6.856 Project Report: Algebraic Method in Graph Algorithms

Lijie Chen Xiao Mao

May 14, 2019

1 Introduction

In 2010, it was a major surprise that Andreas Bjorklund discovered a randomized algorithm for undirected Hamiltonicity in $O(1.657^n)$ time [Bjö14], which was thought to be impossible by many researchers in this field. This algorithm improved the previous textbook $2^n \cdot \text{poly}(n)$ time dynamic algorithm by Bellman [Bel62] and independently Held and Karp [HK62], which had been refusing improvement for nearly fifty years.

The major idea behind Bjouklund's breakthrough is the use of algebraic techniques to solve graph problems. This also leads to several ingenious algorithms for other problems, including Williams' $2^k \cdot \text{poly}(n)$ time algorithm for finding a k-path in a graph [Wil09], Bjorklund and Husfeldt's polynomial-time algorithm for shortest two disjoint path problem [BH14], and Bjorklund et al.'s algorithm for finding the shortest simple cycle passing through the given k elements [BHT12]. This paper will focus on explaining the underlying ideas behind all these algorithms.

We first define these problems formally:

- **Undirected Hamiltonicity.** [Bjö14]. Given an undirected graph G of n vertices. [Bjö14] showed how to determine whether there is a Hamiltonian cycle (a simple cycle visiting all vertices *exactly once*) in 1.657^n time.

  We remark that for the case of *directed* hamiltonicity, it is still an open question that whether there is a c^n time algorithm for a constant $c < 2$. In 2013, Bjorklund and Husfeldt [BH13] made some progress by presenting a 1.619^n time algorithm for determining the parity of the number of directed Hamiltonian cycles.

- **Shortest Two Disjoint Paths.** Given an undirected graph G with two pairs of vertices (s_i, t_i) for $i \in [2]$, the shortest two disjoint paths problem asks to find two *disjoint* paths P_1, P_2 (that is, P_1 and P_1 don't share any vertices) with minimum total length $|P_1| + |P_2|$ such that P_i connects s_i and t_i for $i \in [2]$. Despite the fact that it appears to be a classical problem which *should have been solved before 1980s*, the first polynomial-time algorithm was only given by Bjorklund and Husfeldt in 2014 [BH14]. Previously, even a sub-exponential time algorithm is unknown.

- **Shortest K-Cycle.** Given an undirected graph $G = (V, E)$ with a subset $K \subseteq V$ of size k. The Shortest K-Cycle problem asks to find the shortest simple cycle C passing all vertices in K. In 2012, Bjorklund et al. [BHT12] presented an illuminating algorithm which solves this problem in $O(2^k \cdot \text{poly}(n))$ time. Previously, only algorithms of running time *double-exponential* in k were known, (i.e., $2^{2^{O(k)}} \cdot \text{poly}(n)$ time).

- **k-Path.** Given a directed graph $G = (V, E)$ and an integer k. The k-Path problem asks to find a simple path in G with length k. In 2009, building on an earlier work of Koutis [Kou08], Williams [Wil09] showed how to solve this problem in $O(2^k \cdot \text{poly}(n))$ time. We remark the naive search algorithm runs in $n^{O(k)}$ time, which is super-polynomial for any $k = \omega(1)$.

The Main Theme: Polynomials and Magical Cancellation

Let us try to explain the most important ideas in these algorithms—*Algebrization via polynomials* and *Cancellation of unwanted items*. We take the beautiful algorithm for Shortest K-Cycle as an example.

Given a graph $G = (V, E)$, for each edge $e \in E$, we assign a different variable x_e. Given a closed walk $w = v_0, v_1, \ldots, v_\ell = v_0$, we define its corresponding polynomial to be $W(w) := \prod_{i=0}^{\ell-1} x_{(v_i, v_{i+1})}$.

Ideally, we want to be able to evaluate the following polynomial:

$$P_{\text{simple-}K\text{-cycle}} := \sum_{w \text{ is a simple cycle of length } \ell \text{ visiting all vertices in } K} W(w).$$

To avoid double counting, we can fix a specific $s \in K$ as the starting point ($v_0 = s$), and force $v_1 < v_{\ell-1}$ to decide an orientation of the cycle. Suppose we were able to evaluate $P_{\text{simple-}K\text{-cycle}}$ on random assignments, then we can check the existence of a K-cycle of length ℓ by simply testing whether this polynomial is nonzero. This also suffices to find the actual shortest K-Cycle using self-reducibility of this problem.

Of course $P_{\text{simple-}K\text{-cycle}}$ looks hard to evaluate directly. So the algorithm evaluates the following polynomial instead:

$$P_{\text{closed-}K\text{-walk}} := \sum_{w \text{ is a certain good closed-walk of length } \ell} W(w).$$

We do not specify what is a good walk here (which is pretty technical, the purpose here is to give reader a general feeling of how things are achieved without too much details; see Section 4 for the actual details). But clearly, a simple K-cycle should be a good walk. Also, note that the above polynomial could also contain some *unwanted items*, which are good walks but not simple K-cycles. The ingenious idea from [BHT12] is, if we carefully define *good walks*, and work with a field $\mathbb{F}_q$ where q is a power of 2 (in such a field, $2 = 0$), we can make sure all unwanted items (good walks but not simple K-cycles) are counted an even number of times. In other words, they cancel each other in $P_{\text{closed-}K\text{-walk}}$!

