Improved Distance Queries and Cycle Counting by Frobenius Normal Form

Piotr Sankowski*[1] and Karol Węgrzycki†[2]

1 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
`sank@mimuw.edu.pl`

2 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
`k.wegrzycki@mimuw.edu.pl`

Abstract

Consider an unweighted, directed graph G with the diameter D. In this paper, we introduce the framework for counting cycles and walks of given length in matrix multiplication time $\widetilde{O}(n^\omega)$. The framework is based on the fast decomposition into Frobenius normal form and the Hankel matrix-vector multiplication. It allows us to solve the following problems efficiently:

- All Nodes Shortest Cycles – for every node return the length of the shortest cycle containing it. We give an $\widetilde{O}(n^\omega)$ algorithm that improves Yuster [30] $\widetilde{O}(n^{(\omega+3)/2})$ algorithm for unweighted digraphs.
- We show how to compute all D sets of vertices lying on cycles of length $c \in \{1, \dots, D\}$ in time $\widetilde{O}(n^\omega)$ randomized time. It improves upon [9] where algorithm that computes a single set is presented.
- We present a functional improvement of distance queries [32] for directed, unweighted graphs.
- All Pairs All Walks – we show almost optimal $\tilde{O}(n^3)$ time algorithm for all walks counting problem. We improve upon the naive $O(Dn^\omega)$ time algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Frobenius Normal Form, Graph Algorithms, All Nodes Shortest Cycles

Digital Object Identifier 10.4230/LIPIcs.STACS.2017.56

1 Introduction

The *All-Pairs Shortest Paths* (APSP) problem asks to find distances between all pairs of vertices in a graph. For a directed graphs with weights in $\mathbb{R}$, there is a classical $O(n^3)$ time algorithm Floyd and Warshall [12, 28]. Currently best upper bound for this problem is due to Williams [29] $O(\frac{n^3}{2^{\Omega(\log n)^{0.5}}})$ algorithm. It is asymptotically faster than $O(n^3/\log^c n)$ for any $c > 0$ (see survey [6] for earlier algorithms). Showing any algorithm that would work in $O(n^{3-\epsilon})$ time for some $\epsilon > 0$ is a major open problem [29].

If we consider unweighted, directed graphs there are subcubic algorithms that exploit fast matrix multiplication. For the undirected graph Seidel [22] presented the optimal $\widetilde{O}(n^\omega)$ time algorithm, where $\omega < 2.373$ is the matrix multiplication exponent [17]. For the directed case Zwick [33] presented an $O(n^{2.575})$ time algorithm that is based on the fast rectangular

* Piotr Sankowski is supported by the Polish National Science Center, grant no. UMO-2014/13/B/ST6/01811.

† The work of Karol Węgrzycki is part of a project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 677651).

matrix multiplication. Moreover, if we are interested in small integer weights from the set $\{-M, \ldots, M\}$ we have $O(M^{0.68} n^{2.575})$ algorithm [33].

Because APSP in undirected graphs can be solved in $\widetilde{O}(n^\omega)$, diameter, radius, shortest cycle, etc. can be determined in $\widetilde{O}(n^\omega)$ time as well. It is surprising that for a directed case, where merely $O(n^{2.575})$ APSP is known there are also $\widetilde{O}(n^\omega)$ algorithms for determining these properties. After a long line of improvements Cygan et al. [9] showed an $\widetilde{O}(Mn^\omega)$ time algorithms for finding minimum weight perfect matching, shortest cycle, diameter and radius. Also, they showed an application of their techniques that improves upon Yuster [30] $\widetilde{O}(Mn^\omega t)$ time algorithm for the following problem: *determine the set of vertices that lie on some cycle of length at most t.* Cygan et al. [9] managed to solve this problem in $\widetilde{O}(Mn^\omega)$ time using Baur-Strassen's theorem.

All of these algorithms are effective only in the case of a dense graphs. For graphs with the small number of edges there are better algorithms (e.g., APSP in $\tilde{O}(|V||E|)$ time [26]). But these algorithms are $\Theta(n^3)$ when $|E| = \Theta(n^2)$.

2 Related Work

2.1 Distance Queries

Yuster and Zwick [32] considered the weighted, directed graphs with weights in $\{-M, \ldots, M\}$. They showed an algorithm that needs $\widetilde{O}(Mn^\omega)$ preprocessing time. After preprocessing each distance $\delta(u, v)$ in the graph can be computed exactly in $O(n)$ query time. In the special case $M = 1$ they showed $\widetilde{O}(n^\omega)$ algorithm that solves *Single Source Shortest Paths* (SSSP).

We will match their bounds (up to the polylogarithmic factors) using Frobenius normal form. Next we will extend their algorithm so it will return more information about a graph in the same query/preprocessing time.

2.2 Counting Cycles

For a given graph G determining whether G contains a simple cycle of length exactly k is NP-hard (in particular determining whether a graph contains a Hamiltonian cycle is NP-complete). However, if we fix k to be a constant this problem can be solved in polynomial time.

