CHENNAI MATHEMATICAL INSTITUTE

Paths, Cycles and Permanent: A bridge between Combinatorics and Algebra

Author:
Kishlaya JAISWAL

Supervisor:
Dr. Samir DATTA

A thesis submitted in fulfillment of the requirements for the degree

Master of Science

in

Computer Science

May 27, 2021

Declaration of Authorship

I, Kishlaya JAISWAL, declare that this thesis titled, "Paths, Cycles and Permanent: A bridge between Combinatorics and Algebra" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Kishlaya Jaiswal

Date: May 27, 2021

Abstract

In this report, we survey papers on disjoint paths and cycles, which include Björklund and Husfeldt's algorithm for finding shortest 2-disjoint paths in undirected graphs [BH19] and Wahlström's algorithm for finding a cycle passing through given points [Wah13], both of which solve a combinatorial problem by encoding it into a similar algebraic structure.

[BH19] algorithm requires computing the permanent mod 4 of matrix of integer polynomials which is based on Gaussian Elimination, known to be highly sequential. We present a parallel algorithm for computing permanent mod 2^k of a matrix of univariate integer polynomials. It places the problem in $\oplus\mathsf{L} \subseteq \mathsf{NC}^2$. This extends the techniques of [Val79], [BKR09] and [BH19] and yields a (randomized) parallel algorithm for shortest 2-disjoint paths improving upon the recent (randomized) polynomial time algorithm [BH19]. We also recognize the disjoint paths problem as a special case of finding disjoint cycles, and present (randomized) parallel algorithms for finding a shortest cycle and shortest 2-disjoint cycles passing through any given fixed number of vertices or edges.

Towards the end, we also discuss Schrijver's algorithm of finding disjoint paths in planar directed graphs [Sch94], which addresses a similar problem but using a completely different set of algebraic tools.

Acknowledgements

I will take this opportunity to express my gratitude to my advisor, professor Samir Datta, who recognized a potential in me and directed it towards the right path. Even though, as the COVID crisis prevailed, I was uncertain if I will be able to complete the work, but with his immense patience he always kept me motivated.

I extend my gratitude to professor Partha Mukhopadhyay for his valuable comments on a preliminary version of our work. I also want to thank professor BV Rao, to whom I whole-heartedly owe most of thought development and learning at my time in CMI.

I am also grateful to my friend Deeparaj Bhat for taking time to walk me through some mathematical concepts, which otherwise would have been difficult for me to understand during the course of my thesis.

I would like to end by thanking my parents for their love, care and all the support and my sister for always cheering me up. It would not have been possible to do any of it without them.

To my parents for always supporting me with the tough decisions in life…

Contents

Chapter 1

Introduction

> In the broad light of day
> mathematicians check their equations
> and their proofs, leaving no stone
> unturned in their search for rigour.
> But, at night, under the full moon, they
> dream, they float among the stars and
> wonder at the miracle of the heavens.
> They are inspired. Without dreams
> there is no art, no mathematics, no life.
>
> Michael Atiyah

1.1 Overview

1.1.1 Disjoint Paths

Broadly we are concerned with the problem of finding disjoint paths between specified pairs of vertices in any given graph, and several variants of this problem. For instance, is the graph directed or undirected? Is the graph planar? Can we find shortest such paths? and many other interesting questions.

To begin with, let us first formally describe the problem $DP(k)$:

> **Input**: a graph G and k pairs of vertices $\{(s_i, t_i) \mid i \leq k\}$ on G
>
> **Output**: paths P_i from s_i to t_i such that for any $1 \leq i < j \leq k$, P_i and P_j are disjoint

When k **is not fixed** (and is part of input) then the problem is known to be NP-hard even for undirected planar graphs [Lyn75]. However, linear time algorithms are known when further restricting directed planar graphs to the case:

- when all terminals lie on outer face [SAN90], or
- when all the s_i-terminals lie on one common face while all the t_i-terminals lie on another common face [RWW96]

If we further ask for paths with minimal total length in the latter problem, then [VS11] presented a $O(kn \log n)$ running time algorithm to achieve the same.

When k **is fixed** the problem remains NP-hard for directed graphs, even for $k = 2$ [FHW80], who had also given given a poly time algorithm for the restricted case of directed acyclic graphs. In the restriced setting of directed planar graphs, [Sch94]

presented a $n^{O(k)}$ running time algorithm, which was further improved to a fixed parameter tractable algorithm by [Cyg+13].

Shifting our focus to undirected graphs, the celebrated work of Robertson and Seymour [RS95] gave a $O(n^3)$ algorithm for finding k disjoint paths in an undirected graph, for any fixed k. [Dat+18] gave a parallel algorithm for class of planar graphs where all the terminals lie either on one or two faces. All this while, the question of finding *shortest* disjoint paths in general undirected graphs, remained open for many years until recently, Björklund and Husfeldt [BH19] gave a (randomized) polynomial time algorithm for finding the shortest 2-disjoint paths. For general k, this problem still remains open. Björklund and Husfeldt also gave a parallel algorithm to count shortest 2-disjoint paths but only for cubic planar graphs [BH18].

1.1.2 Permanent

Given a $n \times n$ matrix $A = (a_{ij})_{i,j\in[n]}$, determinant and permanent of A are defined as

$$\det(A) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma(i)} \qquad \operatorname{perm}(A) = \sum_{\sigma\in S_n} \prod_{i=1}^{n} a_{i\sigma(i)}$$

The problem of computing the determinant of a matrix has been a very well studied problem in the past, and several fast (both sequential and parallel) algorithms are known. On the contrary, the problem of computing permanent, an algebraic analogue of determinant, of an integer matrix was first shown to be NP-hard by Valiant [Val79], where he also presented a $O(n^{4k-3})$ running time algorithm to compute permanent modulo 2^k. It was also shown that computing permanent modulo any odd prime still remains hard. Zanko [ZAN91] gave a proof for hardness of permanent under many-one reductions strengthening the result from the weaker Turing reductions used by Valiant.

Valiant's algorithm for permanent mod 2^k uses Gaussian elimination which is known to be highly sequential and so it is desirable to have a parallel algorithm. This was resolved by Braverman, Kulkarni and Roy [BKR09] who presented a $\oplus\mathsf{SPACE}(k^2 \log n) \subseteq \mathsf{NC}^2$ algorithm.

Moreover, NC algorithms for computing determinant of matrices over arbitrary commutative rings are also known, e.g. [MV97]. We would like to ask a similar question for the permanent. One natural extension would be to consider the ring of polynomials with integer coefficients. Björklund and Husfeldt [BH19] gave a $d^3 n^{O(k)}$ time algorithm to compute permanent modulo 2^k of matrices over integer polynomials where the entries are of degree atmost d and their algorithm uses Gaussian elimination.

1.1.3 The Bridge

The permanent of a matrix can be regarded as the weighted sum of cycle covers of a directed graph. This gives a combinatorial interpretation to a seemingly pure algebraic quantity. Let us formalize this notion now.

Let G be a weighted directed graph (not necessarily loopless) with the associated weight function w. Our first definition is about cycle covers in a graph. As the name suggests, a cycle cover is a collection of cycles in the given graph such that every vertex appears in some cycle. Formally,

Definition 1.1.1. *We say $C \subseteq E(G)$ is a cycle cover of G if*

- *C is a union of vertex-disjoint simple directed cycles in G*
- *every vertex of G is incident to some edge in C*

Note. Loops are allowed as simple cycles in the above definition.

Now we extend the weight function to define the weight of cycle cover C as follows: $\tilde{w}(C) = \prod_{e \in C} w(e)$. Finally, we encode our given graph into a matrix.

Definition 1.1.2. *Let $V(G) = \{v_1, \ldots, v_n\}$ then we say $A_G = (a_{ij})_{i,j \in [n]}$ is the $(n \times n)$ adjacency matrix of G if*

$$a_{ij} = \begin{cases} w(e) & \text{if } e = (v_i, v_j) \in E(G) \\ 0 & \text{otherwise} \end{cases}$$

With this setup we can finally bridge the cycle covers and permanent as follows:

Lemma 1.1.3. *Let (G, w) be a weighted directed graph and A_G be its adjacency matrix. Then we have* $\operatorname{perm}(A_G) = \sum \tilde{w}(C)$ *where the sum is taken over all cycle covers C of G*

Proof. The crucial observation is that every cycle cover C can be identified with a permutation $\sigma \in S_n$ and in that case, $\tilde{w}(C) = \prod_{i=1}^{n} a_{i\sigma(i)}$.

To see this, we write C as the union of vertex-disjoint directed cycles $c_1 \cup c_2 \cup \ldots \cup c_k$, then $\sigma = \prod_i \sigma_i$ where σ_i is the cyclic permutation corresponding to the directed cycle c_i. That is, given a cycle $c' = (v_{j_1} \to v_{j_2} \to \cdots \to v_{j_k})$, the corresponding cyclic permutation is $\sigma' = (j_1, j_2, \ldots, j_k)$ and with slight abuse of notation we have that $\tilde{w}(c') = a_{j_1 j_2} a_{j_2 j_3} \ldots a_{j_{k-1} j_k}$.

Furthermore, if $\sigma \in S_n$ is such that it does not correspond to any cycle cover then we claim that $\prod_i a_{i\sigma(i)} = 0$. Consider the following subset $S \subseteq V(G) \times V(G)$ defined as $S = \{(v_i, v_{\sigma(i)}) \mid i \in [n]\}$. If $S \subseteq E(G)$, then clearly S is a cycle cover with the corresponding permutation σ, which is a contradiction. Therefore, there exists an i such that $(v_i, v_{\sigma(i)}) \notin E(G) \implies a_{i\sigma(i)} = 0$.

Hence we get

$$\operatorname{perm}(A_G) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)} = \sum_{C} \tilde{w}(C)$$

□

Now when restricted to weighted undirected graphs (G, w), we view them as a directed graph by replacing each undirected edge $\{u, v\}$ with two directed edges (u, v) and (v, u) and we assign *symmetrical* weights to these edges, that is $w((u, v)) = w((v, u)) = w(\{u, v\})$. A direct application of this is seen in [BH19]. Their algorithm requires computing permanent mod 4 of adjacency matrix of *pattern graphs*, to obtain shortest 2-disjoint paths.

1.2 Complexity Classes

We define some of the main complexity classes we will be using in our work. To begin with, we have the class NC which encaptures the notion of parallel computation.

Definition 1.2.1. NC^i *is the class of decision problems solvable in time $O(log^i n)$ on a parallel computer with a polynomial number of processors, or the class of decision problems decidable by uniform boolean circuits with a polynomial number of gates of fan-in 2 and depth $O(log^i n)$.*

Clearly, $\mathsf{NC}^i \subseteq \mathsf{NC}^{i+1}$, for all $i \geq 1$ and NC ("Nick's Class") is defined as $\mathsf{NC} = \bigcup \mathsf{NC}^i$

Definition 1.2.2. $\oplus\mathsf{L}$ *is the class of decision problems solvable by an* NL *machine such that*

- *If the answer is 'yes', then the number of accepting paths is odd.*
- *If the answer is 'no', then the number of accepting paths is even.*

Remark. Matrix powering over $\mathbb{Z}_2$ is complete for $\oplus\mathsf{L}$

Definition 1.2.3. TC^0 *contains all languages which are decided by Boolean circuits with constant depth and polynomial size, containing only unbounded fan-in AND gates, OR gates, NOT gates, and majority gates.*

Remark. TC^0 contains several important problems, such as integer division.

Lemma 1.2.4. *We have the following well-known subset relations*

- $\oplus\mathsf{L} \subseteq \mathsf{NC}^2$
- $\mathsf{NL} \subseteq \mathsf{NC}^2$
- $\mathsf{TC}^0 \subseteq NC^1$

1.3 Our Contribution

To compute permanent of a matrix A over integer polynomials, we closely follow the analysis of [BKR09] but immediately hit an obstacle. They give a reduction from $\text{perm}(A) \pmod 4$ to several computations of $\text{perm}(.) \pmod 2$, which crucially uses the fact that $\mathbb{Z}_2$ is a field. More precisely, when mimicking the proof, firstly it is required to find a non-trivial solution of $Av = 0$ with the property that atleast one of the entries of this vector is invertible. This fails[1] over $\mathbb{Z}_2[x]$. Moreover, their algorithm also uses the fact that a non-singular matrix admits a LU decomposition iff all the leading principal minors are non-zero, which is known to hold in general only for matrices over fields.

Therefore, replacing $\mathbb{Z}$ with $\mathbb{Z}[x]$ in their analysis doesn't work as $\mathbb{Z}_2[x]$ isn't a field. Furthermore, any finite field $\mathbb{F}$ of characteristic 2 only corresponds to modulo 2 arithmetic. We need a way to extend the field structure so that it supports modulo 2^k arithmetic as well. If $\mathbb{F}$ was realized as $\mathbb{Z}_2[x]/(p(x))$ where $p(x)$ is irreducible over $\mathbb{Z}_2$ then a possible candidate is the ring $\mathbb{Z}[x]/(2^k, p(x))$. Therefore, the appropriate algebraic structure to consider would be the ring $\mathfrak{R} = \mathbb{Z}[x]/(p(x))$

Now we see that replacing $\mathbb{Z}$ with $\mathfrak{R}$ solves the above mentioned problems in the analysis, primarily because of the fact that $\mathfrak{R} \pmod 2$ is a finite char 2 field. With a slight bit of modification in the proof, we achieve that: given a matrix A over $\mathfrak{R}$, we can find $\text{perm}(A) \pmod{2^k}$ or in other words if A is a matrix over $\mathbb{Z}[x]$, we can compute $\text{perm}(A) \pmod{2^k, p(x)}$.

We are still not done because our aim was to compute $\text{perm}(A) \pmod{2^k}$ over $\mathbb{Z}[x]$. To achieve that, we choose $p(x)$ such that its degree is larger than the degree of polynomial $\text{perm}(A)$. This requires doing computations over a large field. Alternatively, we develop a new way of interpolation over $\mathfrak{R}$, which allows us to

[1] Let $A = \begin{pmatrix} x & x+1 \\ x & x+1 \end{pmatrix}$ then there does not exist any null vector of the form $\begin{pmatrix} f \\ 1 \end{pmatrix}$ or $\begin{pmatrix} 1 \\ f \end{pmatrix}$ for any $f \in \mathbb{Z}_2[x]$

choose $p(x)$ such that its degree is of logarithmic order of degree $\text{perm}(A)$, but with a tradeoff of computing several (polynomially many) more permanents. We present this technique for its novelty. Therefore, we achieve the following

Theorem 1.3.1. *Let $k \geq 1$ be fixed and A be a $n \times n$ matrix of integer polynomials, such that the degree of each entry is atmost $\mathsf{poly}(n)$. We can compute $\text{perm}(A) \pmod{2^k}$ in $\oplus\mathsf{L} \subseteq \mathsf{NC}^2$*

Wahlström [Wah13] addressed the question of finding a cycle passing through given vertices. We ask if we can also find shortest such cycle. And furthermore, can we also find shortest 2-disjoint cycles passing through these vertices? We combine techniques of [Wah13] and [BH19] to answer the above questions, by reducing them to computing permanents modulo 2 of 2^{k-1} and modulo 4 of $2^{k-1} + 2^{k-2}$ matrices respectively. These matrices are adjacency matrix of what we refer to as *pattern* graphs. Notice that for $k = 2$ finding shortest 2-disjoint cycles corresponds to finding shortest 2-disjoint paths (by connecting each pair of terminals with a common vertex), and in this case our pattern graphs are exactly those presented in [BH19].