Therefore, magically, $P_{\text{closed-}K\text{-walk}}$ indeed equals $P_{\text{simple-}K\text{-cycle}}$. And we can proceed *as if we can evaluate* $P_{\text{simple-}K\text{-cycle}}$, assuming that $P_{\text{closed-}K\text{-walk}}$ is easier to evaluate.

Organization of the Paper

Basing on the ways these algorithms construct the corresponding polynomials, we categorize them into two classes.

- **Permanent/Determinant Polynomial and Cycle Cover.** Given a matrix A, the determinant of A, denoted as $\det(A)$, is one of the most fundamental concept in linear algebra (perhaps also in mathematics). (See Section 3 for a formal definition). $\det(A)$ can be seen as a degree-n polynomial over entries of the matrix A, and moreover it is computable over any *rings* in *polynomial time*. We recall that over *fields*, the classical Gaussian elimination method gives an $O(n^3)$ time algorithm.

  The first two algorithms surveyed by us, for **Undirected Hamiltonicity** and **Shortest Two Disjoint Paths**, heavily exploit the properties of the determinant polynomial (e.g., $\det(A) = \text{per}(A)$ when working with $\mathbb{F}_q$ for a q which is a power of 2). We present these two algorithms in Section 3.

- **Dynamic Programming.** Another powerful method to construct polynomials is through *dynamic programming*. The last two algorithms surveyed, for **k-Path** and **Shortest K-Cycle**, build the corresponding polynomial via sophisticated dynamic programming. These two algorithms are presented in Section 4.

2 Preliminaries about Polynomials

All algorithms surveyed by us highly exploit properties of polynomials. Here we first recall some notations about polynomials. We use $\mathbb{F}_q$ to denote the finite field of size q, and $\mathbb{F}_q[x_1, x_2, \ldots, x_n]$ to denote the set

of all polynomials on variables $x_1, \ldots, x_n$ over $\mathbb{F}_q$. A monomial M is a polynomial of the form $\prod_{i=1}^{n}(x_i)^{e_i}$, where e_i's are non-negative integers. The degree of M is $\sum_{i=1}^{n} e_i$, and the degree of a polynomial p is the maximum degree of all monomials in p.

The following classic lemma allows us to test whether a low-degree polynomial is nonzero by assigning random values to its variables.

Lemma 2.1 (DeMillo-Lipton-Schwartz-Zippel)**.** *Let $p \in \mathbb{F}_q[x_1, x_2, \ldots, x_n]$ be a nonzero polynomial of degree at most d. Then, for $z_1, z_2, \ldots, z_n \in \mathbb{F}_q$ selected independently and uniformly at random, $p(z_1, z_2, \ldots, z_n) = 0$ with probability at most d/q.*

3 Permanent Polynomial and Cycle Cover

The first set of algorithms surveyed by us makes crucial use of the *permanent* polynomials over a suitable field (ring) and its combinatorial characterization via ***cycle covers***.

We first recall the definition of a permanent polynomial.

Permanent and Determinant. Let $x_1, x_2, \ldots, x_m$ be some variables, and $R = \mathbb{F}_q[x_1, x_2, \ldots, x_m]$ (that is, R is the set of polynomials of x_i's over $\mathbb{F}_q$). Consider a matrix $A \in R^{n \times n}$ (that is, A is an n by n matrix whose entries are polynomials). We define

$$\mathrm{per}(A) := \sum_{\sigma:[n]\leftrightarrow[n]} \prod_{i=1}^{n} A_{i,\sigma(i)},$$

where the sum is over all *permutations* σ on $[n]$ ($[n] \leftrightarrow [n]$ means the set of bijections).

The above bears some similarity with the definition of the *determinant*, which is defined as

$$\det(A) := \sum_{\sigma:[n]\leftrightarrow[n]} (-1)^{\mathrm{sign}(\sigma)} \prod_{i=1}^{n} A_{i,\sigma(i)},$$

where $\mathrm{sign}(\sigma) \in \{0, 1\}$ denotes the sign of the permutation σ.

We remark that both $\det(A)$ and $\mathrm{per}(A)$ are polynomials from R as well. The most important observation is that, when q is a power of 2, we have $-1 = 1$ in $\mathbb{F}_q$, and therefore $\det(A) = \mathrm{per}(A)$ (the definition of $\mathrm{sign}(\sigma)$ is not important at all here). Since $\det(A)$ can be evaluated in $O(n^3)$ time, $\mathrm{per}(A)$ can also be evaluated in $O(n^3)$ when we are working over field $\mathbb{F}_q$ where q is a power of 2.