Alon et al. [4] introduced a color coding technique. For a fixed k if a graph $G(V, E)$ contains a simple cycle of size exactly k then such cycle can be found in $\widetilde{O}(|V|^\omega)$ time. Unfortunately, their algorithm depends exponentially $2^{O(k)}$ on the length of the cycle and in consequence is inapplicable for large k. In the next years, Alon et al. [5] showed (using a different technique) that for $k \leq 7$ it is possible to count the number of cycles of length exactly k in a graph in $\widetilde{O}(|V|^\omega)$ time. In [31] it is shown that for any even k, cycles of length k can be found in $O(|V|^2)$ time in undirected graphs (if they contain such a cycle). Alon et al. [5] showed more methods that depend solely on a number of edges in a graph. For example for odd k they showed $O(E^{2-\frac{2}{k+1}})$ algorithm for finding a cycles of length k. However, for dense graphs these results are worse than Alon et al. [4].

It appears that to break the exponential dependence on the length of the cycle we can do one of the following:

- Consider non-simple cycles (the vertices can reoccur) of length exactly k,
- Determine cycles of length at most k.

To detect whether a non-simple cycle of length exactly k exists one can use the folklore algorithm. It starts by taking the adjacency matrix A of a graph G. Subsequently, in $\widetilde{O}(n^\omega)$

time compute A^k by repeated squaring. If $\mathrm{Tr}\left[A^k\right] > 0$ then a non-simple k length cycle exists[1].

Yuster [30] considered the following problem: *for every vertex in a graph find a shortest cycle that contains it.* He called this problem *All-Nodes Shortest Cycle* (ANSC). He showed a randomized algorithm that solves ANSC for undirected graphs with weights $\{1,\ldots,M\}$ in $\widetilde{O}(\sqrt{M}n^{(\omega+3)/2})$ time. He noted that for simple digraphs (directed graphs with no anti-parallel edges) it is reduced to All-Pairs Shortest Paths problem. The fastest known APSP algorithm for unweighted, directed graphs runs in $O(n^{2.575})$ due to [33]. Here, we will show how to solve ANSC in $\widetilde{O}(n^\omega)$ for general, unweighted, directed graphs. Unfortunately, our techniques will allow us only to find the length of such a cycle (not determining it). But we can return the set of points, that lie on some cycle of a given length. Independently to our work Agarwal and Ramachandran [3] proved that ANSC can be solved in $\widetilde{O}(n^\omega)$ for unweighted, undirected graphs using a completely different technique.

Yuster [30] also considered following problem: *given a graph and an integer t. Let $S(k)$ denote the set of all vertices lying in a cycle of length $\leq k$. Determine $S(t)$.* He considered directed graphs with weights in $\{-M,\ldots,M\}$ and showed $\widetilde{O}(Mn^\omega t)$ algorithm

Recently, Cygan et al. [9] improved his algorithm. They showed that for a fixed $t \in [0, nM]$ the set $S(t)$ can be computed in $\widetilde{O}(Mn^\omega)$ randomized time. We show, that for an unweighted $(M = 1)$ directed graphs we can compute sets $S(1), S(2), \ldots, S(D)$ in $\widetilde{O}(n^\omega)$ time with high probability.

3 Preliminaries

Let $T(n)$ be the minimal number of algebraic operations needed to compute the product of $n \times n$ matrix by an $n \times n$ matrix. We say that ω is the exponent of square matrix multiplication. For now the best known upper bound on ω is due to Le Gall [17]:

▶ **Theorem 1** (Le Gall [17]). *For every $\epsilon > 0$, $T(n) < O(n^{\omega+\epsilon})$ where $\omega < 2.37287$.*

In this paper, we will omit ϵ in definition and will assume (like in many other papers) that $O(n^\omega)$ operations are needed to multiply two matrices. The best lower bound for the exponent of matrix multiplication is $\omega \geq 2$. For convenience in this paper we will assume that $\omega > 2$. The $\tilde{O}$ notation hides polylogarithmic factors in the complexity. We will use it to emphasize that all our algorithms need the polylogarithmic number of calls to the fast matrix multiplication algorithm.

▶ **Theorem 2** (Storjohann [24]). *The Frobenius canonical-form of a matrix can be computed deterministically using $\widetilde{O}(n^\omega)$ field operations.*

The comprehensive description of Frobenius normal form will be presented in Section 4. The properties of Frobenius normal form used in this paper are well known in literature [24, 23, 10]. There are also probabilistic algorithms that compute this form in expected $\widetilde{O}(n^\omega)$ with high probability over small fields [11]. In this paper, all algorithms are deterministic if we set the upper bound on the number of distinct walks W. Then, due to the time of a single field operation we need additional $O(\log W)$ factor in the complexity. However, if we are only interested if cycle/walk of a given length exists in a graph, we can set sufficiently small field $\mathbb{Z}_p$ (p has $O(\log n)$ bits). This way when algorithm returns nonzero we are sure

[1] $\mathrm{Tr}\,[A]$ denotes the trace of a matrix A, i.e., the sum of elements on main diagonal.

that some walk exists there. If algorithm returns zero, then with high probability there is no such walk.

In this paper, randomization occurs only because for some graphs number of walks can be exponential (e.g., $W = O(2^n)$) and the output requires to return them.

For matrices $A \in \mathbb{R}^{n\times k}$ and $B \in \mathbb{R}^{n\times l}$ the $A \oplus B \in \mathbb{R}^{n\times(k+l)}$ denotes the concatenation of their columns. $C_{a,b} \in \mathbb{R}^{n\times(b-a)}$ denotes a matrix constructed by concatenating columns $c_a, c_{a+1}, \dots, c_b$ of the matrix $C \in \mathbb{R}^{n\times m}$.