Theorem 1.3.2. *Let $k \geq 1$ be fixed and G be an undirected graph with k marked vertices. We can find shortest 2-disjoint cycles passing through the marked vertices in $\oplus\mathsf{L}/\mathsf{poly}$ (and RNC)*

1.4 Organization

In Chapter 1, we first introduce the topic, preliminaries and some notation that we shall be using throughout this report. Chapter 2 discusses Björklund and Husfeldt's algorithm for finding shortest 2 disjoint paths in undirected graphs [BH19]. Building up on this and Wahlström's algorithm for finding a cycle passing through given points [Wah13], we present an algorithm for our theorem 1.3.2, in Chapter 3.

In Chapter 4 we present proof of our main theorem 1.3.1 about computing permanent modulo 2^k, following which we also discuss the complexity of required computations over the ring $\mathfrak{R}$ which shows that our algorithm is in $\oplus\mathsf{L}$. We also present an alternative proof for our main theorem in section 4.2 which uses new techniques. Chapter 5 discusses how to apply the same techniques to hafnian and hence it gives an alternate proof of the already known result that counting perfect matchings modulo 2^k is in P.

Finally in the last Chapter 6, we discuss Schrijver's algorithm of finding disjoint paths in planar directed graphs [Sch94], with which we close our discussion in Chapter 7 with a few ending remarks.

Chapter 2

Shortest Disjoint Paths

> Just as music comes alive in the performance of it, the same is true of mathematics. The symbols on the page have no more to do with mathematics than the notes on a page of music. They simply represent the experience.
>
> Keith Devlin

> There is geometry in the humming of the strings, there is music in the spacing of the spheres
>
> Pythagoras

2.1 Shortest?

Coming back to disjoint paths problem on undirected graphs, we want to add the constraint that we require our disjoint paths to be *shortest*.

But first we need to identify the metric in regard to which we are referring to these paths as shortest. Here is one such metric: paths P_i are such that $\sum_i |P_i|$ is minimum. $|P_i|$ denotes the length of the path if the graph is unweighted otherwise it is the sum of weights of the edges occuring on P_i.

Another interesting metric to consider would be: paths P_i are such that each P_i is individually the shortest path from s_i to t_i. Notice this forms a subcase of the above because we could use Djikstra's algorithm to compute the individually shortest distances and then check if the shortest disjoint paths (with regard to first metric) is equal to the sum of these lengths or not.

Other metrics, interesting in their own right, could also be considered but we shall work with the first one described above. So let us formally define the problem $SDP(k)$ as follows:

> **Input**: a weighted undirected graph (G, w) and k pairs of marked vertices $\{(s_i, t_i) \mid i \leq k\}$ on G
>
> **Output**: paths $\mathcal{P}_i$ from s_i to t_i such that for any $1 \leq i < j \leq k$, $\mathcal{P}_i$ and $\mathcal{P}_j$ are disjoint and $\sum_i w(\mathcal{P}_i)$ is minimal

Here $w : E(G) \to \mathbb{R}_+$ is the given weight function on the edges of G and $w(P_i) = \sum_{e \in P_i} w(e)$

2.2 Pre-processing

Given a graph $G = (V, E, w)$ and k pairs of marked vertices $\{(s_i, t_i)\}_{i \leq k}$, assign weight $x^{w(e)}$ to the edge e of G and add self loops (weight 1) on all vertices except $\{s_i, t_i\}_{i \leq k}$. Since the non-zero terms appearing in the permanent of adjacency matrix correspond to a cycle cover in G, we add directed edges e_i from s_i to t_i and delete all other outgoing eges from s_i, for each i. This will force the cycle cover to use these edges e_i and now the *good* cycle covers will determine the required disjoint paths. To only obtain the *good* cycle covers, we shall use pair up the terminals differently, which we refer to as *patterns*.

Formally, define a pattern P as an ordered pairing of terminals $\{s_i, t_i \mid 1 \leq i \leq k\}$. Furthermore, we view each undirected edge $\{u, v\}$ in G as two directed edges (u, v) and (v, u) with the same weight. For any pattern P, define a pattern graph G_P with the same vertex/edge set as of G but such that if $(u, v) \in P$ then all outgoing edges from u, except edge (u, v), are deleted. We denote by A_P the adjacency matrix of G_P.

2.3 Shortest 2-Disjoint Paths

Consider the following patterns, as also depicted in Figure 2.1:

- $P_0 = \{(s_1, t_1), (s_2, t_2)\}$
- $P_1 = \{(s_1, t_1), (t_2, s_2)\}$
- $P_2 = \{(s_1, s_2), (t_1, t_2)\}$

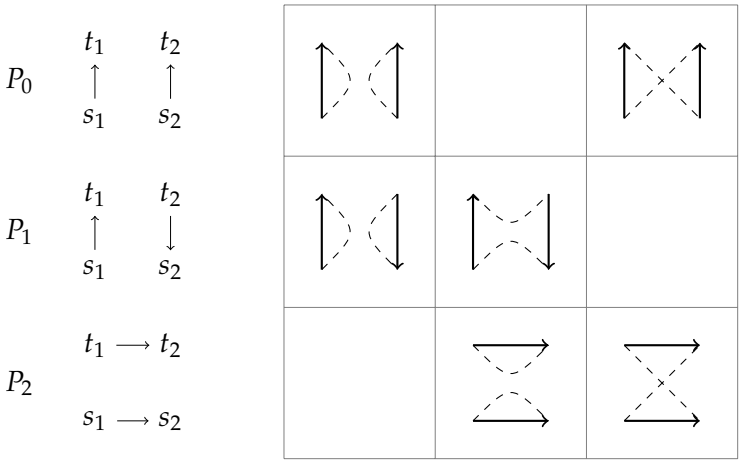


FIGURE 2.1: Patterns and Cycle Covers

Then the following combination of permanents

$$f(x) = \text{perm}(A_{P_0}) + \text{perm}(A_{P_1}) - \text{perm}(A_{P_2})$$

gives us the disjoint paths. Ideally, we would like to express $f(x)$ as twice the sum of all solutions but some problems remain.

First of all, computing permanent is hard, so we still need to do some work. Notice that since each cycle cover corresponding to any term in $f(x)$ appears in both graphs corresponding to patterns P_0 and P_1 with the same weight, so $f(x) \equiv 0 \pmod 2$. Now the hope is that $f(x) \pmod 4$ gives us the desired result. But

observe that if for each solution S there was another solution S' with the same weight then again $f(x) \equiv 0 \pmod 4$

At this point, we recall that our aim was to only find the shortest disjoint paths and not *all* disjoint paths. So we only need to look at the exponent of smallest non-zero term in $f(x) \pmod 4$. Therefore, under the assumption that the shortest 2-disjoint paths were unique, we get that exponent of non-zero coefficient term in $f(x) \pmod 4$ gives us the weight of shortest 2-disjoint paths!

To verify our above claim, our main observation is that the two cycles corresponding to the solution can only be traversed in one direction because the s_i, t_i edge is directed. So we check that if there are extraneous cycles in our cycle cover alongwith the solution then

- If the extraneous cycle is of length atleast 3, then we can traverse this cycle either in clockwise or counter-clockwise direction, corresponding to two different permutations σ and σ^{-1} and because such a cycle cover would occur in two of the pattern graphs, we get that such cycle covers appear with a coefficient of atleast 4 and hence do not appear in $f(x) \pmod 4$.
- So we only need to consider the extraneous 2-cycles (matchings) which do not contribute an additional factor of 2 and hence remain in our solution. Notice that for each matching edge, a weight of $x^{w(e)}$ is added to our cycle cover C. Therefore, consider the cycle cover C' which is same as C except each matching edge is replaced with two self-loops on the vertices incident to matching edges. This cycle cover corresponds to a solution and weight of C' is strictly smaller than the weight of C.

That is, we can express $f(x) \pmod 4 = 2\sum_{S,M} x^{w(S)+w(M)}$ where the sum is taken over all solutions S for the 2-disjoint paths problem and matchings M on any subset of vertices not covered by the solution S.

Let S_0 be the unique-weight minimum solution, then the smallest non-zero term in $f(x) \pmod 4$ is $2x^{w(S_0)}$, so we can read off the weight of our required solution from smallest non-zero term in $f(x) \pmod 4$.

Further notice that we have only obtained the weights of shortest solution. In the next chapter, we shall see how to recover the paths itself from these weights. We shall also see how to drop the assumption of unique minimum weight solution by using isolation lemma.

Chapter 3

Shortest Disjoint Cycles

Rose smells the same regardless of
what you call it
...but in mathematics, a different name
might suggest some new ideas

BV Rao

Having understood the solution for $SDP(2)$, naturally one would ask if there is a linear combination of some patterns which would give us $SDP(3)$.

It is not clear that if such a scheme exists and the same techniques can be applied for $SDP(k)$. But let us again look at $SDP(2)$ from a different perspective. If we fix the edges (s_1, t_1) and (s_2, t_2) then we need to find shortest 2 disjoint cycles passing through these 2 edges. To generalize this idea, we recall SDP and also introduce our new SDC problem.

$SDP(k)$: Given a weighted undirected graph with k pairs of marked vertices $\{(s_i, t_i) \mid 1 \leq i \leq k\}$, find the minimum of sum of weight of paths between each pair s_i and t_i such that all paths are pairwise disjoint.

$SDC(l, k)$: Given a weighted undirected graph with k marked vertices, find the minimum of sum of weight of l cycles such that they pass through all the marked vertices and are pairwise disjoint and each cycle is incident to atleast one of the marked vertices.

Note. We only consider *non-trivial cycles* that is we don't consider self-loops or matching in the above SDC problem.

Given an instance of the $SDP(2)$ problem, join the pairs of vertices (s_1, t_1) and (s_2, t_2) with new vertices u_1 and u_2 respectively, as show in Figure 3.1. Notice that any two disjoint cycles passing through u_1 and u_2 give us two disjoint paths between (s_1, t_1) and (s_2, t_2).

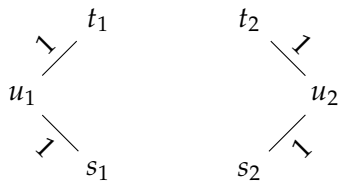


FIGURE 3.1: Converting an instance of $SDP(2)$ to $SDC(2, 2)$

Similarly, connecting the k-pairs of vertices via another new vertex and edges of weight $x^0 = 1$, gives us a reduction from k-disjoint paths to k-disjoint cycles via k-vertices. Since this reduction preserves the weight of the path/cycle, it is indeed a reduction from $SDP(k)$ to $SDC(k, k)$.

To apply the techniques of [BH19] to disjoint cycles problem, we instead consider the following variant $SDCE(l,k)$: Given a weighted undirected graph with k marked *edges*, find the minimum of sum of weight of l cycles such that they pass through all the marked edges and are pairwise disjoint.

It can be easily seen that there is a logspace reduction from $SDC(l,k)$ to $SDCE(l,k)$ as follows: Let $(G, \{v_1, v_2, \ldots, v_k\})$ be an instance of $SDC(l,k)$. Assume without loss of generality that the marked vertices form an independent set, or otherwise split the edge into two by introducing a new vertex in the middle. For each i, choose a vertex u_i, a neighbour of v_i, such that $i \neq j \implies u_i \neq u_j$, we solve $(G, \{e_1, e_2, \ldots e_k\})$ an instance of $SDCE(l,k)$ where $e_i = \{u_i, v_i\}$ and output the smallest solution amongst all the instances of $SDCE$ thus created. Since for each i, $\deg(v_i) < n$, number of instances of $SDCE$ created are bounded by $O(n^k)$ all of which can be solved in parallel as k is fixed.

3.1 Pre-processing

The pre-processing step is similar to as in the previous case of 2-disjoint paths. For the sake of clarity we mention it once again.

Given a graph $G = (V, E, w)$ and k marked edges $\{e_i = \{s_i, t_i\}\}_{i \leq k}$, assign weight $x^{w(e)}$ to the edge e of G and add self loops (weight 1) on all vertices except $\{s_i, t_i\}_{i \leq k}$. Observe that all the non-zero terms appearing in the permanent of adjacency matrix correspond to a cycle cover in G. To force these k-edges in our cycle cover, we direct these edges in a certain way which we shall call as a *pattern*.

Formally, define a pattern P as an ordered pairing of terminals of given edges $\{s_i, t_i \mid 1 \leq i \leq k\}$. Furthermore, we view each undirected edge $\{u, v\}$ in G as two directed edges (u, v) and (v, u) with the same weight. For any pattern P, define a pattern graph G_P with the same vertex/edge set as of G but such that if $(u, v) \in P$ then all outgoing edges from u, except edge (u, v), are deleted. We denote by A_P the adjacency matrix of G_P.

Now we shall show how to solve the $SDCE(1,k)$ problem for any $k \geq 1$. This algorithm was already reminiscent in the work of Magnus Wahlström on finding a cycle passing through given points [Wah13]. Since our technique is also almost same as that of him, we attribute our solution to this particular problem to him. Next, we also present how to solve the $SDCE(2,k)$ problem for any $k \geq 2$. As far as we know, no algorithm (better than brute force) was known apriori to our work for $k \geq 3$.

3.2 Shortest Cycle

Magnus Wahlström presented a FPT algorithm to find a cycle passing through given vertices on a graph [Wah13], by calculating determinant of Tutte matrix of 2^k pattern graphs, over a field of characteristic two. With slight modifications to his techniques, we can in fact achieve shortest cycle. And therefore, we shall discuss the modified technique to achieve the stronger result.