Cycle Cover. Let $A \in R^{n \times n}$ and $\sigma : [n] \leftrightarrow [n]$ be a permutation on $[n]$. We construct a directed graph on n vertices, such that for $(i, j) \in [n] \times [n]$, the edge $i \to j$ is labeled by polynomial $A_{i,j} \in R$.

Now, consider the sub-graph formed by edges $\{i \to \sigma_i : i \in [n]\}$. Since in this sub-graph each vertex has both in-degree 1 and out-degree 1, it actually consists of several ***disjoint directed cycles***, which cover all the vertices exacty once.

We define a cycle cover C of n vertices, as a collection of disjoint directed cycles which cover all the vertices in $[n]$. And we define the weight of C as $W(C) = \prod_{(i\to j)\in C} A_{i,j}$. It is easy to see

$$\mathrm{per}(A) = \sum_{C \in cc([n])} W(C),$$

where $cc([n])$ is all cycle covers on $[n]$. See Figure 1 for two possible cycle covers on a graph of 5 vertices, while the first one has two disjoint cycles, and the second one has a big cycle.

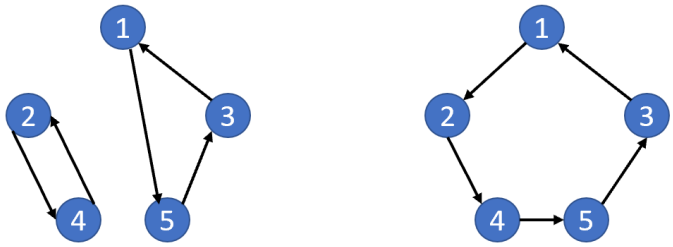

Figure 1: Two cycle covers for a graph with 5 vertices, corresponding to permutation $(1,5,3),(2,4)$ and permutation $(1,2,4,5,3)$.

3.1 An $2^{n/2}\cdot\text{poly}(n)$ time algorithm for Hamiltonicity on a Undirected Bipartite Graph

Now we explain how [Bjö14] used the ideas of permanent polynomial and its cycle cover characterization to decide whether an *undirected*, *bipartite* graph has a Hamiltonian cycle in $2^{n/2}\cdot\text{poly}(n)$ time. Note that [Bjö14] indeed also gave an 1.657^n time algorithm for *general* undirected graphs as well, but we think this special case of bipartite graphs is easier to present, and already contains most of his ingenious ideas.

Note that a Hamiltonian cycle must alternate between two sides of the bipartite graph, therefore the bipartite graph must be *balanced*.

Reduction to Labeled Hamiltonicity on $n/2$ Vertices

The first insight of [Bjö14] is to reduce this problem to another problem on $n/2$ vertices; and solve the later problem in $2^{n/2}\cdot\text{poly}(n)$ time.

Let two sides of vertices in this balanced bipartite graph G be U and V. We have $|U|=|V|=n/2$. If we fix the starting vertex to be $s=u_1\in U$, then a Hamiltonian cycle on G must have the following structure: for each $i\in[n/2]$, it first goes from u_i to $v_i\in V$, and then goes back to $u_{i+1}\in U$, and finally ends at $u_{n/2+1}=u_1$.

The idea is to treat vertices in V as *labels* (each vertex $v\in V$ is assigned to a unique label in $[n/2]$), and focus on vertices in U. Instead of going from $u_i\to v_i\to u_{i+1}$, we interpret it as going from $u_i\to u_{i+1}$ directly via an edge with *label* v_i.

Now we arrive at the following problem:

Problem 1 (Labeled Hamiltonicity)**.** *Given an undirected graph on $[n/2]$ vertices, with edges labeled by integers in $[n/2]$. We want to find a Hamiltonian cycle such that all edges labels on it are unique (in other words, the edge labels used are exactly $[n/2]$).*

The reduction is straightforward to implement: for each pair of vertices $a,b\in U$, if for $v\in V$, there are edges (a,v) and (v,b), then we add an edge (a,b) with label v in the new graph.

Solving the Labeled Hamiltonicity Problem in $2^{n/2}\cdot\text{poly}(n)$ Time.

For consistency, we still use U to denote the set of all vertices in the labeled Hamiltonicity problem after the reduction, and V to denote the sets of all labels. Now we define some variables.

Recall that s is the special fixed vertex treated as the start of the cycle. For vertex $u\in U\setminus\{s\}$ and label $v\in V$, we define a new variable $x_{u,v}$, and set $x_{v,u}:=x_{u,v}$. For $s\in U$ and label $v\in V$, we define two different variables $x_{s,v}$ and $x_{v,s}$ (they are not equal by definition).

For an edge $u_1\to u_2$ with label v, we define its polynomial as $f_{\mathsf{edge}}(u_1\to u_2,v):=x_{u_1,v}\cdot x_{v,u_2}$. Also, if there is no edge from $u_1\to u_2$ with label v, we set $f_{\mathsf{edge}}(u_1\to u_2,v):=0$.