4 Consequences of Frobenius Normal Form

Let K be a commutative field. For any matrix $A \in K^{n\times n}$ there exists an invertible U over K such that:

$$U^{-1}AU = F = \begin{bmatrix} C_1 & & & & 0 \\ & C_2 & & & \\ & & C_3 & & \\ & & & \ddots & \\ 0 & & & & C_k \end{bmatrix}.$$

and F is the Frobenius-canonical-form [2] of A. The diagonal block C_i is called the companion matrix:

$$C_i = \begin{bmatrix} 0 & & \dots & & 0 & -c_0 \\ 1 & 0 & & & 0 & -c_1 \\ & 1 & \ddots & & \vdots & -c_2 \\ & & \ddots & 0 & & \vdots \\ & & & 1 & 0 & -c_{r-2} \\ 0 & & & & 1 & -c_{r-1} \end{bmatrix} \in K^{r\times r}.$$

Each companion matrix corresponds to the monic polynomial $C_i(x) = x^r + c_{r-1}x^{r-1} + \dots + c_0 \in K[x]$ and is called the *minimal polynomial* of A. Each minimal polynomial has a property that $C_i(A) = 0$. To guarantee that matrix has only one decomposition into Frobenius normal form we require that every polynomial must divide the next one, i.e., $C_i(x)|C_{i+1}(x)$. The final list of polynomials is called the *invariant factors* of matrix A [24].

4.1 Cyclic Subspaces

Frobenius decomposition can be used to get the desired power of a matrix (analogously to the diagonal decomposition):

$$A^k = (UFU^{-1})^k = UF(U^{-1}U)F \cdots F(U^{-1}U)FU^{-1} = UF^kU^{-1}.$$

Moreover, we will use the property that the power of block diagonal matrix F is block diagonal:

$$F^k = \begin{bmatrix} C_1^k & & & & 0 \\ & C_2^k & & & \\ & & C_3^k & & \\ & & & \ddots & \\ 0 & & & & C_l^k \end{bmatrix}.$$

[2] Sometimes called the rational-canonical form.

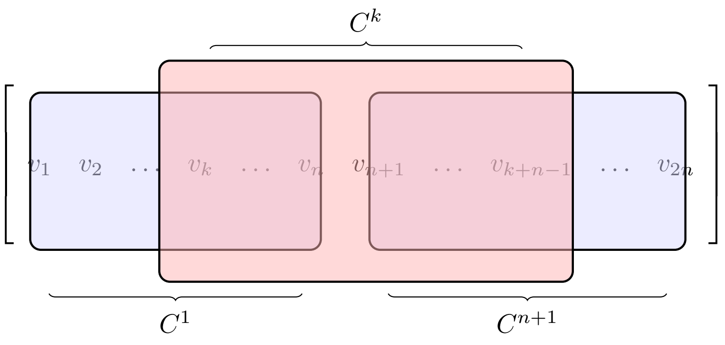

C^k
v_1 v_2 $\dots$ v_k $\dots$ v_n v_{n+1} $\dots$ v_{k+n-1} $\dots$ v_{2n}
C^1
C^{n+1}

Figure 1 Visualisation of the cyclic property (Definition 3).

Now, we need a property of companion matrices that will enable us to power them efficiently.

▶ **Definition 3** (Cyclic Property). Let $v_1, \dots, v_n$ be the columns of a matrix $C \in K^{n\times n}$. Let $v_{n+1}, \dots, v_{2n}$ be the columns of matrix C^{n+1}. If, for every $1 \le k \le n$ the columns of matrix C^k are $v_k, v_{k+1}, \dots, v_{k+n}$ then the C has a **cyclic property**.

▶ **Theorem 4** (Folklore [14], see [18] for generalization). *Every companion matrix has a cyclic property.*

5 Matching Distance Queries on Directed Unweighted Graphs

In this section, we will present a simple algorithm that matches the best known upper bounds of Yuster and Zwick [32] distance queries in directed unweighted graphs.

5.1 Answering Distance Queries by Using Frobenius Normal Form

We take the adjacency matrix A of a graph G (i.e., $n\times n$ matrix with $a_{u,v} = 1$ when $(u, v) \in G$ and 0 otherwise). The k-th power of the adjacency matrix of the graph G holds the number of walks, i.e., an $a_{u,v}$ element of A^k is *the count of distinct walks from u to v of length k in the graph*. The shortest path between vertices u, v is the smallest k such that A^k has nonzero element $a_{u,v}$. For a brief moment, we will forget about graph theory interpretation and focus only on finding such k.

We decompose matrix A into the Frobenius normal form. Storjohann [24] showed an algorithm that returns U and F deterministically in $\widetilde{O}(n^\omega)$ time (note that matrix inverse can also be computed in $\widetilde{O}(n^\omega)$ time).

5.2 Single Invariant Factor

To better explain the idea, for a start we will consider a simple case when a number of invariant factors of A is exactly 1. In that situation, the matrix F is a companion matrix $C \in K^{n\times n}$.

First, we compute the $(n + 1)$-th power of the companion matrix F^{n+1}. This can be done in $\widetilde{O}(n^\omega)$ time by repeated squaring (compute $F, F^2, F^4, \dots, F^{n+1}$ with $O(\log n)$ matrix multiplications)[3].