Let $\{e_i = \{s_i, t_i\}\}_{i \leq k}$ be given k-edges. For each binary sequence $b = (b_1, b_2, \ldots, b_{k-1})$ of length $k-1$, consider the following pattern P_b:

- $(s_1, t_1) \in P_b$
- $\forall 2 \leq i \leq k$, if $b_{i-1} = 0$ then $(s_i, t_i) \in P_b$ else $(t_i, s_i) \in P_b$

So $\{P_b\}_b$ is the collection of patterns with the orientation of e_1 fixed and all possible orientations of the other edges $\{e_i\}_{i\geq 2}$, as dictated by each binary sequence.

Claim 3.2.1. Under the assumption that the shortest cycle is unique, the smallest exponent with non-zero coefficient in $f_1(x) \pmod 2$ is the weight of unique shortest cycle passing through the given edges, where

$$f_1(x) = \sum_b \text{perm}(A_{P_b})$$

Proof. Let C be any cycle cover which consists of atleast 2 non-trivial cycles. Consider the cycle in C which doesn't contain edge e_1 - there are two ways of orienting this cycle, namely clockwise and counter-clockwise. So this cycle cover contributes to $f_1(x)$ for atleast two such b-sequences and so it vanishes modulo $f_1 \pmod 2$.

Thus the only terms that survive in $f_1 \pmod 2$ are the cycle covers which consist of one cycle passing through all the given edges and self-loops on the remaining vertices, and furthermore number of cycles of this weight must be odd.

Since the shortest weight cycle was unique by our assumption, we get the desired result. □

3.2.1 Ensuring uniqueness via randomness

To drop the assumption that a unique minimum weight solution exists, we instead assign modified weights $2nmw(e) + w'(e)$ where $n = |V(G)|$, $m = |E(G)|$, $w(e)$ is the given weight of edge e and $w'(e) \in \{0, 2, \ldots, 2m-1\}$ is choosen independently and uniformly at random for each edge e. Then isolation lemma [MVV87] tells us that, with probability $\frac{1}{2}$, the minimum weight cycle is unique. Hence if the term x^j survives as the smallest exponent with non-zero coefficient term in $f_1(x) \pmod 2$ then we get the weight of shortest cycle as $\lfloor j/2nm \rfloor$.

3.3 Shortest 2-Disjoint Cycles

We shall first prove the following stronger result:

Theorem 3.3.1. *Given a set of k-edges $\{e_i\}_{i\leq k}$, we can find weight of the shortest 2-disjoint cycles passing through these edges such that e_1 and e_2 appear in different cycles in* $\oplus\mathsf{L}/\mathsf{poly}$ *(and* RNC*)*

First notice that with a mere re-phrasing of the above theorem, we get the following corollary

Corollary 3.3.2. *Given 2 pairs of terminals s_1, t_1 and s_2, t_2 and a set of k edges, we can find weight of the shortest 2-disjoint paths from s_i to t_i, for $i \in \{1, 2\}$ such that the paths also pass through the given k edges in* $\oplus\mathsf{L}/\mathsf{poly}$ *(and* RNC*)*

Let $\{P_b\}_b$ be the patterns as defined above. Furthermore, for each binary sequence $c = (c_1, c_2, \ldots, c_{k-2})$ of length $k-2$, define pattern Q_c as

- $(s_1, s_2) \in Q_c$
- $(t_1, t_2) \in Q_c$
- $\forall 3 \leq i \leq k$, if $c_{i-2} = 0$ then $(s_i, t_i) \in Q_c$ else $(t_i, s_i) \in Q_c$

With a combination of these patterns, we can get our desired cycle covers. We claim that the non-zero terms appearing in

$$f_2(x) = \sum_b \text{perm}(A_{P_b}) - \sum_c \text{perm}(A_{Q_c})$$

correspond to cycle covers in G_{P_b} such that edges e_1 and e_2 appear in different cycles.

To prove our claim, we need to argue that the cycle covers of G_{P_b} in which e_1 and e_2 appear in the same cycle are exactly the cycle covers of G_{Q_c} Let

$$\mathcal{C}_Q = \bigsqcup_c \text{ cycle covers of } G_{Q_c}$$

$$\mathcal{C}_P = \bigsqcup_b \text{ cycle covers of } G_{P_b} \text{ such that edges } e_1 \text{ and } e_2 \text{ appear in the same cycle}$$

where each cycle cover is counted with repetitions in $\mathcal{C}_P$ and $\mathcal{C}_Q$.

Claim 3.3.3. There is a one-one correspondence between $\mathcal{C}_P$ and $\mathcal{C}_Q$.

Proof. We define the mapping $\varphi : \mathcal{C}_P \to \mathcal{C}_Q$ as follows. Given a cycle cover in $\mathcal{C}_P$, remove the edges e_1 and e_2 and add edges (s_1, s_2) and (t_1, t_2), refer to Figure 3.2 below.

To show that this a well-defined mapping and indeed a bijection, we partition $\mathcal{C}_P$ into type 1 and type 2 cycle covers, depending upon the orientation of the edge e_2. Consider the cycle in which e_1 and e_2 appear together. Then if the edge e_2 is oriented as (s_2, t_2) then we call it type 1 cycle cover otherwise we call it a type 2 cycle cover.

Similarly, we partition $\mathcal{C}_Q$ into type 1 and type 2 cycle covers. If the edges $\{s_1, s_2\}$ and $\{t_1, t_2\}$ appear in the same cycle then we call it a type 1 cycle cover otherwise we call it a type 2 cycle cover.

Fix a type 1 cycle cover of $\mathcal{C}_P$. Then it contains a cycle of the form

$$(s_1 \xrightarrow{e_1} t_1 \to P_1 \to s_2 \xrightarrow{e_2} t_2 \to P_2 \to s_1)$$

Applying φ to this cycle cover we get the cycle

$$(s_1 \xrightarrow{e_1} s_2 \to P_1^{\text{reverse}} \to t_1 \xrightarrow{e_2} t_2 \to P_2^{\text{reverse}} \to s_1)$$

and the other cycles remain intact. This constitutes a type 1 cycle cover in $\mathcal{C}_Q$

Similarly, consider a type 2 cycle cover of $\mathcal{C}_P$ with the cycle

$$(s_1 \xrightarrow{e_1} t_1 \to P_1 \to t_2 \xrightarrow{e_2} s_2 \to P_2 \to s_1)$$

Applying φ to this cycle cover we get two cycles

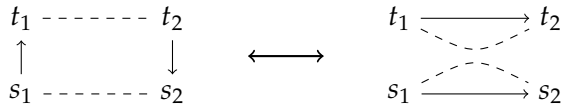
$$(s_1 \xrightarrow{e_1} s_2 \to P_2^{\text{reverse}} \to s_1)$$

$$(t_1 \xrightarrow{e_2} t_2 \to P_1^{\text{reverse}} \to t_1)$$

and the other cycles remain intact. This constitutes a type 2 cycle cover in $\mathcal{C}_Q$.

Therefore, φ is a well-defined mapping and furthermore type i cycle covers of $\mathcal{C}_P$ are mapped to type i cycle covers of $\mathcal{C}_Q$, $i \in \{1, 2\}$. Now consider $\psi : \mathcal{C}_Q \to \mathcal{C}_P$ defined as follows. Given a cycle cover in $\mathcal{C}_Q$, remove the edges (s_1, s_2) and (t_1, t_2) in the cycle and insert edges e_1 and e_2 with the orientation decided by the type. By an similar argument as above, we get that ψ is well-defined and clearly ψ is inverse of φ.

Type 1 cycle cover in $\mathcal{C}_P$ Type 1 cycle cover in $\mathcal{C}_Q$



Type 2 cycle cover in $\mathcal{C}_P$ Type 2 cycle cover in $\mathcal{C}_Q$

FIGURE 3.2: Bijection between $\mathcal{C}_P$ and $\mathcal{C}_Q$

□

Now suppose C is a cycle cover of G such that edges e_1 and e_2 appear in different cycles. We have two cases:

Case 1: number of non-trivial cycles in C is more than 2. Consider any two cycles in C such that e_1 is not incident on them. We can orient these two cycles in both clockwise and anti-clockwise direction and so we get that C is a cycle cover in G_{P_b} for atleast 4 b-sequences. Hence, the term corresponding to C cancels out in $f_2 \pmod 4$

Case 2: number of non-trivial cycles is exactly two. Then C is a cycle cover in G_{P_b} for exactly two b-sequences , that is the the cycle passing through e_2 has two possible orientations whereas cycle passing through e_1 has a fixed orientation (as orientation of e_1 remains fixed in all G_{P_b}). Hence, the term corresponding to C appears with a coefficient of two in $f_2 \pmod 4$. Therefore, the non-zero terms in $f_2 \pmod 4$ correspond to only the cycle covers in which edges e_1 and e_2 appear in different cycles and number of non-trivial cycles is exactly 2. Assuming a unique shortest 2-disjoint cycle exists, it's weight can be obtained from the smallest exponent with a non-zero coefficient in $f_2 \pmod 4$. Finally, to drop this assumption, we again assign random weights as done previously to ensure that the minimum weight solution is unique, with high probability. In the next section we discuss a common weighting scheme to obtain a $\oplus\mathsf{L}/\mathsf{poly}$ algorithm. This shall complete the proof of Theorem 3.3.1.

Corollary 3.3.4. *(Theorem 1.3.2 restated) Given a set of k-edges $\{e_i\}_{i\leq k}$, we can find weight of the shortest 2 cycles passing through these edges in $\oplus\mathsf{L}/\mathsf{poly}$ (and RNC)*

Proof. For each pair of edges e_i and e_j, we can find weight of the shortest cycles separating them using the above algorithm. Hence taking the minimum over all pairs, we get our desired result. □

3.4 Common Weighting Scheme

We have only exhibited a randomized $\oplus\mathsf{L}$ algorithm (that is RNC algorithm). To further show that a common poly weight scheme exists for all graphs of size n, we use the well-known result of [RA00] as follows: Call a weighted undirected graph (G, w) (w is the given weight function on edges) *min-k-unique*, if for any k marked

edges on G, there exists unique shortest l disjoint cycles passing through these k edges. Our goal is to show for each $n > 0$ there exists a set of n^2 weight functions $w_1, \ldots, w_{n^2}$ such that given a graph G on n vertices, (G, w_i) is *min-k-unique* for some $i \in [1, n^2]$.

Given a graph G on n vertices and k marked edges $e_1, \ldots, e_k$, let $\mathcal{F}(e_1, \ldots, e_k)$ be the family of all l disjoint cycles passing through $e_1, \ldots, e_k$. Using isolation lemma [MVV87], if w is a random weight function, that is each edge is assigned a weight from $[1, 4n^{2k+2}]$ independently and uniformly at random, then probability that $\mathcal{F}(e_1, \ldots, e_k)$ has a unique minimum weight element is atleast $1 - 1/4n^{2k}$. Therefore, probability that (G, w) is not *min-k-unique* for a random weight function w is atmost

$$\Pr[\exists e_1, \ldots, e_k : \mathcal{F}(e_1, \ldots, e_k) \text{ doesn't have a minimum weight element}] \leq \sum_{e_1, \ldots, e_k} \frac{1}{4n^{2k}} \leq 1/4$$

Now we claim that there exists a set of n^2 weight functions $W = (w_1, \ldots, w_{n^2})$ such that for any given graph G on n vertices, (G, w_i) is *min-k-unique* for some $1 \leq i \leq n^2$. We say W *is bad* if it doesn't meet this criteria and in particular W *is bad for* G, if none of (G, w_i) is *min-k-unique*. For a randomly choosen W, that is each w_i is chosen independently and uniformly at random, then

$$\Pr[W \text{ is bad for } G] \leq \Pr[\forall i : (G, w_i) \textit{ is not min-k-unique}] \leq \left(\frac{1}{4}\right)^{n^2}$$

$$\implies \Pr[W \text{ is bad}] \leq \Pr[\exists G : W \text{ is bad for } G] \leq 2^{n^2} \left(\frac{1}{4}\right)^{n^2} < 1$$

Hence there exists some $W = (w_1, \ldots, w_{n^2})$ which satisfies the above property and so W is the required poly advice.

To complete the argument for $SDCE(1, k), SDCE(2, k) \in \oplus\mathsf{L}/\mathsf{poly}$, we obtain the weight of shortest cycle(s), by replacing replace w', the random weight function, with each of the weight functions w_i and output the minimum amongst them.

3.5 Constructing Cycles

We remark that under the assumption that the shortest cycle(s) are unique, we can recover these cycles C just from the knowledge of their weight $w(C)$. This follows the standard strategy of solving search via isolation as in [MVV87]. For each edge $e \notin \{e_1, \ldots, e_k\}$, delete the edge e and call the resulting graph G_e. Running our algorithm on $(G_e, \{e_1, \ldots, e_k\})$, if the shortest cycle(s) weight is more than $w(C)$, then discard e otherwise add e to the set C, which gives us the required cycle(s).

Chapter 4

Parallel Polynomial Permanent

> You do not study mathematics because it helps you build a bridge. You study mathematics because it is the poetry of the universe. Its beauty transcends mere things.
>
> Johnathan David Farley

Finally, we embark on our journey of inspecting the bridge from combinatorics to algebra. Let's take a deep breathe and begin.

4.1 Permanent over $\mathfrak{R}$ Mod 2^k

First of all, we fix some general notation. Let $p(x)$ be an irreducible polynomial over $\mathbb{Z}_2[x]$ such that $\deg(p(x))$ is atmost $\text{poly}(n)$. Denote by $\mathbb{F}$ the finite field of char 2, which is realized as $\mathbb{Z}_2[x]/(p(x))$ and by

$$\mathfrak{R}_k = \mathbb{Z}[x]/(2^k, p(x))$$

In particular $\mathfrak{R}_1 \cong \mathbb{F}$. Now as already discussed, we essentially replace $\mathbb{Z}$ by $\mathfrak{R}$ in the algorithm of [BKR09]. We are ready to state the main theorem.

Theorem 4.1.1. *Let $k \geq 1$ be fixed and $A \in \mathfrak{R}^{n\times n}$. We can compute* $\text{perm}(A) \pmod{2^k}$ *in* $\oplus\mathsf{L}$

Proof is by induction on k. We start with the base case $k = 1$. Note that $\text{perm}(A) \equiv \det(A) \pmod 2$. Using corollary 4.1.8 we can find $\text{perm}(A) \pmod 2$ in $\oplus\mathsf{L}$. Now suppose $k > 1$, we shall reduce it to computing several such determinants modulo 2, all of which can be computed in parallel. In doing so, we first illustrate an algorithm which is sequential and then we shall see how to parallelize it.

4.1.1 Sequential algorithm for computing permanent modulo 2^k

We present the algorithm from [BKR09] for computing permanent but translated within our framework. Let $A = (a_{ij})_{i,j\in[n]} \in \mathfrak{R}^{n\times n}$ be such that $\det(A) \equiv 0 \pmod 2$. Therefore we can find a non-zero vector $v \in \mathbb{F}^n$ such that $A^T v = 0$ over $\mathbb{F}$. Assume without loss of generality $v_1 = 1$.