Labeled Cycle Cover. Since now our edges are labeled, it is natural to generalize the concept of cycle covers to labeled cycle covers. For a cycle cover $C \in cc(U)$, we assign a label from V to all of its edges, and thus form a *labeled cycle cover* $L \in lcc(U)$. That is, L is specified by $\{(e, L_e) : e \in C\}$, where L_e is the corresponding label for edge e. See Figure 2 for two instances of labeled cycle covers.

We use $Lab(L)$ to denote the set of labels in L, and $W(L)$ to denote the polynomial:

$$W(L) := \prod_{(e,L_e)\in L} f_{\mathsf{edge}}(e, L_e).$$

Now, consider the following polynomial

$$P := \sum_{L\in lcc(U), Lab(L)=V} W(L).$$

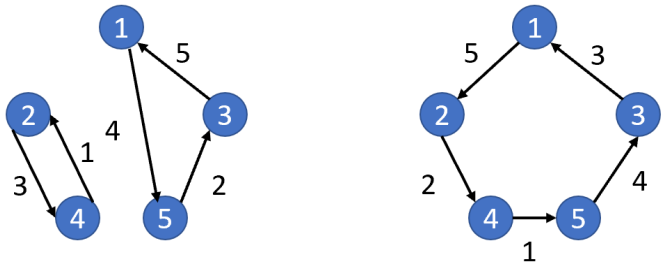

Figure 2: Two labeled cycle covers for a graph with 5 vertices, corresponding to permutation $(1, 5, 3), (2, 4)$ and permutation $(1, 2, 4, 5, 3)$, and with distinct labels.

We claim that the labeled hamiltonicity problem has a solution if and only if P is non-zero.

- To see this, the key observation is that, for all L which is non-hamiltonian (that is, consisting of multiple *disjoint cycles*), $W(L)$ is counted an even number of times, and therefore the contribution of the corresponding $W(L)$'s cancel each other. (recall that we are working with $\mathbb{F}_q$ where q is a power of 2, in which $2 = 0$.)

  Formally, let L be a labeled cycle cover of U, with label $Lab(L) = V$. In particular, this means all edges have distinct labels. We show that if L has multiple cycles, we can always pair it up with a different cycle cover L' with $W(L) = W(L')$, so they cancel each other (the pairing also maps L' to L so it is consistent).

  Let w be the smallest indexed vertex which is not in the same cycle with s, and c be the corresponding cycle of w. We simply reverse the directions of all edges in c. There are two cases: (1) $|c| \geq 3$, in this case, after the reversal c becomes a different cycle. (2) $|c| = 2$, in this case, reversing the direction doesn't change the cycle itself. But one can see that the labels of this two edges in c are switched (recall that the two edges must be $a \to b$ and $b \to a$ for some a, b, and since $Lab(L) = V$, the two edges have distinct labels). In both cases, it is easy to see that we have mapped L to another different labeled cycle cover L' with $W(L') = W(L)$.

- If L is Hamiltonian. The key observation is that reversing the cycle direction changes $W(L)$, as $f_{\mathsf{edge}}(s \to u, v) = x_{s,v} \cdot x_{v,u} \neq f_{\mathsf{edge}}(u \to s, v) = x_{u,v} \cdot x_{v,s}$. (By definition, $x_{s,v} \neq x_{v,s}$. This is also the reason why we have to pick a cycle without s above). It is then not hard to see that $W(L)$ cannot be canceled by other labeled cycle covers.

The Evaluation of P. Fix a subset $S \subseteq V$, consider the following polynomial P_S:

$$P_S := \sum_{C \in cc(U)} \prod_{(a,b) \in C} \left(\sum_{v \in S} f_{\mathsf{edge}}(a \to b, v) \right).$$

One can see that P_S can also be defined via labeled cycle covers as

$$P_S := \sum_{L \in lcc(U), Lab(L) \subseteq S} W(L).$$

That is, the sum of $W(L)$ for all labeled cycle covers, such that it is forbidden to use labels outside of S. By an inclusion-exclusion principle (first consider all labeled cycle covers with labels from $[n]$; then subtract those with labels from $S \subset [n]$ with $|S| = n-1$; then add back those with labels from $S \subset [n]$ with $|S| = n-2$, and etc.), we have that

$$P := \sum_{S \subseteq V} (-1)^{|V \setminus S|} \cdot P_S.$$

Given a random assignment to all the variables $x_{u,v}$'s and $x_{v,u}$'s, one can compute P_S in $O(n^3)$ time by computing the corresponding determinant. Therefore, P on this random assignment can be computed in $2^{n/2} \cdot \mathrm{poly}(n)$ total time. By Lemma 2.1, we know that this works with high probability, provided that we choose $q = 2^r \gg n$.

3.2 A Polynomial-Time Algorithm for Shortest Two Disjoint Paths

Recall that in the shortest two disjoint path problem, we are given a undirected graph G with two pairs of vertices (s_i, t_i) for $i \in [2]$, and the goal is to find two vertex-disjoint paths P_1 and P_2 such that P_i connects s_i and t_i for $i \in [2]$, and the sum of $|P_1| + |P_2|$ is minimized.