[3] One can compute F^{n+1} even faster. Namely, in $\tilde{O}(n^2)$ time by applying the cyclic property (see Definition 3)

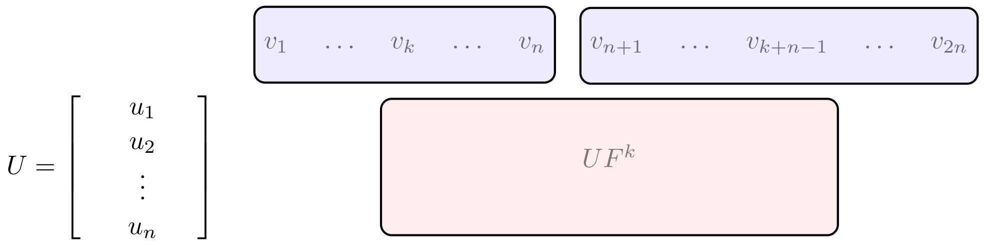

v_1 ... v_k ... v_n
v_{n+1} ... v_{k+n-1} ... v_{2n}
$U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$
UF^k

Figure 2 Construction of UF^k from matrices UF and UF^{n+1}.

If the matrix UF has columns $v_1, \ldots, v_n$ and the matrix UF^{n+1} has columns $v_{n+1}, \ldots, v_{2n}$, then the columns $v_k, \ldots, v_{k+n-1}$ construct UF^k (see Figure 2). It is because the matrix F has the cyclic property.

This step takes just 2 matrix multiplications, because we need to multiply U times F and F^{n+1}. The preprocessing phase takes only $\widetilde{O}(n^\omega)$ time.

Now, if a user wants to know the number of distinct walks from vertices u to v of length exactly k he takes:

- The u-th row of matrix UF^k (n numbers),
- The v-th column of matrix U^{-1},
- Multiplies them in $O(n)$ time (dot product of n dimensional vectors).

This will give us the u, v element of matrix $UF^kU^{-1} = A^k$. To get the length of the shortest path (i.e., the minimal k that $w_{u,v} > 0$), we will modify our matrices slightly to get the number of walks of length $\leq k$. At the end, we will fit in $\widetilde{O}(n)$ query time (by using binary search) and $\widetilde{O}(n^\omega)$ preprocessing time.

Basically, for a given k we need to get the u, v element of matrix $A + A^2 + \cdots + A^k$. It suffices to add consecutive columns of matrix $UF \oplus UF^{n+1} = v_1 \oplus v_2 \oplus \ldots \oplus v_{2n}$ in the following manner [4]:

$$M' = \left[\begin{array}{ccccccc} v_1 & v_1 + v_2 & v_1 + v_2 + v_3 & \ldots & \sum_{i=1}^{k} v_i & \ldots & \sum_{i=1}^{2n} v_i \end{array} \right] \in \mathbb{R}^{n \times 2n}.$$

Now, to get $A + A^2 + \cdots + A^k$ one needs to multiply $M'_{k,k+n-1}U^{-1}$ and subtract $M'_{1,n}U^{-1}$ for a balance [5].

We can transform matrices U and F to matrix M' in $O(n^2)$ time during preprocessing. During query, we need to compute 2 dot products (u-th row of $M'_{k,k+n-1}$ times v-th column of U^{-1} and u-th row of $M'_{1,n}$ times v-th column of U^{-1}) and subtract them.

We have an algorithm that for a given vertices $u, v \in G$ and integer $k \in \{1, \ldots, n\}$ can answer: *how many walks from u to v of length less or equal k are in the graph G* in $\widetilde{O}(n)$ query and $\widetilde{O}(n^\omega)$ preprocessing.

Because the result of the query is increasing in k we can use binary search. We can determine the first k for which the query will answer nonzero value in $O(\log n)$ tries. Hence, in $\widetilde{O}(n)$ we can find the length of the shortest path. This generalized query can easily return the number of walks of length exactly k, i.e., $q(u, v, k) - q(u, v, k-1)$.

For now, we only matched the result of Yuster and Zwick [32] for unweighted graphs with a single, invariant factor. In the next section, we will show how to generalize our technique for

[4] Operation $\oplus$ denotes concatenation.
[5] $X_{a,b}$ denotes a matrix constructed by concatenating columns $x_a, x_{a+1}, \ldots, x_b$ of a matrix X.

graphs with any number of invariant factors. Then, we will extend the Yuster and Zwick [32] algorithm. Namely, we will show that in $\widetilde{O}(n^{\omega})$ preprocessing time and $\widetilde{O}(n)$ query time we can get D numbers (where D is the graph diameter): $w^1_{u,v}, w^2_{u,v}, \dots, w^D_{u,v}$. The number $w^i_{u,v}$ tells how many walks of length exactly i are from vertex u to v.

With such an algorithm we can easily implement Yuster and Zwick [32] distance queries by linearly scanning (see Section 6). Thus, for the multiple invariant factors we will skip the description of algorithm that returns the number of walks of length smaller than k (the technique is the same).

5.3 Multiple Invariant Factors

Now, we will consider a case when $k \geq 1$, i.e., matrix F has multiple invariant factors. First of all, we need to note that this generalization is not perfect and will allow only the walks of length up to D (the longest distance in a graph, i.e., diameter).

In the real world applications of our framework (detecting cycles, determining distance between nodes, etc.) we do not need walks longer than the longest possible distance in a graph. It is natural that the diameter is considered to be a bound of an output in graph problems [8, 9, 1, 2].

5.3.1 Relation of the Graph Diameter and Frobenius Normal Form

We begin with relating the graph diameter to the Frobenius normal form. It turns out that the graph diameter is bounded by the degree of a smallest invariant factor.