Let r_i denote the i^{th} row of A and define A' to be the matrix where the 1^{st} row in matrix A is replaced with $\sum_i v_i r_i$. Now if we expand the permanent along the first

row then we get

$$\text{perm}(A') = \sum_{i=1}^{n} v_i \text{perm}(A[1 \leftarrow i]) = \text{perm}(A) + \sum_{i=2}^{n} v_i \text{perm}(A[1 \leftarrow i]) \tag{4.1}$$

where $A[i \leftarrow j]$ is the matrix A but with i^{th} row replaced with the j^{th} row. For $I, J \subseteq [n]$ denote by $A[\widehat{I}, \widehat{J}]$ the matrix obtained from A by deleting rows indexed by I and columns indexed by J. With this equation, modulo 2^k computation reduces to modulo 2^{k-1} computations of the minors as follows:

$$\text{perm}(A') = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} v_i a_{ij} \right) \text{perm}(A[\widehat{\{1\}}, \widehat{\{j\}}])$$

Since $A^T v = 0 \pmod 2$, we can write $\sum_i v_i a_{ij} = 2b_j \pmod{2^k}$ for some $b_j \in \mathfrak{R}_k$, therefore, we can re-write the above permanents as:

$$\text{perm}(A') \pmod{2^k} = 2 \left(\sum_{j=1}^{n} b_j \text{perm}(A[\widehat{\{1\}}, \widehat{\{j\}}]) \pmod{2^{k-1}} \right)$$

Similarly, expanding $\text{perm}(A[1 \leftarrow i])$ along the 1^{st} and i^{th} rows, we get the reduction:

$$\text{perm}(A[1 \leftarrow i]) = \sum_{j \neq k} a_{ij} a_{ik} \text{perm}(A[\widehat{\{1,i\}}, \widehat{\{j,k\}}])$$

$$\text{perm}(A[1 \leftarrow i]) \pmod{2^k} = 2 \left(\sum_{j<k} a_{ij} a_{ik} \text{perm}(A[\widehat{\{1,i\}}, \widehat{\{j,k\}}]) \pmod{2^{k-1}} \right)$$

Substituting these equations back in 4.1, we get

$$\begin{aligned} \text{perm}(A) \pmod{2^k} = {} & 2 \left(\sum_{j=1}^{n} b_j \text{perm}(A[\widehat{\{1\}}, \widehat{\{j\}}]) \pmod{2^{k-1}} \right) \\ & - 2 \sum_{i=2}^{n} v_i \left(\sum_{\substack{j,k=1 \\ j<k}}^{n} a_{ij} a_{ik} \text{perm}(A[\widehat{\{1,i\}}, \widehat{\{j,k\}}]) \pmod{2^{k-1}} \right) \end{aligned}$$

Since addition and multiplication over $\mathfrak{R}_k$ is in $\oplus\mathsf{L}$ (see 4.1.13) we get that $\text{perm}(A) \pmod{2^k}$ $\oplus\mathsf{L}$-reduces to $\text{perm}(.) \pmod{2^{k-1}}$. Hence by induction, we can compute $\text{perm}(A) \pmod{2^k}$ in $\oplus\mathsf{L}$, provided that $\text{perm}(A) \equiv 0 \pmod 2$.

Let us see how to drop this assumption. We expand the permanent of A along the i^{th} row, then

$$\text{perm}(A) = \sum_{j} a_{ij} \text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}])$$

If $\text{perm}(A) \not\equiv 0 \pmod 2$, then $\exists i, j$ such that $\text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}]) \not\equiv 0 \pmod 2$. Consider the matrix C where all entries are same as A except the $(i,j)^{th}$ entry which is replaced with $a_{ij} + y$. Then, we get $\text{perm}(C) = \text{perm}(A) + y\text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}])$. Notice that $\text{perm}(A) + y\text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}]) \equiv 0 \pmod 2$ is a linear equation in y over the field $\mathbb{F}$ and so there exists a unique y which satisfies this equation, which

is $y_0 = \text{perm}(A)\text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}])^{-1}$. Setting $y = y_0$ we get $\text{perm}(C) \equiv 0 \pmod 2$, so we can compute $\text{perm}(C) \pmod{2^k}$ and then compute $\text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}])$ recursively as $A[\widehat{\{i\}}, \widehat{\{j\}}]$ is a smaller $(n-1) \times (n-1)$ size matrix. Hence we obtain $\text{perm}(A) = \text{perm}(C) - y_0\text{perm}(A[\widehat{\{i\}}, \widehat{\{j\}}]) \pmod{2^k}$. This yields a sequential algorithm for computing permanent modulo 2^k over $\mathfrak{R}$.

4.1.2 Parallel algorithm for computing permanent modulo 2^k

The bottleneck was finding i, j such that $A[\widehat{\{i\}}, \widehat{\{j\}}]$ is non-singular over $\mathbb{F}$. We fix this by again appealing to the fact that we are working over a field, and modifying A such that all leading principal minors are non-zero. This modification essentially derives from the following fact.

Theorem 4.1.2. *([OJ05] Corollary 1) Let A be an invertible matrix over a field $\mathbb{F}$, then all leading principal minors are non-zero iff A admits an LU decomposition*

Every invertible matrix admits a PLU factorization [OJ05] so let $A = PLU$. Denote by $Q = P^{-1}$, then $QA = LU$. Since Q is also a permutation matrix, we get that $\text{perm}(QA) = \text{perm}(A)$ (because permanent is invariant under row swaps). Therefore, it suffices to give a $\oplus\mathsf{L}$ algorithm to find Q so that we can replace A by QA which is an invertible matrix such that all leading principal minors are non-zero. Thus computing $\text{perm}(A) \pmod{2^k}$ reduces to the problem of computing (in parallel) permanent modulo 2^k of $n-1$ matrices with $\text{perm} \equiv 0 \pmod 2$. This gives a $\oplus\mathsf{L}$ algorithm to compute permanent modulo 2^k over $\mathfrak{R}$.

To find Q, we closely follow [Ebe91]. For each $1 \leq i \leq n$, let A_i be the matrix formed from A by only taking the first i columns. Let A_i^j matrix obtained from A_i by only taking the first j rows. We construct a set $S_i \subseteq [n]$ inductively as follows:

- Base case: $l \in S_i$ if $rank(A_i^l) = 1$ and $rank(A_i^k) = 0$ for all $k < l$
- Include $j \in S_i$ iff $rank(A_i^j) = 1 + rank(A_i^{j-1})$

Since $rank(A_i) = i$, we get $|S_i| = i$. Furthermore note that $S_i \subset S_{i+1}$. So let $S_1 = \{s_1\}$ and for each $i \geq 2$, denote by $s_i \in S_i \setminus S_{i-1}$. Consider the following permutation $Q = (n, s_n) \dots (2, s_2)(1, s_1)$. Thus Q is our desired permutation, such that QA has all leading principal minors non-zero.

As a corollary, we immediately get our desired result.

Corollary 4.1.3. *(Theorem 1.3.1 restated) Given a $n \times n$ matrix $A = (a_{ij})_{i,j\in[n]}$ over $\mathbb{Z}[x]$ with $\deg(a_{ij})$ atmost $\mathsf{poly}(n)$, we can compute $\text{perm}(A) \pmod{2^k}$ in $\oplus\mathsf{L}$ for any fixed $k \geq 1$*

Proof. Let $N = n \max\{\deg(a_{ij})\} + 1$ and choose $l = \lceil \log_3(N/2) \rceil$. Consider $p(x) = x^{2.3^l} + x^{3^l} + 1$ which is irreducible over $\mathbb{Z}_2[x]$ (see [Lin13] Theorem 1.1.28)

Since $\deg(p(x)) \geq N > \deg(\text{perm}(A))$, using this $p(x)$ in above theorem, we get $\text{perm}(A) \pmod{2^k}$ for any fixed k. □

4.1.3 Complexity Analysis

We discuss the complexity results for arithmetic operations over the ring $\mathfrak{R}_k$ and matrix operations over the field $\mathbb{F}$, which were required in our above algorithm. To begin with, we state a well-known fact about integer polynomials matrix multiplication modulo 2. This shall form our basis for showing computations over $\mathbb{F}$ in $\oplus\mathsf{L}$.

Lemma 4.1.4. *(Folklore, [Dam90]) Let $A_1, A_2, \ldots A_n \in \mathbb{Z}_2[x]^{n\times n}$ then the product $A_1A_2\ldots A_n$ can be computed in $\oplus\mathsf{L}$*

To obtain an analogous result over $\mathbb{F}$ we first perform multiplication over $\mathbb{Z}_2[x]$ and then divide all entries by $p(x)$, using the following polynomial division, as demonstrated by Hesse, Allender and Barrington in [HAB02], to get that iterated matrix product over $\mathbb{F}$ is in $\oplus\mathsf{L}$

Lemma 4.1.5. *([HAB02] Corollary 6.5) Given $g(x), p(x) \in \mathbb{Z}[x]$ of degree atmost $\mathsf{poly}(n)$, we can compute $g(x) \pmod{p(x)}$ in $\mathsf{DLOGTIME}-$uniform $\mathsf{TC}^0 \subseteq \mathsf{L}$*

In particular, it follows that given an irreducible polynomial $p(x)$ (over $\mathbb{Z}_2[x]$), then for any $g(x) \in \mathbb{Z}[x]$ of degree atmost $\mathsf{poly}(n)$ we can find $g(x) \pmod{2^k, p(x)}$ in $\oplus\mathsf{L}$, for any fixed $k \geq 1$.

Corollary 4.1.6. *Let $A_1, A_2, \ldots A_n \in \mathbb{F}^{n\times n}$ such that the degree of each entry is atmost $\mathsf{poly}(n)$ then the product $A_1A_2\ldots A_n$ can be computed in $\oplus\mathsf{L}$*

Our algorithm also requires computing inverse of non-zero elements. To compute inverse over $\mathbb{F}^*$ we adopt the techniques from Fich and Tompa [Ebe84; FT88]. Since $\mathbb{F} = \mathbb{Z}_2[x]/(p(x))$ with $N = \deg(p(x))$ which is atmost $\mathsf{poly}(n)$, then given $a \in \mathbb{F}^*$, we observe that $a^{-1} = a^{q-2}$ where $q = 2^N = |\mathbb{F}|$.

We interpret this equation over $\mathbb{Z}_2[x]$, that is we need to compute $a(x)^{q-2} \pmod{p(x)}$ over $\mathbb{Z}_2[x]$. First we show how to compute $a(x)^2 \pmod{p(x)}$. Construct the $N \times N$ matrix Q whose i^{th} row $(Q_{i,0}, Q_{i,1}, \ldots, Q_{i,N-1})$ is defined as:

$$\sum_{j=0}^{N-1} Q_{i,j}x^j = x^{2i} \pmod{p(x)}$$

for each $0 \leq i \leq N-1$. Matrix Q can be computed in $\oplus\mathsf{L}$ using the divison lemma 4.1.5. Then the elements of the row vector $(a_0, a_1, \ldots, a_{N-1})Q$ are the coefficients of $a(x)^2 \pmod{p(x)}$ as explained in section 3 of [FT88]. Furthermore, the coefficients of $a(x)^{2^k} \pmod{p(x)}$ are given by $(a_0, a_1, \ldots, a_{N-1})Q^k$. From lemma 4.1.4 we get that $a(x)^{2^k} \pmod{p(x)}$ can be computed in $\oplus\mathsf{L}$, for any k bounded by $\mathsf{poly}(n)$. Therefore. writing $q-2 = (c_0, c_1, \ldots, c_{N-1})$ in binary,

$$a(x)^{q-2} \pmod{p(x)} = \prod_{i=0}^{N-1} a(x)^{c_i 2^i} \pmod{p(x)}$$

can be computed in $\oplus\mathsf{L}$, which gives us a^{-1}.

Mahajan and Vinay [MV97] describe a way to reduce the computation of a determinant over a commutative ring to a semi-unbounded logarithmic depth circuit with addition and multiplication gates over the ring. In fact, the following is an easy consequence of their result:

Lemma 4.1.7. *(Mahajan-Vinay [MV97]) Let $A \in R^{n\times n}$ be a matrix over a commutative ring. Then there exist $M \in R^{(2n^2)\times(2n^2)}$ and two vectors $a, b \in R^{2n^2}$ such that $det(A) = a^T M^n b$. Moreover, each entry of the matrix M_{ij} and the vectors a, b is one of the entries $A_{i',j'}$ or a constant from $\{0,1\}$ and the mapping ϕ where for every $(i,j) \in [2n^2] \times [2n^2]$, $\phi(i,j) \in A_{[n]\times[n]} \cup \{0,1\}$ is computable in Logspace.*

Proof. (Sketch) In [MV97], given a matrix A they construct a graph H_A whose vertex set is $\{s, t_+, t_-\} \cup Q$ where $Q = \{[p,h,u,i] : p \in \{0,1\}, h,u \in [n], i \in \{0,\ldots,n-1\}\}$. Moreover, the edges are one of the following forms $(s,q), (q,q'), (q,t_+)$ and (q,t_-)

where $q, q' \in Q$ and have weights $w(q, q')$ that each depend on a single entry of A or are one of the constants $0, 1$. Moreover the mapping is very simple to describe. Let us focus on the induced subgraph $H_A[Q]$. Notice that $|Q| = 2n^3$ and each "layer" of $H_A[Q]$ is identical. In other words, $e_i = \langle [p, h, u, i], [p', h', u', i+1] \rangle$ is an edge in $H_A[Q]$ iff $e_j = \langle [p, h, u, j], [p', h', u', j+1] \rangle$ is an edge in $H_A[Q]$ and both have the same weights for every $i, j \in \{0, \ldots, n-1\}$. Thus define the matrix M by putting $M_{[p,h,u],[p',h',u']}$ as the weight of any of the edges e_i.