We can assume the four vertices $\{s_1, s_2, t_1, t_2\}$ are distinct (as P_1 and P_2 are forbidden to share any vertices). To simplify the presentation, we focus on the case where the optimal solution is *unique*, which already contains most of the important ideas of this algorithm.[1]

Given a graph G, we construct the following matrix $A(G)$ such that

$$A(G)_{i,j} = \begin{cases} x & \text{if } (i,j) \in E; \\ 1 & \text{if } i = j; \\ 0 & \text{otherwise.} \end{cases}$$

Forcing edges in the Cycle Cover. For two pairs of vertices $(a_1, b_1), (a_2, b_2)$, consider the following matrix $A_G[a_1 \to b_1, a_2 \to b_2]$ defined as follows:

$$A(G)[a_1 \to b_1, a_2 \to b_2]_{i,j} := \begin{cases} A(G)_{i,j} & \text{if } i \notin \{a_1, a_2\}; \\ 1 & \text{if } (i,j) = (a_1, b_1) \text{ or } (a_2, b_2); \\ 0 & \text{otherwise.} \end{cases}$$

That is, $A(G)[a_1 \to b_1, a_2 \to b_2]$ deletes all outgoing edges from a_1 and a_2, except for the edges $a_1 \to b_1$, $a_2 \to b_2$. It also sets the labels of these two edges to be 1.

The intuition for doing this operation will become clear if we think about the cycle covers of $A(G)[a_1 \to b_1, a_2 \to b_2]$. Essentially, this operation forces the cycle cover to go through edges $a_1 \to b_1$ and $a_2 \to b_2$, as they are the only non-zero edges leaving a_1 and a_2.

[1] It is possible to reduce the general case to the unique case using the isolation lemma. For an interested reader, it is encouraged to figure out how to do this.

Combinatorial Understanding of per$(A(G)[t_1 \to s_1, t_2 \to s_2])$. By the previous discussion, we have

$$\text{per}(A(G)[t_1 \to s_1, t_2 \to s_2]) = \sum_{C \in cc(G), (t_1 \to s_1), (t_2 \to s_2) \in C} W[t_1 \to s_1, t_2 \to s_2](C),$$

where

$$W[t_1 \to s_1, t_2 \to s_2](C) = \sum_{(a,b) \in C \setminus \{t_1 \to s_1, t_2 \to s_2\}} A(G)_{a,b}.$$

The above is a polynomial in x. For ease of notation, we simply use $W(C)$ to denote $W[t_1 \to s_1, t_2 \to s_2](C)$. Now, note that there are two types of cycle cover C containing edges $(t_1 \to s_1), (t_2 \to s_2)$:

- $(t_1 \to s_1), (t_2 \to s_2)$ are in two different cycles. In this case, C contains two disjoint paths P_1 and P_2, connected by $s_1 \to t_1$ and $s_2 \to t_2$ respectively.
- $(t_1 \to s_1), (t_2 \to s_2)$ are in the same cycle. In this case, C contains a big cycle which passes t_1, s_1, t_2, s_2 in order, which corresponds to two paths $s_1 \to t_2$ and $s_2 \to t_1$ in the graph G.

Let Q_1, Q_2 be the corresponding sum of $W(C)$'s of cycle covers of case 1 and case 2. We have $\text{per}(A(G)[t_1 \to s_1, t_2 \to s_2]) = Q_1 + Q_2$.

For now, let us ignore the contributions of Q_2 and focus on Q_1. Then, given two disjoint paths P_1, P_2 connecting (s_1, t_1) and (s_2, t_2) respectively, consider all cycle cover containing cycles $s_1 \xrightarrow{P_1} t_1 \to s_1$ and $s_2 \xrightarrow{P_2} t_2 \to s_2$, one can see that $W(C)$ is a multiplier of $x^{|P_1|+|P_2|}$.

Therefore, consider the unique solution P_1 and P_2 to the problem, and consider the cycle cover C with cycles $s_1 \xrightarrow{P_1} t_1 \to s_1$ and $s_2 \xrightarrow{P_2} t_2 \to s_2$, and all other vertices are covered by self-loops. We have $W(C) = x^{|P_1|+|P_2|}$.

It is not hard to see that this $x^{|P_1|+|P_2|}$ is the *minimum term* in Q_1. Hence, if we can somehow subtract the annoying term Q_2 from $\text{per}(A(G)[t_1 \to s_1, t_2 \to s_2])$, and compute the polynomial Q_1, we would solve the problem.

The idea is to also consider some other types of edge-forcing. For example, we can also reverse s_2 and t_2 without changing the problem, as it is an undirected graph.