▶ **Theorem 5** ([7]). *Given a directed, unweighted graph G with a diameter D. Let μ denote the degree of the smallest invariant factor (i.e., the dimension of the smallest block in the Frobenius matrix F) of an adjacency matrix of the graph G. Then $D \leq \mu$.*

The bounds of this inequality are tight. There are graphs with diameter $D = \mu$ and graphs with $\mu = n$ and arbitrary small diameter [7]. Our algorithms are able to return walks up to the length μ. We use the bound on D solely because it is easier to interpret diameter than the *smallest degree of the invariant factor.*

5.3.2 Generalization to Multiple Invariant Factors

Let k denote the number of blocks in the Frobenius matrix F and μ be the number of columns of the smallest block. To multiply matrix U by F we can start by multiplying strips of matrix U by appropriate blocks of F and concatenate them later (see Figure 3).

We start by splitting the matrix U into k strips with rows corresponding to the appropriate blocks of F (strip U_i has as many columns as block F_i). Then we multiply UF and have k strips: $U_1F_1, U_2F_2, \dots U_kF_k$ (each with at least μ columns). Next, we multiply UF^{μ} and we keep k strips: $U_1F_1^{\mu}, U_2F_2^{\mu}, \dots, U_kF_k^{\mu}$. Our goal is to get a data structure such that if we need UF^k, we can quickly choose appropriate columns and append them.

The matrix U_iF_i has l_i columns: $v_1, \dots, v_{l_i}$. Because F_i is a companion matrix, the $U_iF_i^{\mu}$ has the cyclic property (Definition 3). And the matrix $U_iF_i^{\mu}$ has columns: $v_{\mu}, \dots, v_{\mu+l_i}$. There are some duplicate columns in U_iF_i and $U_iF_i^{\mu}$, when $\mu < l_i$. Hence, we only need to keep columns $v_1, \dots, v_{\mu+l_i}$ for this strip. We do this for all strips (see Figure 4).

We are left with a matrix that has at most $2n$ columns (because $l_1+\mu+l_2+\mu+\dots l_k+\mu = k\mu + \sum_{i=1}^{n} l_i = n + k\mu \leq 2n$). To generate it we need to power F to μ and do multiplications $U \cdot F$ and $U \cdot F^{\mu}$. This can be computed in $\widetilde{O}(n^{\omega})$ time.

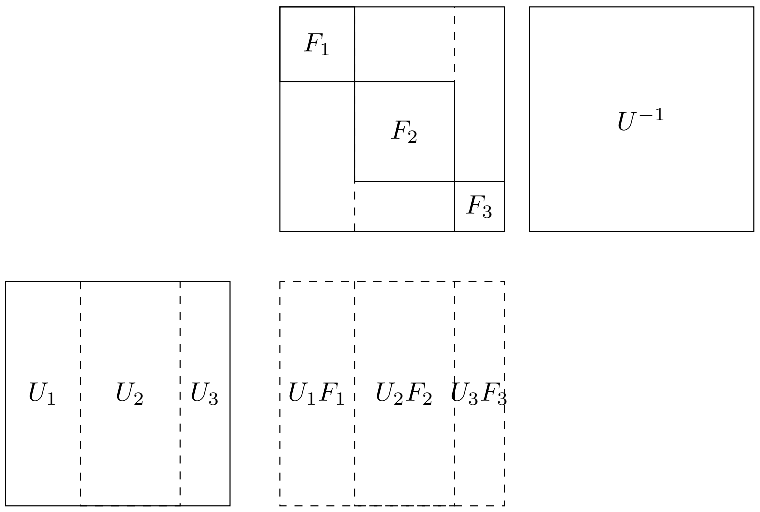


Figure 3 Multiplication of the block matrix. Example for 3 invariant factors.

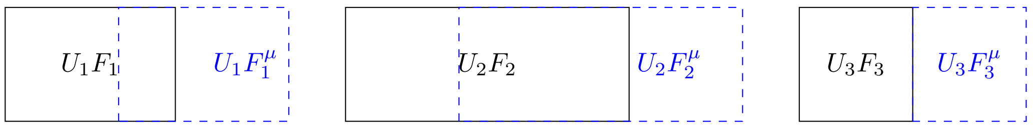


Figure 4 Combining strips into a single matrix.

5.3.3 Queries with Multiple Invariant Factors

When a query for the number of walks of length k from node u to v comes, we do:

1. For each strip i take the u-th row of $U_iF_i \oplus U_iF_i^{\mu}$ concatenate them (see Figure 5) into vector $\bar{u}$,
2. Take v-th column of U^{-1} matrix and denote it $\bar{v}$,
3. Return the dot product $\bar{u} \cdot \bar{v}$.

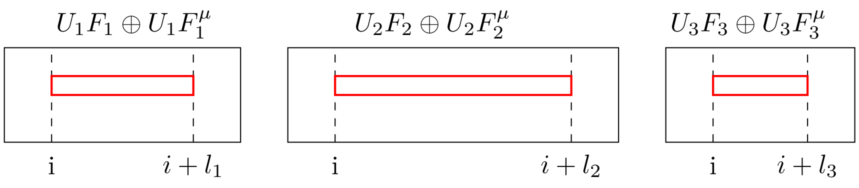
Because $l_1 + l_2 + \ldots + l_k = n$ the vector $\bar{u} \in K^n$. Query needs $O(n)$ time.