Finally to define a, b: let $a_{[n \bmod 2,1,1]} = 1$ and $a_q = 0$ for all other q. $b_{[1,h,u]} = a_{uh}$ and $b_{[0,h,u]} = -a_{uh}$. The correctness of our Lemma then follows from the proof of Lemma 2 of [MV97]. □

Using above lemma 4.1.7, we reduce determinant over $\mathbb{F}$ to matrix powering over $\mathbb{F}$, which can be computed in $\oplus\mathsf{L}$ using corollary 4.1.6. Hence we get

Corollary 4.1.8. *Let $A \in \mathbb{F}^{n \times n}$ then $\det(A)$ can be computed in $\oplus\mathsf{L}$*

Now we demonstrate two results: computing rank and a null vector a matrix over $\mathbb{F}$ in $\oplus\mathsf{L}$. We use Mulmuley's algorithm [Mul87], which requires finding determinant over the ring $\mathbb{F}[y, t]$, which reduces to matrix powering over $\mathbb{F}[y, t]$ by the above result. We shall further reduce this to matrix powering over $\mathbb{F}$ as follows: Let R be an arbitrary commutative ring. We associate with each polynomial $f(x) = \sum_{i=0}^{d-1} f_i x^i \in R[x]$ a $d \times d$ lower-triangular matrix

$$P(f) = \begin{bmatrix} f_0 & & & & \\ f_1 & f_0 & & & \\ f_2 & f_1 & f_0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ f_{d-1} & f_{d-2} & f_{d-3} & \cdots & f_0 \end{bmatrix} \in R^{d \times d}$$

Suppose we have two polynomials $f(x)$ and $g(x)$ of degree d_1 and d_2 respectively. We can interpret them as degree $d_1 + d_2$ polynomials (with higher exponent coefficients as 0). Then we have that $P(f+g) = P(f) + P(g)$ and $P(fg) = P(f)P(g)$.

Theorem 4.1.9. *Let R be any commutative ring and $A_1, A_2, \ldots A_n$ be $n \times n$ matrices over $R[x]$ such that the degree of each entry is atmost $\mathsf{poly}(n)$. Denote by $\mathbf{A} = \prod A_l$. There exists $\mathsf{poly}(n) \times \mathsf{poly}(n)$ matrices $B_1, B_2, \ldots B_n$ over R such that the coefficient of x^k in $\mathbf{A}_{ij}$ is equal to $\mathbf{B}_{\psi(i,j,k)}$ where $\mathbf{B} = \prod B_l$ and ψ is logspace computable.*

In other words, iterated matrix multiplication over $R[x]$ is logspace reducible to iterated matrix multiplication over R.

Proof. Let $N = n \max_{i,j,k \in [n]} \{\deg((A_i)_{jk})\}$ where $(A_i)_{jk}$ denotes the $(j,k)^{\text{th}}$ entry of A_i. By our assumption, N is atmost $\mathsf{poly}(n)$. Now for each $1 \leq i \leq n$, compute the matrix $B_i \in R^{nN \times nN}$ obtained from A_i by replacing each polynomial $(A_i)_{jk}$ with the $N \times N$ matrix $P((A_i)_{jk})$. These matrices B_i can be computed in log space. Now the coefficient of x^k in $\mathbf{A}_{ij}$ can be read from the entry $\mathbf{B}_{\psi(i,j,k)}$ where $\psi(i,j,k) = ((i-1)N + k + 1, (j-1)N + 1)$ is logspace computable. The correctness follows from our observation $P(fg) = P(f)P(g)$. □

Remark. This gives us an alternate proof of the fact that iterated matrix multiplication over $\mathbb{Z}_2[x]$ is in $\oplus\mathsf{L}$, as it follows immediately from the definition of $\oplus\mathsf{L}$ that iterated matrix multiplication over $\mathbb{Z}_2$ is in $\oplus\mathsf{L}$.

Lemma 4.1.10. *[Mul87] Let $A \in \mathbb{F}^{m \times n}$ then rank(A) can be computed in $\oplus\mathsf{L}$*

Proof. We can assume that A is a square $(n \times n)$ symmetric matrix because otherwise replace A with $\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$ which has rank twice that of A. Let Y be a $n \times n$ diagonal matrix with the $(i,i)^{th}$ entry as y^{i-1}. And let m be the smallest number such that t^m has a non-zero coefficient in the characteristic polynomial of YA, that is $\det(tI - YA)$. Then rank of $A = n - m$.

Suffices to show that $\det(tI - YA)$ can be computed in $\oplus\mathsf{L}$. Notice that $(tI - YA) \in \mathbb{F}[y,t]^{n\times n}$ and so $\det(tI - YA)$ is logspace reducible to matrix powering over $\mathbb{F}[y,t]$. Using the canonical isomorphism $\mathbb{F}[y,t] \cong \mathbb{F}[y][t]$, repeated application of theorem 4.1.9 logspace reduces it to matrix powering over $\mathbb{F}$. □

Observation 4.1.11. *Let $A \in \mathbb{F}^{n\times n}$ be an invertible matrix then A^{-1} can be computed in $\oplus\mathsf{L}$*

This follows from the fact that computing A^{-1} involves computing the determinant of A and n^2 cofactors, that is determinants of n^2 matrices of size $(n-1) \times (n-1)$. Notice that this also requires inverting the determinant, an element of $\mathbb{F}^*$, which has been explained above.

Corollary 4.1.12. *Let $A \in \mathbb{F}^{n\times n}$ then finding a non-trivial null vector (if it exists) is in $\oplus\mathsf{L}$*

Proof. Let $rank(A) = m$, then permute the rows and columns of A so that we can express $A = \begin{pmatrix} B & C \\ D & E \end{pmatrix}$ such that B is an invertible $m \times m$ matrix. Let $\begin{pmatrix} x \\ y \end{pmatrix}$ be a column vector where $x \in \mathbb{F}^m$ and $y \in \mathbb{F}^{n-m}$, such that

$$A\begin{pmatrix} x \\ y \end{pmatrix} = 0 \implies \begin{pmatrix} B & C \\ D & E \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = 0$$

This reduces to the set of equations: $Bx + Cy = 0$ and $Dx + Ey = 0$. But the later is a redundant set of equations because $\begin{pmatrix} D & E \end{pmatrix}$ can be written in terms of $\begin{pmatrix} B & C \end{pmatrix}$. More precisely, there exists a matrix $V \in \mathbb{F}^{n-m\times m}$ such that $D = VB$ and $E = VC$ and so $Dx + Ey = VBx + VCy = V(Bx + Cy) = 0$. Therefore setting $x = \mathbf{1}$ and $y = -B^{-1}C\mathbf{1}$, gives us the desired null vector. So it suffices to give a $\oplus\mathsf{L}$ algorithm to transform A to the form as specified above, which follows from [Ebe91]. Let A_i be the matrix formed from first i rows of A. We construct a set $S \subseteq [n]$ as follows:

- Base case: $i \in S$ if $rank(A_i) = 1$ and $rank(A_j) = 0$ for all $j < i$
- Include $k \in S$ iff $rank(A_k) = 1 + rank(A_{k-1})$

It follows that $|S| = m$ and let $S = \{i_1 < i_2 < \cdots < i_m\}$ and P_r be the permutation matrix described by $(m, i_m) \ldots (2, i_2)(1, i_1)$. Then P_rA is the required matrix having first m rows as linearly independent. Next, consider the matrix $A' = (P_rA)^T$ and apply the above algorithm to get a permutation matrix P_c such that first m rows of P_cA' are linearly independent. Then $P_rAP_c^T$ is the required matrix such that the leading principal $m-$minor is non-singular. □

Finally, to conclude our result, we discuss arithmetic over $\mathfrak{R}_k$

Lemma 4.1.13. *Let $k \geq 1$ be fixed then the following operations can be done in $\oplus\mathsf{L}$*

- *Multiplication : Given $a, b \in \mathfrak{R}_k$ compute ab*
- *Iterated Addition: Given $c_1, c_2, \ldots, c_n \in \mathfrak{R}_k$ compute $\sum_i c_i$*

Proof. We use the fact that the arithmetic operations mentioned in the statement of lemma, but over $\mathbb{Z}_{2^k}$ are in $\oplus\mathsf{L}$ (see for e.g. [HAB02])

- Let $a(x), b(x) \in \mathfrak{R}_k$ and write $a(x) = \sum_{i=0}^{D} a_i x^i$ and $b(x) = \sum_{i=0}^{D'} b_i x^i$, then $a(x)b(x) = \sum_{i=0}^{D+D'} \left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i$. Finally, using lemma 4.1.5, divide $a(x)b(x)$ by $p(x)$ to obtain $ab \in \mathfrak{R}_k$
- Similarly, let $c_1(x), c_2(x), \ldots, c_n(x) \in \mathfrak{R}_k$ and write $c_i(x) = \sum_{j=0}^{D_i} c_{ij} x^j$ for each $i \in [n]$, then $\sum_{i=1}^{n} c_i(x) = \sum_{j=0}^{\max\{D_i\}} \left(\sum_{i=1}^{n} c_{ij}\right) x^j$ where we assume $c_{ij} = 0$ if $j > D_i$

□

4.1.4 Examples

Let $A = \begin{pmatrix} 1 & x+1 & x+2 \\ x & x^2 & x^2+x \\ x^2 & 3 & x^2+3 \end{pmatrix}$, $p(x) = x^6 + x^3 + 1$ be the irreducible polynomial and we want to evaluate $\text{perm}(A) \pmod 4$ over the ring $\mathfrak{R} = \mathbb{Z}[x]/(p(x))$. First of all, a direct computation gives us $\text{perm}(A) = 2x^5 + 6x^4 + 2x^3 + 12x^2 + 12x$. Now we demonstrate the steps taken by our algorithm.

Step 1: We start by evaluating $\text{perm}(A) \pmod 2$. We directly notice here that last column is the sum of first two columns and so $\det(A) = 0 \implies \text{perm}(A) \equiv 0 \pmod 2$

Step 2: Since $\det(A) \equiv 0 \pmod 2$, we solve the equation $A^T v = 0$ over $\mathbb{F}$ by our method as follows: $\begin{pmatrix} 1 & x & x^2 \\ x+1 & x^2 & 1 \\ x & x^2+x & x^2+1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = 0$

Since rank of the principal 2×2 submatrix is already 2, we set $v_3 = 1$ and solve the equation: $\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = -\begin{pmatrix} 1 & x \\ x+1 & x^2 \end{pmatrix}^{-1} \begin{pmatrix} x^2 \\ 1 \end{pmatrix} 1$ to get $v_1 = x^3 + 1$ and $v_2 = x^5 + x$.

Step 3: For each $j = 1, 2, 3$, we find b_j such that $\sum_i v_i a_{ij} = 2b_j \pmod 4$

$$\begin{aligned} j = 1: &\quad (x^3+1) + x(x^5+x) + x^2 = 2x^2 \\ j = 2: &\quad (x+1)(x^3+1) + x^2(x^5+x) + 3 = 2x^3 \\ j = 3: &\quad (x+2)(x^3+1) + (x^2+x)(x^5+x) + x^2 + 3 = 2x^3 + 2x^2 \end{aligned}$$

Step 4: We have the formula

$$\begin{aligned} \text{perm}(A) \pmod 4 = 2 &\left(\sum_{j=1}^{3} b_j \text{perm}(A[\widehat{\{3\}}, \widehat{\{j\}}]) \pmod 2 \right) \\ &- 2 \sum_{i=1}^{2} v_i \left(\sum_{\substack{j,k=1 \\ j<k}}^{3} a_{ij} a_{ik} \text{perm}(A[\widehat{\{3,i\}}, \widehat{\{j,k\}}]) \pmod 2 \right) \end{aligned}$$

Step 4.1:

$$\text{perm}(A[\widehat{\{3\}}, \widehat{\{1\}}]) = \text{perm}\begin{pmatrix} x+1 & x+2 \\ x^2 & x^2+x \end{pmatrix} = x \pmod 2$$

$$\text{perm}(A[\widehat{\{3\}}, \widehat{\{2\}}]) = \text{perm}\begin{pmatrix} 1 & x+2 \\ x & x^2+x \end{pmatrix} = x \pmod 2$$

$$\text{perm}(A[\widehat{\{3\}}, \widehat{\{3\}}]) = \text{perm}\begin{pmatrix} 1 & x+1 \\ x & x^2 \end{pmatrix} = x \pmod 2$$

$$\implies \sum_{j=1}^{3} b_j \text{perm}(A[\widehat{\{3\}}, \widehat{\{j\}}]) = ((x^3) + (x^4) + (x^4 + x^3)) = 0 \pmod 2$$

Step 4.2:

$$\sum_{\substack{j,k=1 \\ j<k}}^{3} a_{1j}a_{1k}\text{perm}(A[\widehat{\{1,3\}}, \widehat{\{j,k\}}])$$

$$= (x+1)(x^2+x) + (x+2)x^2 + (x+1)(x+2)x = x^3 + x^2 + x \pmod 2$$

$$\sum_{\substack{j,k=1 \\ j<k}}^{3} a_{2j}a_{2k}\text{perm}(A[\widehat{\{2,3\}}, \widehat{\{j,k\}}])$$

$$= x^3(x+2) + x(x^2+x)(x+1) + x^2(x^2+x) = x^4 + x^3 + x^2 \pmod 2$$

$$\sum_{i=1}^{2} v_i \left(\sum_{\substack{j,k=1 \\ j<k}}^{3} a_{ij}a_{ik}\text{perm}(A[\widehat{\{3,i\}}, \widehat{\{j,k\}}]) \pmod 2 \right) = x^5 + x^4 + x^3 \pmod 4$$

Therefore, $\text{perm}(A) \pmod 4 = 2x^5 + 2x^4 + 2x^3$ which matches with our direct computation.

Example 2 Consider $A = \begin{pmatrix} 1 & x & x^2 \\ x & x^2 & 1 \\ 1 & x^2 & x \end{pmatrix}$, and so $\text{perm}(A) = x^5 + x^4 + x^2 + x$. Therefore, we now have $\det(A) \not\equiv 0 \pmod 2$

Step 1: Find Q such that QA has all leading principal minors are non-zero. In this case, we will get $Q = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \implies QA = \begin{pmatrix} 1 & x & x^2 \\ 1 & x^2 & x \\ x & x^2 & 1 \end{pmatrix}$

Step 2: Consider matrix C whose all entries are same as A except the last one which is incremented by y, that is $C = \begin{pmatrix} 1 & x & x^2 \\ 1 & x^2 & x \\ x & x^2 & 1+y \end{pmatrix}$, then $\text{perm}(C) = \text{perm}(A) + y\text{perm}(A[\widehat{\{3\}}, \widehat{\{3\}}])$. Again construct C' same as $A[\widehat{\{3\}}, \widehat{\{3\}}]$ but replace the last entry incremented by y', that is $C' = \begin{pmatrix} 1 & x \\ x & x^2+y' \end{pmatrix} \implies \text{perm}(C') =$

$\text{perm}(A[\widehat{\{3\}}, \widehat{\{3\}}]) + y'\text{perm}(A[\widehat{\{2,3\}}, \widehat{\{2,3\}}])$. Written as one equation, we get

$$\text{perm}(A) = \text{perm}(C) - y\left(\text{perm}(C') - y'a_{11}\right)$$

In this equation, both C, C' are matrices with $\det \equiv 0 \pmod 2$ with the correct choice of y, y', which were:

$$y_0 = \text{perm}(A)\text{perm}(A[\widehat{\{3\}}, \widehat{\{3\}}])^{-1} \pmod 2 = (x^5 + x^4 + x^2 + x)(x^4 + x^3 + x^2) = x^3 + 1$$
$$y_0' = \text{perm}(A[\widehat{\{3\}}, \widehat{\{3\}}])\text{perm}(A[\widehat{\{2,3\}}, \widehat{\{2,3\}}])^{-1} \pmod 2 = x^2 + x$$

So we can compute $\text{perm}(C)$ and $\text{perm}(C')$ by above method and substitute it into previous equation to get $\text{perm}(A) \pmod 4$

4.2 Permanent via Interpolation

We now demonstrate another technique to compute permanent modulo 2^k, which doesn't resort to computations over exponentially sized fields. This proceeds by choosing small degree polynomial $p(x)$. The techniques developed in this section are new and hence interesting by themself.