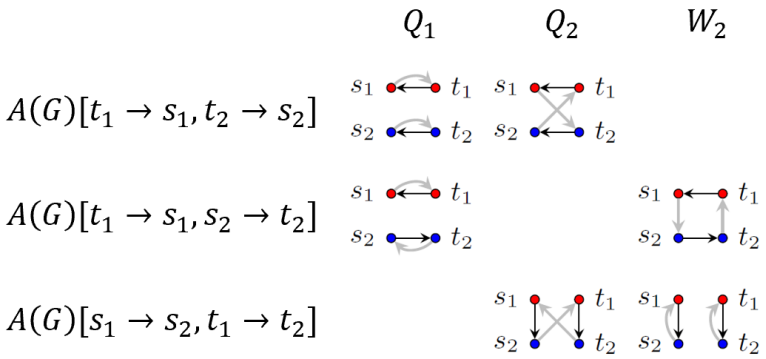

Figure 3: An illustration of the three types of edge-forcing discussed here. Solid edges denote the forced edges, while gray edges denote actual direct paths in the graph.

per$(A(G)[t_1 \to s_1, s_2 \to t_2])$**.** We have

$$\mathrm{per}(A(G)[t_1 \to s_1, s_2 \to t_2]) = \sum_{C \in cc(G), (t_1 \to s_1), (s_2 \to t_2) \in C} W[t_1 \to s_1, s_2 \to t_2](C).$$

Again, for ease of notation, we simply use $W(C)$ to denote $W[t_1 \to s_1, s_2 \to t_2](C)$. There are two types of cycle covers containing edges $(t_1 \to s_1)$ and $(s_2 \to t_1)$:

- $(t_1 \to s_1), (s_2 \to t_2)$ are in two different cycles. In this case, C contains two disjoint paths P_1 and P_2, connected by $s_1 \to t_1$ and $t_2 \to s_2$ respectively.
- $(t_1 \to s_1), (s_2 \to t_2)$ are in the same cycle. In this case, C contains a big cycle which passes t_1, s_1, s_2, t_2 in order, which corresponds to two paths $s_1 \to s_2$ and $t_1 \to t_2$ in the graph G.

Let W_1, W_2 be the corresponding sum of $W(C)$'s of cycle covers of case 1 and case 2. We have $\mathrm{per}(A(G)[t_1 \to s_1, s_2 \to t_2]) = W_1 + W_2$.

Note that since the graph is undirected, we have $Q_1 = W_1$. If we sum up $\mathrm{per}(A(G)[t_1 \to s_1, t_2 \to s_2])$ and $\mathrm{per}(A(G)[t_1 \to s_1, s_2 \to t_2])$, we would have $2 \cdot Q_1 + Q_2 + W_2$.

per$(A(G)[s_1 \to s_2, t_1 \to t_2])$**.** The final piece is to realize that one can indeed remove both Q_2 and W_2 simultaneously, by considering $\mathrm{per}(A(G)[s_1 \to s_2, t_1 \to t_2])$. There are two types of cycles covers containing edges $(s_1 \to s_2)$ and $(t_1 \to t_2)$:

- $(s_1 \to s_2), (t_1 \to t_2)$ are in two different cycles. In this case, C contains two disjoint paths P_1 and P_2, connected by $s_2 \to s_1$ and $t_2 \to t_1$ respectively.
- $(s_1 \to s_2), (t_1 \to t_2)$ are in the same cycle. In this case, C contains a big cycle which passes s_1, s_2, t_1, t_2 in order, which corresponds to two paths $s_2 \to t_1$ and $t_2 \to s_1$ in the graph G.

By comparing with previous discussions (and noting that the graph is undirected), one can see $\mathrm{per}(A(G)[s_1 \to s_2, t_1 \to t_2]) = Q_2 + W_2$.

Final Algorithm. Now, we have

$$2Q_1 = \mathrm{per}(A(G)[t_1 \to s_1, t_2 \to s_2]) + \mathrm{per}(A(G)[t_1 \to s_1, s_2 \to t_2]) - \mathrm{per}(A(G)[s_1 \to s_2, t_1 \to t_2]).$$

Note that the minimum degree term of $2Q_1$ would be $2 \cdot x^{OPT}$. So it suffices to compute $2Q_1$ over $Z_4[x]$. The final ingredient of [BH14] is a polynomial-time algorithm for computing the permanent in $Z_4[x]$, which completes the whole picture of the algorithm.

4 Algebrization via Dynamic Programming

4.1 A $2^k \cdot \mathrm{poly}(n)$-Time Algorithm for Shortest K-Cycle

Given an undirected graph $G = (V, E)$ and a subset $K \subseteq V$ of size k, the shortest K-Cycle problem asks to find the shortest simple cycle of G which goes through all vertices in K. In the following we present an algorithm for solving this problem in $2^k \cdot \mathrm{poly}(n)$ time.

For each edge $e \in E$, we assign a new variable x_e. We fix a special vertex $s \in K$ as the starting vertex of the cycle. A closed walk w of length ℓ is defined to be a sequence of adjacent vertices $v_0, v_1, v_2, \ldots, v_\ell = v_0$, and we define $W(w) = \prod_{i=0}^{\ell-1} x_{(v_i, v_{i+1})}$.

We fix q to be a big power of 2 (q is (say) the first power of 2 which is bigger than n^{20}), and will be working with the field $\mathbb{F}_q$.