Finally, this dot product is $a_{u,v}$ element of the matrix UF^kU^{-1}, for a fixed $k \leq \mu$. Analogously to Section 5.2 one can extend this result to return the number of walks of length less or equal k. This matches (up to the polylogarithmic factor) the result of Yuster and Zwick [32]. However in the next section we will show a more general result.

6 Almost Optimal Query

In the previous section, we showed how to preprocess a directed, unweighted graph in $\widetilde{O}(n^{\omega})$ time in such a way that in $O(n)$ query we can return a number of distinct walks of a length k from vertex u to v. However, in linear time $O(n)$ we return only a single number. Our goal is to get far richer information in $\widetilde{O}(n)$ query time.

▶ **Theorem 6.** *Let $G = (V, E)$ be a directed, unweighted graph with n vertices and a diameter D. There exists an algorithm that after some preprocessing can answer queries for any given $u, v \in V$:*



$U_1F_1 \oplus U_1F_1^{\mu}$
$U_2F_2 \oplus U_2F_2^{\mu}$
$U_3F_3 \oplus U_3F_3^{\mu}$
i
$i + l_1$
i
$i + l_2$
i
$i + l_3$

Figure 5 Schema of obtaining vector $\bar{u}$ for the cycles of length i.

- *Query returns $\{w_i \mid 1 \le i \le D\}$, where w_i is the number of distinct walks from u to v of length exactly i,*
- *Preprocessing takes $\widetilde{O}(n^{\omega})$ and query takes $\widetilde{O}(n)$ field operations.*

The algorithm is deterministic (but there are $O(\log W)$ factors in derandomized version, see Section 3 for explication).

This algorithm is a significant improvement over Yuster and Zwick [32]:
- One can use Theorem 6 to find the distance between u, v by linearly scanning the array and returning the first k such that $w_k > 0$,
- Theorem 6 can count cycles. In contrast the Yuster and Zwick [32] cannot, because the distance from u to u is always 0 (we will see that in the next section),
- Theorem 6 is almost optimal (when $D = O(n)$ its output and time are $O(n)$).

On the other hand, Theorem 6 is only a functional improvement and it does not break the $\widetilde{O}(n^{\omega})$ of the *Single Source Shortest Paths* (SSSP) for dense, directed graphs.

6.1 Hankel Matrix

Now, we will focus on the proof of Theorem 6. First, we need to introduce the Hankel matrix and its properties.

$$H = \begin{pmatrix} c_1 & c_2 & \dots & c_n \\ c_2 & c_3 & \dots & c_{n+1} \\ \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & \dots & c_{2n-1} \end{pmatrix}$$

Hankel matrix is defined by its first row and last column ($2n - 1$ numbers define $n \times n$ Hankel matrix). The numbers from the previous row are left-shifted by one and the new number is added at the end. Hankel matrices have some similarities with Topelitz and Circulant matrices.

The basic property we need is that the product of Hankel matrix and vector can be computed in $O(n \log n)$ time (see [16, 25]) even though explicitly writing the Hankel matrix as $n \times n$ matrix takes $O(n^2)$ time. The algorithm takes $2n - 1$ parameters that define the Hankel matrix and n parameters that define the vector. The technique is based on the Fast Fourier Transformation [16, 25].

6.2 Using Hankel Matrices to Return Number of Walks

The algorithm from Section 5.3.3 concatenates the strips U_iF_i and builds a single vector. Subsequently, that vector is multiplied by a column of matrix U^{-1}. But we can also do it in a different order: first we multiply the strip by a section of matrix U^{-1} and sum the

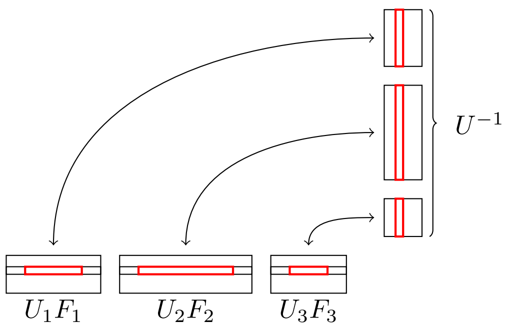


Figure 6 Multiplication of strips by U^{-1} matrix. As you can see, matrix U^{-1} can be split into sections, that multiply only U_iF_i strips.

results at the end. Thus, we perform k (number of strips) dot products of smaller vectors (see Figure 6).

Consider the query for a number of walks of length exactly k. The strips in the matrix U^{-1} do not depend on k (vector $(u_0, \ldots, u_l)$). However, the vector taken from U_iF_i (vectors $(x_i, \ldots, x_{i+l})$) will be left shifted if we want to compute the next one.

$$\begin{matrix} (x_0 & x_1 & x_2 & \ldots & x_l) \\ (x_1 & x_2 & \ldots & x_l & x_{l+1}) \\ (x_2 & \ldots & x_l & x_{l+1} & x_{l+2}) \\ \vdots & & & & \vdots \\ (x_\mu & & \ldots & & x_{\mu+l}) \end{matrix} \times \begin{pmatrix} u_0 \\ \\ \vdots \\ \\ u_l \end{pmatrix}$$

As you can see, the subsequent rows can be written as the Hankel matrix (we need to add some zeros and discard certain results to get square matrix, but it will not influence asymptotic complexity). By using the *fast Hankel matrix-vector multiplication* we can compute μ values for every strip i in time $O(l_i \log l_i)$ (l_i was defined as the length of i-th strip). At the end, we need to sum all results into a single array. The total complexity is $O(\sum_{i=1}^{k} l_i \log l_i))$. Because $\sum_{i=1}^{k} l_i = n$ the algorithm needs $O(n \log n)$ field operations. It proves Theorem 6.