First we mention a result from [JVW20] used to interpolate the coefficients of a polynomial.

Lemma 4.2.1. *([JVW20] Lemma 3.1) Let $\mathbb{F}$ be a finite, characteristic 2, field of order q.*

$$\sum_{a \in \mathbb{F}^*} a^m = \begin{cases} 0 & \text{if } q-1 \nmid m \\ 1 & \text{otherwise} \end{cases}$$

This dichotomy allows us to extract coefficients of any integer polynomial.

Lemma 4.2.2. *([JVW20] Corollary 3.2) Let $f(x) = \sum_{i=0}^{d} c_i x^i$ be a polynomial with integer coefficients and $q > d+1$, then for any $0 \le t \le d$,*

$$\sum_{a \in \mathbb{F}^*} a^{q-1-t} f(a) = c_t \pmod 2$$

But this gives us the coefficients modulo 2 only. How do we get coefficients modulo 2^k?

The crucial observation here is that the above sum was computed over $\mathbb{F}$. So instead we do so over $\mathfrak{R}$ by identifying a copy of $\mathbb{F}^* \hookrightarrow \mathfrak{R}$, and then we have

$$\sum_{a \in \mathbb{F}^*} a^m = \begin{cases} 2\alpha_m & \text{if } q-1 \nmid m \\ 2\beta_m + 1 & \text{otherwise} \end{cases}$$

where $\alpha_m, \beta_m \in \mathfrak{R}$.

Now we use repeated squaring method to obtain our desired modulo 2^k result

Lemma 4.2.3. $\forall m \ge 0, k \ge 1$

$$\sum_{a_1,\ldots,a_{2^{k-1}} \in \mathbb{F}^*} (a_1 a_2 \cdots a_{2^{k-1}})^m = \left(\sum_{a \in \mathbb{F}^*} a^m\right)^{2^{k-1}} = \begin{cases} 0 \pmod{2^k} & \text{if } q-1 \nmid m \\ 1 \pmod{2^k} & \text{otherwise} \end{cases}$$

Note. We remind the reader that the computation here is done over $\mathfrak{R}_k$

Proof. Fix any $m \geq 0$. Clearly $(2\alpha_m)^{2^{k-1}} \equiv 0 \pmod{2^k}$.

Suffices to show $(2\beta_m + 1)^{2^{k-1}} \equiv 1 \pmod{2^k}$. This follows from induction on k. For $k = 1$ the result holds as stated above. Assume that for some $k \geq 1$, the result holds. Then we have $(2\beta_m + 1)^{2^{k-1}} = 2^k\gamma_{m,k} + 1$ where $\gamma_{m,k} \in \mathfrak{R}$

$$(2\beta_m + 1)^{2^k} = \left((2\beta_m + 1)^{2^{k-1}}\right)^2 = (2^k\gamma_{m,k} + 1)^2 = 1 \pmod{2^{k+1}}$$

□

Using this we can interpolate coefficients of an integer polynomial as follows:

$$\sum_{a_1,\ldots,a_{2^{k-1}} \in \mathbb{F}^*} (a_1 \ldots a_{2^{k-1}})^{q-1-t} f(a_1 \ldots a_{2^{k-1}}) = c_t \pmod{2^k}$$

Finally let $A(x)$ be a $n \times n$ matrix of integer polynomials and the permanent polynomial be

$$\text{perm}(A(x)) = \sum_{i=0}^{N} c_i x^i$$

From the above lemma it follows that

$$\sum_{a_1,a_2\ldots \in \mathbb{F}^*} (a_1 a_2 \cdots)^{q-1-t} \text{perm}(A(a_1 a_2 \cdots)) = c_t \pmod{2^k}$$

provided that our field $\mathbb{F}$ is of order atleast $N + 2$. For this, fix $l = \lceil \frac{\log\log N}{\log 3} \rceil$ so that $2^{2.3^l} > N + 1$. Hence the field obtained from the irreducible polynomial $p(x) = x^{2.3^l} + x^{3^l} + 1$ ([Lin13] Theorem 1.1.28) serves the purpose. It suffices to compute $|\mathbb{F}^*|^{2^{k-1}} = O(N^{2^{k-1}})$ many permanents over $\mathfrak{R}_k$ to obtain all the coefficients c_t modulo 2^k, all of which can be computed in parallel. Hence, we can compute the permanent of A modulo 2^k in $\oplus\mathsf{L}$.

Note. The order of $\mathbb{F}$ in this technique is exponentially smaller than in our previous technique.

Chapter 5

Extensions

> Imagination is more important than knowledge. Knowledge is limited, Imagination encircles the world.
>
> Albert Einstein

Finally, we discuss some other indirect applications of the permanent algorithm. We notice that a similar approach gives us an algorithm to compute Hafnians modulo 2^k of symmetric matrices of integers. Unfortunately, unlike the case of the permanent, we weren't able to extend this to a parallel algorithm. But nevertheless it gives a direct proof of the fact that counting number of perfect matchings modulo 2^k, in any general graph, is in P, as proved in [BKR09].

5.1 Hafnians

Similar to permanent and determinant, another pair of well-studied algebraic analogous functions on a matrix are hafnians and pfaffians. Let $A = (a_{ij})$ be a symmetric $2n \times 2n$ matrix over integers, hafnian is defined as

$$\mathrm{hf}(A) = \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \prod_{j=1}^{n} a_{\sigma(2j-1),\sigma(2j)} \tag{5.1}$$

Note that the diagonal entries of A don't contribute in the calculation of hafnians and hence we can assume them to be 0. Let $B = (b_{ij})$ be a skew-symmetric $2n \times 2n$ matrix, pfaffian is defined as

$$\mathrm{pf}(B) = \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \mathrm{sgn}(\sigma) \prod_{j=1}^{n} b_{\sigma(2j-1),\sigma(2j)} \tag{5.2}$$

But notice that $\mathrm{hf}(A) \equiv \mathrm{pf}(A) \pmod 2$. [MSV04] have shown that $\mathrm{pf}(A)$ can be computed in NC and hence as an immediate consequence we get that $\mathrm{hf}(A) \pmod 2$ can be computed in NC. We can reduce the computation of hafnian to several hafnians of smaller submatrices using the following lemma. Denote by $A[i,j]$ the matrix obtained from A by deleting rows i and j, columns i and j.

Lemma 5.1.1. *([HN18] Lemma 2.2)*

$$\mathrm{hf}(A) = \sum_{j:j \neq i} a_{ij} \mathrm{hf}(A[i,j])$$

$$\mathrm{hf}(A) = a_{ij}\mathrm{hf}(A[i,j]) + \sum_{pq:p,q \notin \{i,j\},p \neq q} (a_{ip}a_{jq} + a_{iq}a_{jp})\mathrm{hf}(A[i,j,p,q])$$

Assume $\text{pf}(A) \equiv 0 \pmod 2$, then $\det(A) \equiv 0 \pmod 2$ and we can find a vector $v \in \mathbb{Z}_2^{2n}$ such that $Av = A^T v = 0 \pmod 2$. Assume without loss of generality $v_1 = 1$.

Let r_i, c_i denote the i^{th} row and i^{th} column of A respectively.

- Construct A' by replacing first row with $\sum v_i r_i$ and then replacing first column with $\sum v_i c_i$
- Construct A_i by replacing first row with r_i and first column with c_i.

Then we check that

$$\begin{aligned}
\text{hf}(A') &= \sum_{j>1} \left(\sum_{i \geq 1} v_i a_{ij} \right) \text{hf}(A[1,j]) \\
&= \sum_{i \geq 1} v_i \left(\sum_{j>1} a_{ij} \text{hf}(A[1,j]) \right) \\
&= \sum_{j>1} a_{1j} \text{hf}(A[1,j]) + \sum_{i>1} v_i \left(\sum_{j>1} a_{ij} \text{hf}(A[1,j]) \right) \\
&= \text{hf}(A) + \sum_{i>1} v_i \text{hf}(A_i)
\end{aligned}$$

Computing $\text{hf}(A')$: since $A^T v = 0 \pmod 2 \implies \sum_{i \geq 1} v_i a_{ij} = 2b_j \pmod{2^k}$ for some $c_j \in \mathbb{Z}$ and hence

$$\begin{aligned}
\text{hf}(A') &= \sum_{j>1} \left(\sum_{i \geq 1} v_i a_{ij} \right) \text{hf}(A[1,j]) \\
&= \sum_{j>1} 2b_j \text{hf}(A[1,j]) \\
\implies \text{hf}(A') \pmod{2^k} &= 2 \left(\sum_{j>1} b_j \text{hf}(A[1,j]) \pmod{2^{k-1}} \right)
\end{aligned}$$

Computing $\text{hf}(A_i)$:

$$\begin{aligned}
\text{hf}(A_i) &= a_{ii} \text{hf}(A[1,i]) + \sum_{pq : p,q \notin \{1,i\}, p \neq q} 2 a_{ip} a_{iq} \text{hf}(A[1,i,p,q]) \\
\implies \text{hf}(A_i) \pmod{2^k} &= 2 \left(\sum_{pq : p,q \notin \{1,i\}, p \neq q} a_{ip} a_{iq} \text{hf}(A[1,i,p,q]) \pmod{2^{k-1}} \right)
\end{aligned}$$

Thus, we can compute $\text{hf}(A) \pmod{2^k}$ provided $\text{pf}(A) \equiv 0 \pmod 2$.

Now if $\text{pf}(A) \not\equiv 0 \pmod 2$, then we can find $(i,j), i \neq j$ such that $\text{hf}(A[i,j]) \not\equiv 0 \pmod 2$. Consider the matrix C where all entries are same as in A except a_{ij} is replaced with $a_{ij} + 1$, then we get $\text{hf}(C) = \text{hf}(A) + \text{hf}(A[i,j])$. Since $\text{hf}(C) \equiv 0 \pmod 2$, we can compute $\text{hf}(C) \pmod{2^k}$ as described above and since $\text{hf}(A[i,j])$ is a $(n-2) \times (n-2)$ matrix, we compute it's hafnian recursively modulo 2^k. Therefore, we can compute $\text{hf}(A) = \text{hf}(C) - \text{hf}(A[i,j]) \pmod{2^k}$.

This gives us a P algorithm for computing hafnians modulo 2^k.

5.2 Counting perfect matchings modulo 2^k

Let G be an undirected graph and A_G denote the adjacency matrix of the graph G. If G has odd number of vertices, then clearly there aren't any perfect matchings. Otherwise it is straight-forward to see that number of perfect matchings in G is same as the value $\mathrm{hf}(A_G)$. Hence the result follows.

Chapter 6

Disjoint Paths on Maps

> Nothing is more practical than a good theory
>
> Vladimir Vapnik

As discussed earlier, the disjoint paths problem is NP hard for directed graphs in general but Schrijver remarkably showed that if restricted to planar graphs, then we can find k-disjoint paths in polynomial time [Sch94] for any fixed k. We discuss this paper informally in this chapter, which is a digression from our main bridge but it is yet another beautiful application demonstrating how sophisticated algebraic tools can help us model combinatorial problems leading to polynomial time algorithms.

We shall refer to planar directed graphs as *maps*. This should also draw the reader's attention to the well-known fact that maps can be drawn on both a flat surface or a sphere. More precisely, any planar graph can be embedded on a $\mathbb{S}^2$ and vice-versa, because sphere with a point removed is homeomorphic to plane. Therefore, from here onwards we shall refer to our underlying surface as a plane and sphere interchangeably as required.

6.1 Ideas and Intuition

Before we move onto graphs, let us abstractly consider paths to develop some intuition for the notion we introduce later on, referred to as *homologous paths*. Fix points $\{s_1, t_1 \mid 1 \leq i \leq k\}$ on a sphere and consider paths P_i from point s_i to point t_i as thread which can be stretched or squeezed as required.

Let $\mathcal{P} = \{P_i\}$ and $\mathcal{Q} = \{Q_i\}$ be two set of paths from s_i to t_i. We call $\mathcal{P}$ homologous to $\mathcal{Q}$ if there is a way to transform $\mathcal{P}$ to $\mathcal{Q}$ without breaking/gluing the threads. For example: figure 6.1 shows homologous paths whereas figure 6.2 shows non-homologous paths.

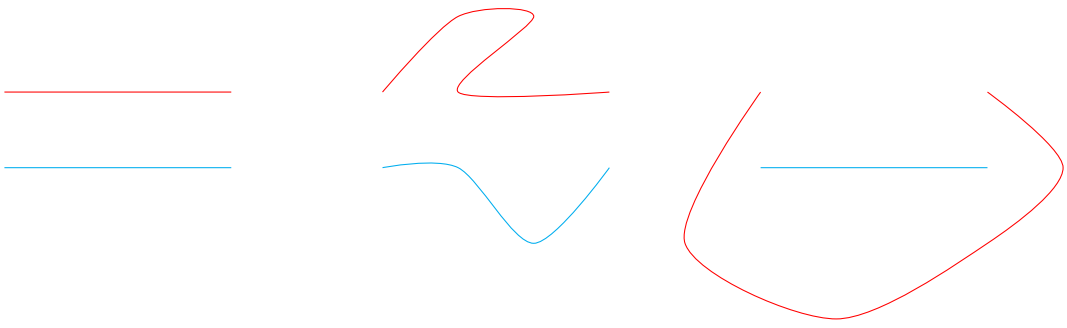


FIGURE 6.1: Homologous Paths

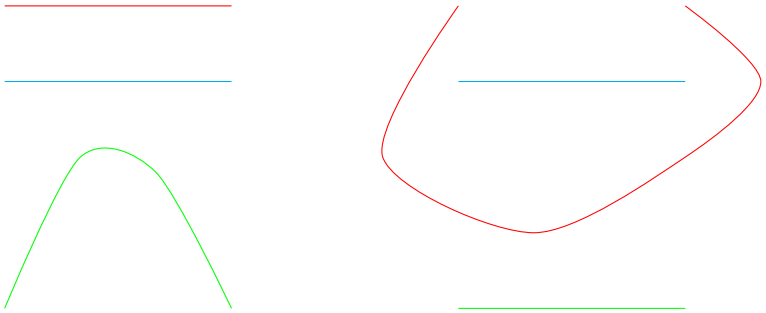


FIGURE 6.2: Non-Homologous Paths

We shall now precisely describe this notion of homologous paths over graphs.