Now, note that a walk w may not be the shortest K-cycle as it could visit the same vertex several times (that is, it is not a simple cycle). If we try to solve this problem via dynamic programming, it is not affordable

to remember all previous visited nodes, as there are 2^n possible states. But still, we could try to add some easy-to-check condition for a walk w.

We say a walk $w = v_0, v_1, v_2, \ldots, v_\ell = v_0$ is a good walk, if all the following conditions are satisfied:

- It starts with s, and $v_1 < v_{\ell-1}$. (Simply a tie breaker, in this way we don't count the same cycle two times.)
- For all i such that $v_i \in K$, we have $v_{i-1} \neq v_{i+1}$ (we set $v_{-1} = v_{\ell-1}$ and $v_{\ell+1} = v_1$). (At least don't revisit a vertex immediately when we leave a vertex from K.)
- All vertices in K are visited *exactly once*.

We use K-cycle to denote a simple cycle passing through all vertices in K. Clearly, a K-cycle is a good walk. But a good walk may not be a K-cycle, as it can still visit some other vertices multiple times. Let W_ℓ be the set of all good walks of length ℓ, and

$$P_\ell := \sum_{w \in W_\ell} W(w).$$

We claim that, the shortest length of a K-cycle is simply the minimum ℓ such that P_ℓ is a nonzero polynomial!

First, suppose there is a K-cycle C of length ℓ, it is not hard to see P_ℓ is nonzero. As it must contain the term $W(C)$ which cannot be canceled by all other $W(w)$'s. (Indeed, if $W(w) = W(C)$, since C is a simple cycle, it implies $w = C$.)

So it suffices to show that if there is no K-cycle of length at most ℓ in the graph, $P_\ell = 0$. To establish this, we can try to pair up all walks in W_ℓ so that the two walks in the pair have the same $W(w)$, and therefore cancel each other.

Let $w \in W_\ell$. Since $w = v_0, v_1, v_2, \ldots, v_\ell$ is not a K-cycle, it must be the case that w visits a vertex $\notin K$ more than one time. Let u be such a vertex which appearing the earliest on the walk, and let v_L and v_R be the first and last occurrence of u. There are two cases:

1. First, if the segment $v_L, v_{L+1}, \ldots, v_R$ is not a *palindrome* (that is, $v_{L\ldots R} \neq v_R, v_{R-1}, \ldots, v_L$), we simply reverse this segment, get another walk $\widetilde{w}$, and stop.
2. Second, if the segment $v_{L\ldots R}$ is a palindrome. We remove the vertices $v_{L+1}, \ldots, v_R$ from w and obtain a shorter walk w'. Note that since $v_{L\ldots R}$ is a palindrome, it can only be of odd length (an even palindrome visits the middle vertex two times in a row, and our graph doesn't contain self-loops). Moreover, $v_{L\ldots R}$ cannot contain any vertex $k \in K$, as if k is not the center of the palindrome, it would be visited twice; otherwise if it is the center of the palindrome, the two adjacent vertices of k would be the same; both cases contradict the definition of a good walk. Hence, one can see that w' is still a good walk, but with shorter length. We can then set $w = w'$ and start again.

Since w is always a good walk, we will always be able to find the first repeated vertex $u \notin K$. Note that at some stage we will eventually fall into case (1). After we get $\widetilde{w}$, we can add back all deleted segments in previous stages one by one, in the order from the last one to the first one. In this way, we have paired the original w with another walk $\widetilde{w}$ such that $W(w) = W(\widetilde{w})$. It is a bit tedious to rigorously argue, but one can see that the above process indeed also pair $\widetilde{w}$ to w.

Putting everything together, we know that since all walks in W_ℓ pair up as discussed above, we have $P_\ell = 0$ in this case. Which completes the proof of the claim.

Evaluating P_ℓ via Dynamic Programming. Given the above claim, the algorithm is rather simple. It simply enumerates ℓ from 0 to n, and check whether $P_\ell \neq 0$ by assigning random values to all the variables x_e's. It then just outputs the first ℓ such that $P_\ell \neq 0$.

So it remains to show how to evaluate P_ℓ, given an assignment $x_e = \alpha_e$ for all $e \in E$.

We can do this by a dynamic programming. Let $f(v_1, v_{\mathsf{last}}, v_{\mathsf{second-last}}, K_{\mathsf{visited}}, \ell)$ be the sum of the weights[2] of all length-ℓ path $v_0, v_1, \ldots, v_\ell$ such that:

- It starts with s, the second vertex is v_1; $v_{\ell-1} = v_{\mathsf{second-last}}$ and $v_\ell = v_{\mathsf{last}}$.
- For all i such that $v_i \in K$, we have $v_{i-1} \neq v_{i+1}$ (we set $v_{-1} = v_{\ell-1}$ and $v_{\ell+1} = v_1$).
- All vertices in K_{visited} are visited *exactly once*, and all vertices in $K \setminus K_{\mathsf{visited}}$ are not visited.