We need to address some issues regarding field operations. As mentioned in the Section 3, the $\tilde{O}$ notation hides the polylogarithmic factors. Here, we silently assume that the number of walks is bounded by W. It means that the complexity is multiplied by $O(\log W)$ factor because of the cost of arithmetic operations. If the number of walks is exponential in n then the cost increases by a linear factor. However, at the beginning we could randomly select the prime number p with $O(\log n)$ bits and do arithmetic operations in the field $\mathbb{Z}_p$. In some applications we can use our algorithm to answer whether with high probability there exists a walk of a given length. The problem of large number of distinct walks is more of a theoretical issue than a practical one.

6.3 All Pairs All Walks (APAW)

Now we will show the application of Theorem 6. We begin with almost optimal algorithm to compute the number of all walks between all pairs of vertices. We are not aware of any other research concerning this problem.

▶ **Definition 7** (All Pairs All Walks problem). Given a directed, unweighted graph G with a diameter D. The task is to return an array A, such that for every pair of vertices $u, v \in G$ and every $k \in \{1, \ldots, D\}$ an element $A[u, v, k]$ is the number of distinct walks from u to v of length k.

The only solution to this problem we are aware of needs $O(Dn^\omega)$ time. The naive approach: *take the adjacency matrix A of graph G and save it in $A[u, v, 1]$. Then, square it to get A^2 and save it in $A[u, v, 2]$. Continue until you fill out complete table.* In the worst case this algorithm needs $D = O(n)$ matrix multiplications, thus its complexity is $O(Dn^\omega)$. At the first glance it is surprising that we can improve it to $\widetilde{O}(n^3)$ especially when $\omega > 2$.

The $\tilde{O}(n^3)$ algorithm works as follows. First, preprocess the algorithm from Theorem 6 which takes $\widetilde{O}(n^\omega)$ time. Then, for every pair of vertices u, v ask a query. A single query takes $\tilde{O}(n)$ time. Then, save it in the table $A[u, v]$ (query gives D numbers $w_1, \dots, w_D$, such that w_i is the number of walks of length i and save it $A[u, v, i] := w_i$).

Because there are $O(n^2)$ pairs and for each pair we need $\tilde{O}(n)$ time, the complexity of our solution is $\tilde{O}(n^3)$. The algorithm is almost optimal because the output in the worst case may be $O(n^3)$.

7 Counting and Determining the Lengths of Cycles

We will use Theorem 6 to solve All-Nodes Shortest Cycles (ANSC) problem efficiently.

▶ **Theorem 8.** *There exists an algorithm that for a given unweighted digraph G with a diameter D:*

- *For every vertex u returns D numbers: $c_u^1, c_u^2, \dots c_u^D$,*
- *The c_u^k is the number of non-simple cycles of length exactly k, that contain vertex u,*
- *Algorithm works in $\widetilde{O}(n^\omega \log W)$ time (where W is the maximum c_u^k).*

Proof. We will use Theorem 6. We start by preprocessing the graph G in time $\widetilde{O}(n^\omega)$. Theorem 6 allows us to ask for a number of walks from u to v and receive D numbers: $w_{u,v}^k$. So, we ask for the number of walks from vertex u to the same vertex u. This is exactly the number of non-simple cycles of a given length that contain vertex u.

Because we need to ask only n queries (it is the number of vertices in a graph) and each query takes $\tilde{O}(n)$ time we have $\widetilde{O}(n^\omega + n^2) = \widetilde{O}(n^\omega)$ algorithm. ◀

7.1 Solving ANSC Faster

To solve ANSC problem in $\widetilde{O}(n^\omega)$ time and beat Yuster [30] $\widetilde{O}(n^{(\omega+3)/2})$ algorithm we do the linear search on the output. For every vertex we search for the first nonzero element linearly. This with high probability is the length of the shortest cycle that contains it. Because the output contains $O(n^2)$ numbers the complexity is equal to the preprocessing time $\widetilde{O}(n^\omega)$.

Similarly, we can scan the output to compute the set $S(c)$ that contains all vertices that lie on some cycle of length $\leq c$. Then, by linear scan we can return the sets $S(1), \dots, S(D)$. This improves upon Cygan et al. [9].

8 Conclusion and Open Problems

We introduced the framework based on Frobenius normal form and used it to solve problems on directed, unweighted graphs in matrix multiplication time. The main open question is to use this framework to prove that APSP on such graphs can be solved in $\widetilde{O}(n^\omega)$ or at least $O(n^{2.5})$. The promising way is to use the algorithms that determine operators of matrices of polynomials (e.g., determinant, solving linear system [19, 15]). Additionally, algorithms for a black-box polynomial degree determination seem to be a promising path.

Another interesting problem is to use this framework to obtain additive approximation for APSP. Currently, the best additive approximation of APSP is due to [21]. However, none of known additive approximation of APSP works in $\widetilde{O}(n^{\omega})$ time.

