2E_2V_2$-palindrome. Such a palindrome contains two occurrences of an edge in E_2 that are in ρ-corresponding positions. These occurrences get different labels under λ and λ'. Thus, $(W', \kappa', \lambda') \neq (W, \kappa, \lambda)$. Finally, we observe that C cannot have length more than 5, because a palindrome of length 7 or more must include a palindrome of length 5, which would contradict the assumption that C is the first closed subwalk in W.

6.2.4 The Algorithm

Now we are ready to describe the algorithm for k-PATH. First consider the following lemma

Lemma 56 (DeMillo-Lipton-Schwartz-Zippel [8, 18])**.** *Let $p(x_1, x_2, ..., x_n)$ be a nonzero polynomial of total degree at most d over the finite field $\mathbb{F}_q$. Then, for $a_1, a_2, ..., a_n \in \mathbb{F}_q$ selected independently and uniformly at random,*

$$\Pr\left(p(a_1, a_2, ..., a_n) \neq 0\right) \geq 1 - \frac{d}{q}$$

Now let us assume that the parameters k, k_1, l_2 have been fixed so that $k_1+l_2 \leq k-1 \leq 2k_1+l_2$ (we will fix it later). Consider the randomized algorithm as shown in Algorithm 5.

Algorithm 5 SIEVE(G, s, k, k_1, l_2)

Input: A graph $G=(V,E)$, $s \in V$, positive integers k, k_1, l_2
Output: if there exists a path of length $k-1$ starting at s, output *yes* otherwise output *no*.

1. Select an ordered partition (V_1, V_2) of V uniformly at random.
2. Evaluate the polynomial

$$P(X) = \sum_{I_1 \subseteq K_1} \sum_{J_2 \subseteq L_2} (-1)^{|I_1|+|J_2|} \sum_{(W,\kappa,\lambda) \in \mathcal{L}[I_1,J_2]} m(W,\kappa,\lambda)$$

at an assignment $\bar{X}$ selected independently and uniformly at random from $\mathbb{F}_{2^{\log 6k}}$.

3. If $P(\bar{X}) \neq 0$ output *yes*, otherwise output *no*

Now we give dynamic programming to evaluate the sum $\sum_{(W,\kappa,\lambda)\in\mathcal{L}[I_1,J_2]} m(W,\kappa,\lambda)$. For parameters k, k_1, l_2, s and string $T = t_1t_2t_3t_4t_5$ over the alphabet $V \cup E$ we compute

$$M(k,k_1,l_2,T) = \sum_{\substack{(W,\kappa,\lambda)\in\mathcal{L}[I_1,J_2] \\ T \text{ is a suffix of } W}} m(W,\kappa,\lambda). \tag{6.2.4}$$

Now by taking the sum over all T, we obtain the sum $\sum_{(W,\kappa,\lambda)\in\mathcal{L}[I_1,J_2]} m(W,\kappa,\lambda)$. The recursion for (6.2.4) is as follows

$$\begin{aligned}
&M(k,k_1,l_2,t_1t_2t_3t_4t_5) \\
&\quad = [t_1t_2t_3t_4t_5 \text{ is a walk and not a } V_2EV_1EV_2\text{-palindrome }] \\
&\quad \times \sum_{\substack{e\in E \\ e=(v,t_1)}} \Big([t_5 \notin V_1] + [t_5 \in V_1] \sum_{j \in K_1 \setminus I_1} y_{t_5,j}\Big)\Big([t_4 \notin E_2] + [t_4 \in E_2] \sum_{j \in L_2 \setminus J_2} z_{t_4,j}\Big) \\
&\qquad \times M(k-1, k_1 - [t_5 \in V_1], l_2 - [t_4 \in E_2], vet_1t_2t_3).
\end{aligned}$$

To set up the base case for recursion, we observe that $M(k,k_1,l_2,T)$ can be computed for all $0 \leq k_1, l_2 \leq k = 3$ and all $T = t_1t_2t_3t_4t_5$ in time polynomial in n.

Furthermore, $M(k, k_1, l_2, T) = 0$ for all $k_1 > k$ or $l_2 > k$, or $k_1 < 0$ or $l_2 < 0$.

Consequently for any given assignment of values for the variables Algorithm 5 can evaluate $P(X)$ in $O(2^{k_1+l_2}k^5n^4)$ arithmetic operations over $\mathbb{F}_{2^{\log 6k}}$. Now we can complete the algorithm by optimizing the parameters for running time and $\Omega(1)$ probability of success. Denoting the probability that a k-path P starting at s is admissible by $P(k, k_1, l_2)$, we have that in r repetitions of the Algorithm 5 at least one repetition finds P admissible with probability $1 - (1 - P(k, k_1, l_2))^r \geq 1 - e^{-P(k,k_1,l_2)r}$. Setting $r = \lceil 1/P(k, k_1, l_2) \rceil$, it follows from Lemma 56 that any fixed k-path starting at s in G is witnessed with probability at least $(1 - e^{-1})/2$ in time $O(2^{k_1+l_2}k^5n^4/P(k, k_1, l_2))$. Setting $k_1 = \gamma_1 k, l_2 = \gamma_2 k$, and approximate $P(k, k_1, l_2)$, we obtain $O^*(1.6569^k)$ time for $\gamma_1 = 0.5$ and $\gamma_2 = 0.207107$. So we get the following theorem

Theorem 57. *There exists a randomized algorithm that given graph $G = (V, E)$ solves k-PATH in $O^*(1.6569^k)$ time. The algorithm cannot give false positives and may give false negatives with constant probability.*

7
Conclusion

We have studied different randomized techniques used in FPT and we have seen that randomized algorithms have better running time compared with that of deterministic ones. The best known algorithm for finding a k-path in a given graph is a randomized algorithm which we described in the last chapter and it takes running time $O^*(1.6569^k)$. The main open problems in this area are–(i) can we derandomize isolation lemma and DeMillo-Lipton-Schwartz-Zippel lemma, and (ii) does there exist a deterministic algorithm for solving k-PATH with running time less than $4^k k^{O(\log^2 k)} n^{O(1)}$.

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. 2004.

[2] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[3] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[4] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.

[5] J. Chen, S. Lu, S.-H. Sze, and F. Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 298–307, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition.* MIT Press, McGraw-Hill Book Company, 2000.

[7] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011.

[8] R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[9] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[10] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[11] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *WG*, pages 58–67, 2006.

[12] I. Koutis. Faster algebraic algorithms for path and packing problems. In *ICALP (1)*, pages 575–586, 2008.

[13] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *STOC*, pages 345–354, 1987.

[14] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *FOCS*, pages 182–191, 1995.

[15] R. Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, March 2006.

[16] N. Robertson and P. D. Seymour. Graph minors. iii. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[17] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM J. Comput.*, 19(5):775–786, 1990.

[18] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, Oct. 1980.

[19] A. Terras. *Fourier Analysis on Finite Groups and Applications*. London Mathematical Society Student Texts. Cambridge University Press, 1999.

[20] R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[21] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. *Combinatorial Optimization - Eureka, You Shrink!, LNCS*, pages 185–207.