We omit the details here, one can calculate f on all valid inputs in $2^k \cdot \mathrm{poly}(n)$ via a standard dynamic programming approach. Then one can compute P_ℓ as

$$\sum_{v_1 < v_{\ell-1} \in N(s)} f(v_1, s, v_{\ell-1}, K, \ell),$$

where $N(s)$ is the set of all neighbors of s.

4.2 A $2^k \cdot \mathrm{poly}(n)$-time algorithm for k-Path

Given an directed graph $G = (V, E)$ and an integer k. The k-path problem asks to find a simple path of G of length k. In the following we describe an algorithm with running time $2^k \cdot \mathrm{poly}(n)$. Here we present a slightly different algorithm by Josh Alamn [Ala19], which is also based on determinant.

Now, for each vertex u in G, we assign $2\cdot(k+2)$ different variables $x_{u,1}, x_{u,2}, \ldots, x_{u,k+2}$, and $y_{u,1}, y_{u,2}, \ldots, y_{u,k+2}$.

For a walk $w = v_0 \to v_1 \to v_2 \to \ldots v_{k-1} \to v_k$ of length k (since we are looking for a k-path, we also require $v_0 \neq v_k$), we define a matrix $A(w)$ of size $(k+2) \times (k+2)$ such that for $i \in [k+1]$ and $j \in [k+2]$,

$$A(w)_{i,j} := x_{v_{i-1},j},$$

and

$$A(w)_{k+2,j} := y_{v_0,j}.$$

That is, the i-th row of $A(w)$ is filled with the x variables on the vertex v_{i-1}. And the last row is filled with the y variables on the vertex v_0. We remark the intention of filling the last row this way is to let $A(w)$ "know" that it starts from v_0, and therefore makes $A(w)$ and $A(\widetilde{w})$ different, where $\widetilde{w}$ is the reversal of the walk w.

And we define

$$W(w) = \mathrm{per}(A(w)).$$

Again, we will work in the field $\mathbb{F}_q$ for a sufficiently large q which is a power of 2. Therefore we also have $W(w) = \det(A(w))$.

Now, the key observation is that, if w visits a vertex multiple times, it means some rows of $A(w)$ are exactly the same, which implies $W(w) = \det(A(w)) = 0$.

Also, if all variables in w are distinct, one can see that $W(w) = \mathrm{per}(A(w))$ is a nonzero polynomial.

Let W_k be the set of all length-k walks. Consider the following polynomial

$$P_k := \sum_{w \in W_k} W(w).$$

We claim that there is a k-path in the graph, if and only if $P_k \neq 0$.

First, if there is no k-path in the graph. Which means for all length-k walk w, it must visit some vertex at least twice. By previous discussions, we know that $W(w) = 0$ in this case, and therefore $P_k = 0$ as well.

Second, if there is a k-path $w = v_0 \to v_1 \to v_2 \to \ldots v_{k-1} \to v_k$ (recall we require $v_0 \neq v_k$) in the graph. Consider a term in $W(w) = \mathrm{per}(A(w))$, it contains x-variables from all vertices visited, and a y-variable from v_0. It is easy to see that this term cannot be canceled by any terms in $W(w')$ where $w' \neq w$. Therefore, we can see $P_k \neq 0$.

[2] the weight is define similarly as $\prod_{i=0}^{\ell-1} x_{(v_i, v_{i+1})}$

Evaluation of P_k via Dynamic Programming. Finally, we need to show how to evaluate P_k on a random assignments to all the variables.

We resort to Ryser's formula to compute $\mathrm{per}(A(w))$. We have

$$\begin{aligned}\mathrm{per}(A(w)) :=& (-1)^{k+2} \cdot \sum_{S\subseteq[k+2]} (-1)^{|S|} \prod_{i=1}^{k+2} \left(\sum_{j\in S} A(w)_{i,j}\right) \\ =& \sum_{S\subseteq[k+2]} \prod_{i=1}^{k+2} \left(\sum_{j\in S} A(w)_{i,j}\right).\end{aligned}$$

The last equality holds because we are working over $\mathbb{F}_q$ where q is a power of 2.

Let $W_S(w) := \prod_{i=1}^{k+2} \left(\sum_{j\in S} A(w)_{i,j}\right)$, we have

$$P_k = \sum_{S\subseteq[k+2]} \sum_{w\in W_k} W_S(w).$$

So it suffices to compute

$$P_{S,k} = \sum_{w\in W_k} W_S(w).$$

The above can be computed via a straightforward $\mathrm{poly}(n)$-time dynamic programming. Therefore, the whole evaluation of P_k on the random assignments takes $2^k \cdot \mathrm{poly}(n)$ time, which completes the description of the algorithm.

References

[Ala19] Josh Alamn. personal communication, 2019.

[Bel62] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[BH13] Andreas Björklund and Thore Husfeldt. The parity of directed hamiltonian cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735, 2013.

[BH14] Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 211–222, 2014.

[BHT12] Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1747–1753, 2012.

[Bjö14] Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014.

[HK62] Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[Kou08] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 575–586, 2008.

[Wil09] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.