6.2 Flows on graphs

We are going to formalize this notion of homologous paths via the way of flows on graphs, so let us take a moment to re-visit flows and see where the known techniques fail.

Assigning a unit capacity to all the edges of G, we see that if there exists a flow of value atleast 1 from s to t, then we can find a path from s to t, for any two vertices s and t.

Now suppose that, instead, we are given sets of vertices $S = \{s_i\}_{i \leq k}$ and $T = \{t_i\}_{i \leq k}$ such that $S \cap T = \phi$, then we can add a super source s and a super sink t, and connect s to all sources in S and all sinks in T to t and so we just have to find a flow of value atleast k from s to t. But notice that, when extracting paths from this flow, we just get disjoint paths from sets S to T and not particularly from s_i to t_i , as shown in the example below. Moreover, adding a super-source and super-sink, can possibly destroy the planarity of the graph, for e.g. see figure 6.3.

Therefore, we need to add more constraints to our above system so that we get paths for each (s_i, t_i) pair. We do this by distinguishing the flow out of s_i (into t_i) with distinct colours. For instance we can say that unit "red" flow from s_1 to t_1 and unit "blue" flow from s_2 to t_2 and so on. Let us identify these colours with variables g_i, $1 \leq i \leq k$. If a flow of unit g_1 is going out of a vertex v, then we'd also like to say that a flow of unit g_1^{-1} is going in v. And finally we should be able to "sum" these outgoing/incoming flows to compute net flow at v. Hence, we want to impose the structure of (free) group G_k on variables $\{g_1, \ldots, g_k\}$.

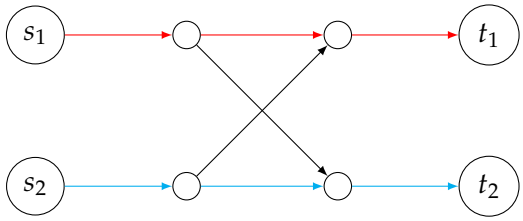


FIGURE 6.3: Distinguishing flows using colours

Formally, let $G = (V, E)$ be a map, with $(s_1, t_1), (s_2, t_2), \ldots (s_k, t_k)$ pairs of vertices. Assume without loss of generality that each s_i and t_i has degree 1.

We call a function $\phi : E \to G_k$ **flow** if:

- Distinct unit flow in/out of each terminal: $\phi(e) = g_i$ if $e = (s_i, u)$ or $e = (v, t_i)$

- Conservation at all other vertices: for every vertex v, $\phi(e_1)^{\epsilon_1}\phi(e_2)^{\epsilon_2}\dots\phi(e_l)^{\epsilon_l} = 1$ where $e_1, \dots, e_l$ are the edges incident on v in clockwise order and $\epsilon_i = 1$ (or -1) if v is tail of e_i (or v is head of e_i)

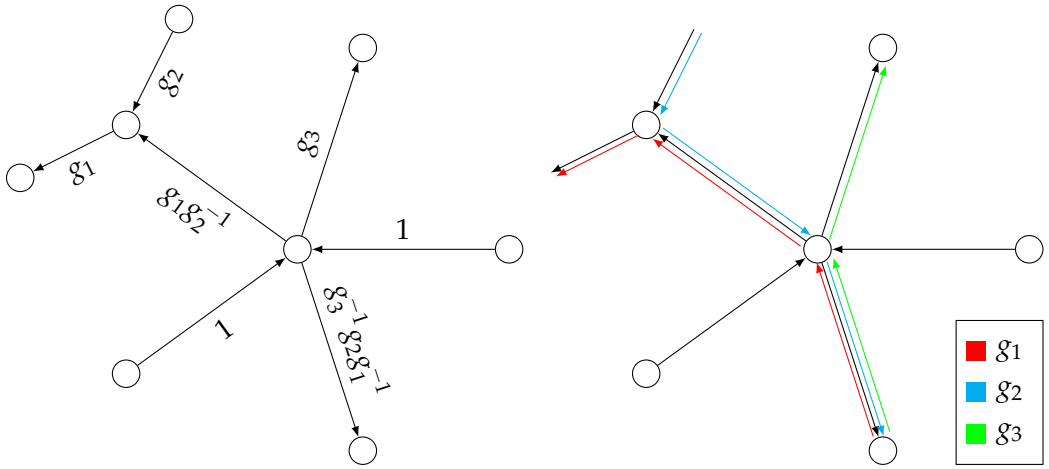


FIGURE 6.4: Interpretation of flow function

Naturally, given a solution $\Pi = (P_1, \dots, P_k)$, we can associate a flow ψ_Π with it, which sends a flow of unit g_i along the path P_i, $1 \le i \le k$.

But it is easy to find some flow in a graph by brute force. So conversely, we need a way to transform a flow into a solution, if possible.

6.3 Disjoint paths from a flow

We are given a flow $\phi : E \to G_k$. We are going to transform this flow by "shifting via faces".

Let F is the set of all faces and fix a face $F_0 \in F$. Now consider any function $f : F \to G_k$ such that $f(F_0) = 1$. We modify this flow is by conjugating by f as follows: $f(L)^{-1}\phi(e)f(R)$ for every edge e, where L and R are the left and right faces of e respectively. This has the effect of pulling the threads $f(L)$ and appending the threads $f(R)$ to the edge e.

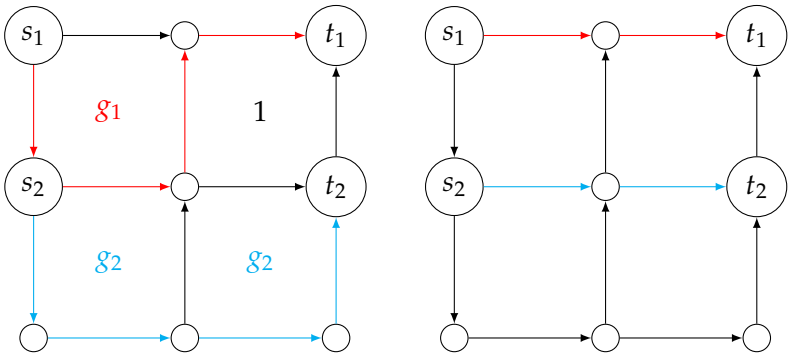


FIGURE 6.5: Shifting via Faces

Formally, we say two flows ϕ, ψ are **homologous** if there exists a function f as above such that we can transform ϕ to ψ using these face shifts prescribed by f, that is

$$\psi(e) = f(L)^{-1}\phi(e)f(R)$$

for each edge e where L, R are left and right faces of the edge e respectively. Also observe that homologous criterion is actually an equivalence relation on the set of all flows.

Now suppose we could modify ϕ and obtain a new flow ψ such that $\psi(e) \in \{1, g_1, \ldots, g_k\}$, then it is clear that we can extract *edge-disjoint* paths from our ψ flow. This is our main goal - to transform ϕ (using the above described transformation rule) to obtain a new flow with this additional restriction.

But instead of a direct approach, we will translate the transformation to dual of the graph, which makes the formulation easier and more intuitive - cycles passing through a fixed vertex. So consider the dual graph G^* and we translate our problem from planar to its dual as follows:

- We had a function (flow) ϕ on E so now we get a function ϕ^* on E^*: for each edge $e \in E$, we have an edge $e^* \in E^*$; set $\phi^* : E^* \to G_k$ as $\phi^*(e^*) = \phi(e)$.
- We had a function f on the faces of G so now we get a function a f on the vertices of G^*.
- In shifting via faces, we had $\psi(e) = f(L)^{-1}\phi(e)f(R)$ where L, R were left and right faces of e respectively. In the dual graph, this will translate to: $\psi^*(e^*) = f(L)^{-1}\phi^*(e^*)f(R)$ where $e^* = (L, R)$.

Therefore, given a directed graph G with a fixed vertex r and two functions $\phi^*, \psi^* : E \to G_k$, we say ψ^* is **cohomologous** to ϕ^* if there exists a function $f : V \to G_k$ such that $f(r) = 1$ and for each edge $e^* = (u, v)$

$$\psi^*(e^*) = f(u)^{-1}\phi^*(e^*)f(v)$$

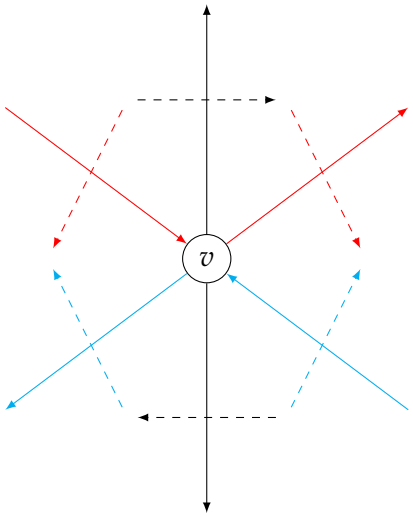


FIGURE 6.6: Transforming into dual

We will show that there exists a polynomial time algorithm (for any directed graph and not necessarily planar) to transform our function ϕ^* into ψ^* (if one exists) so that $\psi^*(e^*) \in \{1, g_1, \ldots, g_k\}$, by finding a function $f : V \to G_k$

We will show how to do this transformation in the next section but let us see first how this helps. We translate back. From ψ^* we can obtain a flow $\psi : E \to G_k$ by setting $\psi(e) = \psi^*(e^*)$. We have $\psi(e) \in \{1, g_1, \ldots, g_k\}$. But that immediately gives us edge-disjoint paths from each s_i to t_i, as discussed above. (It is also interesting to note that these edge-disjoints paths will also be non-crossing).

To obtain *vertex disjoint paths*, we need to do a little more work. We shall use the fact that the cohomologous transformation can be done for any directed graph. So let us see how to rectify this. Consider how vertex-disjoint paths and edge-disjoint paths look like in the dual graph.

We see that if the paths were vertex-disjoint then each face in the dual graph will have edges only of a single colour or no colours (if none of the paths used that vertex). But if two paths shared a vertex, then the face corresponding to that vertex will have edges of different colours (as in the example above).

Main idea: Net flow across any two vertices of a face should be zero or unit.

Before we apply our transformation, we join all the vertices on this face by chords as follows: let u and v be any two vertices on a face, and let $\pi = (e_1^*)^{\epsilon_1} \ldots (e_l^*)^{\epsilon_l}$ be the (undirected) path from u to v obtained by going from u to v in the clockwise direction along this face. Then add an edge e_π from u to v.

Furthermore, we also need to specify the ϕ^* value of these new edges, which we simply set to: $\phi^*(e_\pi) = \phi(e_1)^{\epsilon_1} \ldots \phi(e_l)^{\epsilon_l}$. Now we do the transformation of ϕ^* to ψ^* such that $\psi^*(e^*) \in \{1, g_1, \ldots, g_k\}$ (our old requirement) and $\psi^*(e_\pi) \in \{1, g_1, g_1^{-1}, \ldots, g_k, g_k^{-1}\}$ (new requirement).

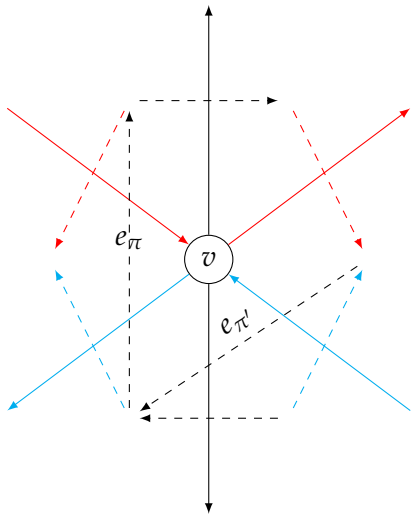


FIGURE 6.7: $\psi^*(e_\pi) = g_2 g_1^{-1}$, $\psi^*(e_{\pi'}) = g_2^{-1}$

Let us justify this new requirement and see how it helps in obtaining vertex-disjoint paths, using the above example 6.7. We have marked undirected paths π, π' and added extra edges $e_\pi, e_{\pi'}$ and so we get that $\psi^*(e_\pi) \in \{1, g_1, g_1^{-1}, \ldots\}$ iff this face had only one (or none) colour edges. Hence adding extra edges and this new restriction gives vertex-disjoint paths.

6.4 Dual and cohomology

Given a graph G (not necessarily planar), a fixed vertex $r \in V$ and a function $\phi : E \to G_k$, we note that ϕ lifts to a homomorphism $\widetilde{\phi} : \pi_1(G, r) \to G_k$ by simply assigning to each cycle

$$C = (r \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \to \cdots \xrightarrow{a_{n-1}} v_{n-1} \xrightarrow{a_n} r) \mapsto \phi(a_1)\phi(a_2)\ldots\phi(a_n)$$

.

We ask the question if there is a function $\psi : E \to G_k$ such that $\widetilde{\psi} = \widetilde{\phi}$ and moreover $\psi(e) \in \{1, g_1, \ldots, g_k\}$?

Here we have assumed that for every edge $e = (u, v)$, we have added an edge $e^{-1} = (v, u)$ and assign $\phi(e^{-1}) = \phi(e)^{-1}$. We observe that: $\widetilde{\psi} = \widetilde{\phi}$ iff $\exists f : V \to G_k$ such that $f(r) = 1$ and $\psi(e) = f(u)^{-1}\phi(e)f(v)$ where $e = (u, v)$. We will reformulate the general problem as follows: Given $\phi : E \to G_k$, find a function $f : V \to G_k$ such that

- $f(r) = 1$
- $f(u)^{-1}\phi(e)f(v) \in \Gamma(e)$ for every $e = (u, v)$

where $\Gamma(e) \subseteq G_k$ is such that if $x \in \Gamma(e)$ and y is a substring of x then $x \in \Gamma(e)$.

We call such functions f *feasible*. It is not clear apriori how to build a feasible function. So we define *pre-feasible* functions. Call a function $f : V \to G_k$ pre-feasible if:

- $f(r) = 1$
- If $f(u)^{-1}\phi(e)f(v) \notin \Gamma(e)$ for some e, then $f(u) = f(v) = 1$

With this, we get a head start as $f \equiv 1$ is a pre-feasible function. But this is a trivial function to start with. So, fix an edge $e = (u, v)$ and consider the function f_e defined as $f_e(u) = \phi(e)$ and $f_e(w) = 1$ for each $w \neq u$ (f_e might not be pre-feasible).