Application in dynamic algorithm also seems to be a promising approach. Frandsen and Sankowski [13] showed an algorithm, that dynamically preserves Frobenius normal form in $O(kn^2)$ time. Our algorithms use fast Hankel matrix-vector multiplication. The algorithm behind fast Hankel matrix-vector multiplication is based on Discrete Fourier Transform (DFT). Reif and Tate [20] presented an $O(\sqrt{n})$ time per request algorithm for DFT. Can we use those approaches to obtain a faster dynamic algorithm?

Finally, it remains open how to apply the Frobenius normal form in the weighted directed graphs with small, integer weights $\{-M, \ldots, M\}$. Cygan et al. [9] took degree M polynomials and used [19] to get radius and diameter in $\widetilde{O}(Mn^{\omega})$ time. We suspect that similar technique can be applied to Frobenius normal form framework.

Acknowledgment. We would also like to thank Agnieszka Ciepielewska for meaningful discussions and useful comments.

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697. SIAM, 2015. doi:10.1137/1.9781611973730.112.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

3 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity and conditional hardness for sparse graphs. *arXiv preprint arXiv:1611.07008*, November 2016.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

6 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 590–598. ACM, 2007. doi:10.1145/1250790.1250877.

7 Fan R. K. Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

8 Fan R. K. Chung, V. Faber, and Thomas A. Manteuffel. An upper bound on the diameter of a graph from eigenvalues associated with its laplacian. *SIAM J. Discrete Math.*, 7(3):443–457, 1994. doi:10.1137/S0895480191217776.

9 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28, 2015. doi:10.1145/2736283.

10 David Steven Dummit and Richard M. Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

11 Wayne Eberly. Black box frobenius decompositions over small fields. In Traverso [27], pages 106–113. URL: `http://dl.acm.org/citation.cfm?id=345542`, `doi:10.1145/345542.345596`.

12 Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962. `doi:10.1145/367766.368168`.

13 Gudmund Skovbjerg Frandsen and Piotr Sankowski. Dynamic normal forms and dynamic characteristic polynomial. In *International Colloquium on Automata, Languages, and Programming*, pages 434–446. Springer Berlin Heidelberg, 2008.

14 F. R. Gantmacher. *The Theory of Matrices, Vol. 1.* Chelsea, 1959.

15 Mark Giesbrecht, Michael J. Jacobson Jr., and Arne Storjohann. Algorithms for large integer matrix problems. In Serdar Boztas and Igor E. Shparlinski, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 14th International Symposium, AAECC-14, Melbourne, Australia November 26-30, 2001, Proceedings*, volume 2227 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2001. `doi:10.1007/3-540-45624-4_31`.

16 Gene H. Golub and Charles F. Van Loan. *Matrix computations (3. ed.).* Johns Hopkins University Press, 1996.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014.

18 Arthur Lim and Jialing Dai. On product of companion matrices. *Linear Algebra and its Applications*, 435(11):2921–2935, 2011.

19 Thom Mulders and Arne Storjohann. Rational solutions of singular linear systems. In Traverso [27], pages 242–249. URL: `http://dl.acm.org/citation.cfm?id=345542`, `doi:10.1145/345542.345644`.

20 John H. Reif and Stephen R. Tate. On dynamic algorithms for algebraic problems. *J. Algorithms*, 22(2):347–371, 1997. `doi:10.1006/jagm.1995.0807`.

21 Liam Roditty and Asaf Shapira. All-pairs shortest paths with a sublinear additive error. *ACM Trans. Algorithms*, 7(4):45:1–45:12, September 2011. `doi:10.1145/2000807.2000813`.

22 Raimund Seidel. On the all-pairs-shortest-path problem. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 745–749. ACM, 1992. `doi:10.1145/129712.129784`.

23 Arne Storjohann. An $O(n^3)$ algorithm for the frobenius normal form. In Volker Weispfenning and Barry M. Trager, editors, *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, ISSAC'98, Rostock, Germany, August 13-15, 1998*, pages 101–105. ACM, 1998. URL: `http://dl.acm.org/citation.cfm?id=281508`, `doi:10.1145/281508.281570`.

24 Arne Storjohann. Deterministic computation of the frobenius form. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 368–377. IEEE Computer Society, 2001. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7601`, `doi:10.1109/SFCS.2001.959911`.

25 Zhihui Tang, Ramani Duraiswami, and Nail A. Gumerov. Fast Algorithms to Compute Matrix-Vector Products for Pascal Matrices. *Technical Reports from UMIACS UMIACS-TR-2004-08*, 2004/03/25/ 2004. URL: `http://drum.lib.umd.edu/handle/1903/1338`.

26 Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999. `doi:10.1145/316542.316548`.

27 Carlo Traverso, editor. *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, ISSAC 2000, St. Andrews, United Kingdom, August 6-10, 2000*. ACM, 2000. URL: `http://dl.acm.org/citation.cfm?id=345542`.

28 Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962. `doi:10.1145/321105.321107`.

29 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 664–673. ACM, 2014. URL: `http://dl.acm.org/citation.cfm?id=2591796`, `doi:10.1145/2591796.2591811`.

30 Raphael Yuster. A shortest cycle for each vertex of a graph. *Inf. Process. Lett.*, 111(21-22):1057–1061, 2011. `doi:10.1016/j.ipl.2011.07.019`.

31 Raphael Yuster and Uri Zwick. Finding even cycles even faster. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, volume 820 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 1994. `doi:10.1007/3-540-58201-0_96`.

32 Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 389–396. IEEE Computer Society, 2005. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10244`, `doi:10.1109/SFCS.2005.20`.

33 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. `doi:10.1145/567112.567114`.