Using the lattice structure on G_k, we obtain a lattice structure on set of all functions $f : E \to G_k$. There exists a $O(|V|^2)$ time algorithm, to change f_e to the *smallest* pre-feasible function $\widetilde{f_e}$. So we get a collection $\widetilde{f_e}$ of pre-feasible functions. We construct a feasible function out of them as follows, by solving a special case of 2SAT.

Call a pair (e, e') compatible if $\widetilde{f_e} \vee \widetilde{f_{e'}}$ is finite and pre-feasible. Call an edge bad if $\phi(e) \notin \Gamma(e)$.

We try to find a subset X of E such that every pair in X is compatible and for each bad edge e: e or e^{-1} is in X. If X is empty, then no feasible function exists. Otherwise, finally set $f = \vee_{e \in X} \widetilde{f_e}$, and f is the feasible function as required.

6.5 Guessing flows

Now that we know how to find a solution from a flow, we need to find some "candidate" flows. We are going to exhibit $poly(n)$ flows such that if there was a solution, then by applying our transformation to one of these flows, we would get a solution.

Consider any edge e, not incident to any of the terminals. Contract it to obtain a new graph G'. Now suppose we have a flow ϕ' on G', then there is a unique flow ϕ on G obtained by setting $\phi(f) = \phi'(f)$ for all edges f except e. And $\phi(e)$ is obtained by flow conservation at the corresponding vertices - as in example below.

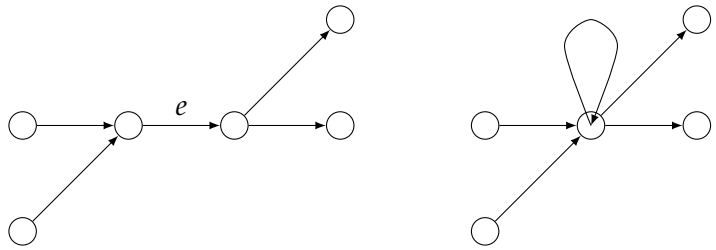


FIGURE 6.8: Contract Edges

Consider the flow on this new graph, as in example below. We want to obtain back a flow on the old graph. We already know the ϕ value of all edges (except the one which was contracted) By flow conservation on the tail of this edge e, we get: $g_1\phi(e)^{-1}g_2^{-1} = 1 \implies \phi(e) = g_1^{-1}g_2$ and $g_1^{-1}g_2$ means that g_1 going out and g_2 coming in - which matches with our intuition of this flow.

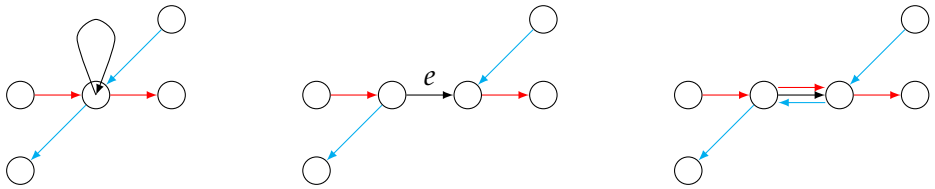


FIGURE 6.9: Obtaining unique flow from contracted edge

So it suffices to obtain these set S of $poly(n)$ flows on the graph where all edges (except the ones incident on terminals) have been contracted. This we do by brute force as there are only $n^{O(k)}$ possible flows in this little graph. The set of flows so obtained has the property that if a solution exists, then it must be homologous to one of the flows in S.

6.6 Main Algorithm

Thus we can obtain a solution for k-disjoint paths in $O(n^k)$ time. To summarize:

- Enumerate all the candidate flows in S
- For each flow $\phi \in S$, translate it to the dual graph ϕ^*
- Add auxillary edges (for vertex-disjointness) to the dual graph
- Now find a feasible function, if one exists, and obtain ψ^* by transforming ϕ^* via f and translate back ψ^* to the original graph ψ to get a solution
- If none of the flows gave a solution, then output NO disjoint paths

6.7 Extensions

The above algorithm can be easily extended to bounded genus graphs. It only suffices to carefully enumerate all the flows and the rest of the algorithm remains the same.

Later Schrijver also showed that by introducing relations amongst the group elements, we can also find partially disjoint paths [Sch15].

Chapter 7

Conclusion

May all be Happy ◊ May all be free from illness;
May all see what is Auspicious ◊ May no one Suffer.

Upanishads

We started by introducing the combinatorial graph problem of finding disjoint paths and the algebraic problem of computing permanent mod powers of 2. Then we discussed a bridge using which we connect these two. Following which we demonstrated an application of this bridge - shortest 2-disjoint paths problem [BH19].

Our first goal was to generalize this problem by putting additional restrictions on these paths so that they pass through given edges. Adjusting our perspective, we realized this as a problem of finding disjoint cycles passing through given vertices/edges. In particular, we exhibited an algorithm to find shortest 2-disjoint cycles passing through given vertices/edges, which also answers our above generalization. Additionally, we also presented an algorithm to find a shortest disjoint cycle passing through any given fixed number of vertices/edges, although this was already reminiscent in the work of [Wah13].

Our next goal was to strengthen the bridge, that is to exhibit a parallel algorithm to compute permanent of matrices of integer polynomials. We started by recognizing the appropriate algebraic structure $\mathfrak{R}$ over which we can present a parallel algorithm to compute permanent modulo 2^k. Then we saw two techniques to get permanent over $\mathbb{Z}[x] \pmod{2^k}$ from $\mathfrak{R} \pmod{2^k}$. First method was to choose a large enough irreducible polynomial for our ring $\mathfrak{R}$. Another method was to interpolate over the ring $\mathfrak{R}$, which was an extension of the commonly known interpolation over finite fields. Finally, this yields a (randomized) parallel algorithm for:

- finding shortest 2-disjoint paths
- finding a shortest disjoint cycle
- finding shortest 2-disjoint cycles

The more general question of finding shortest k disjoint cycles for $k \geq 3$ passing through any given vertices/edges still remains open. Can we use the framework presented in here, of using combination of permanents of different pattern graphs to answer more problems? On the other hand, the problem of computing permanent over arbitrary commutative rings of characteristic 2^k for $k > 1$ also seems interesting.

Bibliography

[BH18] Andreas Björklund and Thore Husfeldt. "Counting Shortest Two Disjoint Paths in Cubic Planar Graphs with an NC Algorithm". In: *CoRR* abs/1806.07586 (2018). arXiv: 1806.07586. URL: http://arxiv.org/abs/1806.07586.

[BH19] Andreas Björklund and Thore Husfeldt. "Shortest Two Disjoint Paths in Polynomial Time". In: *SIAM J. Comput.* 48.6 (2019), pp. 1698–1710. DOI: 10.1137/18M1223034. URL: https://doi.org/10.1137/18M1223034.

[BKR09] Mark Braverman, Raghav Kulkarni, and Sambuddha Roy. "Space-Efficient Counting in Graphs on Surfaces". In: *Comput. Complex.* 18.4 (2009), pp. 601–649. DOI: 10.1007/s00037-009-0266-4. URL: https://doi.org/10.1007/s00037-009-0266-4.

[Cyg+13] Marek Cygan et al. "The Planar Directed K-Vertex-Disjoint Paths Problem Is Fixed-Parameter Tractable". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 197–206. DOI: 10.1109/FOCS.2013.29. URL: https://doi.org/10.1109/FOCS.2013.29.

[Dam90] Carsten Damm. "Problems complete for parityL". In: *Information Processing Letters* 36.5 (1990), pp. 247–250. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(90)90150-V. URL: https://www.sciencedirect.com/science/article/pii/002001909090150V.

[Dat+18] Samir Datta et al. "Shortest k-Disjoint Paths via Determinants". In: *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*. Ed. by Sumit Ganguly and Paritosh K. Pandya. Vol. 122. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 19:1–19:21. DOI: 10.4230/LIPIcs.FSTTCS.2018.19. URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2018.19.

[Ebe84] W. Eberly. "Very Fast Parallel Matrix and Polynomial Arithmetic". In: *25th Annual Symposium onFoundations of Computer Science, 1984.* 1984, pp. 21–30. DOI: 10.1109/SFCS.1984.715897.

[Ebe91] W. Eberly. "Efficient parallel independent subsets and matrix factorizations". In: *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*. 1991, pp. 204–211. DOI: 10.1109/SPDP.1991.218278.

[FHW80] Steven Fortune, John E. Hopcroft, and James Wyllie. "The Directed Subgraph Homeomorphism Problem". In: *Theor. Comput. Sci.* 10 (1980), pp. 111–121. DOI: 10.1016/0304-3975(80)90009-2. URL: https://doi.org/10.1016/0304-3975(80)90009-2.

[FT88] Faith E. Fich and Martin Tompa. "The Parallel Complexity of Exponentiating Polynomials over Finite Fields". In: *J. ACM* 35.3 (June 1988), pp. 651–667. ISSN: 0004-5411. DOI: 10.1145/44483.44496. URL: https://doi.org/10.1145/44483.44496.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. "Uniform constant-depth threshold circuits for division and iterated multiplication". In: *J. Comput. Syst. Sci.* 65.4 (2002), pp. 695–716. DOI: 10.1016/S0022-0000(02)00025-9. URL: https://doi.org/10.1016/S0022-0000(02)00025-9.

[HN18] Hiroshi Hirai and Hiroyuki Namba. "Shortest (A+B)-Path Packing Via Hafnian". In: *Algorithmica* 80.8 (2018), pp. 2478–2491. DOI: 10.1007/s00453-017-0334-0. URL: https://doi.org/10.1007/s00453-017-0334-0.

[JVW20] Ce Jin, Nikhil Vyas, and Ryan Williams. "Fast Low-Space Algorithms for Subset Sum". In: *CoRR* abs/2011.03819 (2020). arXiv: 2011.03819. URL: https://arxiv.org/abs/2011.03819.

[Lin13] J.H. van Lint. *Introduction to Coding Theory*. Graduate Texts in Mathematics. Springer Berlin Heidelberg, 2013. ISBN: 9783662079980. URL: https://books.google.co.in/books?id=6dbqCAAAQBAJ.

[Lyn75] James F. Lynch. "The Equivalence of Theorem Proving and the Interconnection Problem". In: *SIGDA Newsl.* 5.3 (Sept. 1975), pp. 31–36. ISSN: 0163-5743. DOI: 10.1145/1061425.1061430. URL: https://doi.org/10.1145/1061425.1061430.

[MSV04] Meena Mahajan, P. R. Subramanya, and V. Vinay. "The combinatorial approach yields an NC algorithm for computing Pfaffians". In: *Discret. Appl. Math.* 143.1-3 (2004), pp. 1–16. DOI: 10.1016/j.dam.2003.12.001. URL: https://doi.org/10.1016/j.dam.2003.12.001.

[Mul87] Ketan Mulmuley. "A fast parallel algorithm to compute the rank of a matrix over an arbitrary field". In: *Comb.* 7.1 (1987), pp. 101–104. DOI: 10.1007/BF02579205. URL: https://doi.org/10.1007/BF02579205.

[MV97] Meena Mahajan and V. Vinay. "Determinant: Combinatorics, Algorithms, and Complexity". In: *Chic. J. Theor. Comput. Sci.* 1997 (1997). URL: http://cjtcs.cs.uchicago.edu/articles/1997/5/contents.html.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. "Matching is as Easy as Matrix Inversion". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 345–354. ISBN: 0897912217. DOI: 10.1145/28395.383347. URL: https://doi.org/10.1145/28395.383347.

[OJ05] Pavel Okunev and Charles R. Johnson. "Necessary And Sufficient Conditions For Existence of the LU Factorization of an Arbitrary Matrix". In: *arXiv Mathematics e-prints*, math/0506382 (June 2005), math/0506382. arXiv: math/0506382 [math.NA].

[RA00] Klaus Reinhardt and Eric Allender. "Making Nondeterminism Unambiguous". In: *SIAM J. Comput.* 29.4 (2000), pp. 1118–1131. DOI: 10.1137/S0097539798339041. URL: https://doi.org/10.1137/S0097539798339041.

[RS95] N. Robertson and P.D. Seymour. "Graph Minors .XIII. The Disjoint Paths Problem". In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 65–110. ISSN: 0095-8956. DOI: https://doi.org/10.1006/jctb.1995.1006. URL: https://www.sciencedirect.com/science/article/pii/S0095895685710064.

[RWW96] Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe. "Linear-Time Algorithms for Disjoint Two-Face Paths Problems in Planar Graphs". In: *Int. J. Found. Comput. Sci.* 7.2 (1996), pp. 95–110. DOI: 10.1142/S0129054196000087. URL: https://doi.org/10.1142/S0129054196000087.

[SAN90] Hitoshi Suzuki, Takehiro Akama, and Takao Nishizeki. "Finding Steiner Forests in Planar Graphs". In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA*. Ed. by David S. Johnson. SIAM, 1990, pp. 444–453. URL: http://dl.acm.org/citation.cfm?id=320176.320230.

[Sch15] Alexander Schrijver. "Finding *k* partially disjoint paths in a directed planar graph". In: *CoRR* abs/1504.00185 (2015). arXiv: 1504.00185. URL: http://arxiv.org/abs/1504.00185.

[Sch94] Alexander Schrijver. "Finding k Disjoint Paths in a Directed Planar Graph". In: *SIAM J. Comput.* 23.4 (1994), pp. 780–788. DOI: 10.1137/S0097539792224061. URL: https://doi.org/10.1137/S0097539792224061.

[Val79] Leslie G. Valiant. "The Complexity of Computing the Permanent". In: *Theor. Comput. Sci.* 8 (1979), pp. 189–201. DOI: 10.1016/0304-3975(79)90044-6. URL: https://doi.org/10.1016/0304-3975(79)90044-6.

[VS11] Éric Colin de Verdière and Alexander Schrijver. "Shortest vertex-disjoint two-face paths in planar graphs". In: *ACM Trans. Algorithms* 7.2 (2011), 19:1–19:12. DOI: 10.1145/1921659.1921665. URL: https://doi.org/10.1145/1921659.1921665.

[Wah13] Magnus Wahlström. "Abusing the Tutte Matrix: An Algebraic Instance Compression for the K-set-cycle Problem". English. In: *STACS*. 2013, pp. 341–352.

[ZAN91] VIKTÓRIA ZANKÓ. "#P-COMPLETENESS VIA MANY-ONE REDUCTIONS". In: *International Journal of Foundations of Computer Science* 02.01 (1991), pp. 77–82. DOI: 10.1142/S0129054191000066. eprint: https://doi.org/10.1142/S0129054191000066. URL: https://doi.org/10.1142/S0129054191000066.