A Faster Deterministic Algorithm for Minimum Cycle Basis in Directed Graphs

Ramesh Hariharan* Telikepalli Kavitha† Kurt Mehlhorn‡

Abstract

We consider the problem of computing a minimum cycle basis in a directed graph. The input to this problem is a directed graph G whose edges have non-negative weights. A cycle in this graph is actually a cycle in the underlying undirected graph with edges traversable in both directions. A $\{-1,0,1\}$ edge incidence vector is associated with each cycle: edges traversed by the cycle in the right direction get 1 and edges traversed in the opposite direction get -1. The vector space over $\mathbb{Q}$ generated by these vectors is the cycle space of G. A set of cycles is called a cycle basis of G if it forms a basis for this vector space. We seek a cycle basis where the sum of weights of the cycles is minimum.

The current fastest algorithm for computing a minimum cycle basis in a directed graph with m edges and n vertices runs in $\tilde{O}(m^{\omega+1}n)$ time (where $\omega < 2.376$ is the exponent of matrix multiplication). Here we present an $O(m^3n + m^2n^2\log n)$ algorithm. This algorithm is obtained by using fast matrix multiplication over rings and an efficient extension of Dijkstra's algorithm to compute a shortest cycle in G whose dot product with a function on its edge set is non-zero.

*Indian Institute of Science, Bangalore. Email: ramesh@csa.iisc.ernet.in

†Indian Institute of Science, Bangalore. Email: kavitha@csa.iisc.ernet.in. This work was done while visiting the Max-Planck-Institut für Informatik, Saarbrücken.

‡Max-Planck-Institut für Informatik, Saarbrücken. Email: mehlhorn@mpi-sb.mpg.de

1 Introduction

We consider the problem of computing a minimum cycle basis in a directed graph whose edges have non-negative weights. A cycle in this graph is actually a cycle in the underlying undirected graph, i.e., edges are traversable in both directions. A $\{-1,0,1\}$ edge incidence vector is associated with each cycle: edges traversed by the cycle in the right direction get 1, edges traversed in the opposite direction get -1, and edges not in the cycle at all get 0. The vector space over $\mathbb{Q}$ generated by these vectors is the *cycle space* of G. A set of cycles forms a *cycle basis* if it forms a basis for this vector space. The cycle space has dimension $d = m-n+1$ for a connected graph, where m is the number of edges and n the number of vertices. We seek a *minimum cycle basis*, i.e., a cycle basis where the sum of weights of the cycles is as small as possible. Books by Deo [6] and Bollobás [3] have an in-depth coverage of cycle bases.

A related problem pertains to undirected graphs, where we associate a $\{0,1\}$ edge incidence vector with each cycle; edges in the cycle get 1 and others get 0. Unlike directed graphs where the cycle space is defined over $\mathbb{Q}$, cycle spaces in undirected graphs are defined as vector spaces over $\mathbb{F}_2$. Interestingly, transforming cycles in a directed cycle basis by replacing both -1 and 1 by 1 does not necessarily yield a basis for the underlying undirected graph. In addition, lifting a minimum cycle basis of the underlying undirected graph by putting back directions does not necessarily yield a *minimum* cycle basis for the directed graph. We provide examples of both phenomena, presented in [16], in the Appendix. Thus, one cannot find a minimum cycle basis for a directed graph by simply working with the underlying undirected graph.

Applications. Apart from its interest as a natural question, the minimum cycle basis problem is motivated by its use as a preprocessing step in several algorithms. That is, a cycle basis is used as an input for a later algorithm, and using a minimum cycle basis instead of any arbitrary cycle basis usually reduces the amount of work that has to be done by this later algorithm. Such algorithms span diverse applications like structural engineering [4], cycle analysis of electrical networks [5], and chemical ring perception [7]. And in many cases the network graphs of interest are directed graphs. Further, specific kinds of cycle bases of directed graphs have been studied in [14, 15, 8]. One special class is integral cycle bases [14, 15] in which the $d \times m$ cycle-edge incidence matrix has the property that all regular $d \times d$ submatrices have determinant ± 1; such cycle bases of minimum length are important in cyclic timetabling. [8] studied cycle bases in strongly connected digraphs where cycles were forced to follow the direction of the edges; such cycle bases are of particular interest in metabolic flux analysis.

Previous Work. There are several algorithms for computing a minimum cycle basis in an undirected graph [2, 5, 9, 10, 13] and the current fastest algorithm runs in $O(m^2n + mn^2 \log n)$ time [13], where m is the number of edges and n is the number of vertices. The first polynomial time algorithm for computing a minimum cycle basis in a directed graph had a running time of $\tilde{O}(m^4 n)$ [12]. Liebchen and Rizzi [16] gave an $\tilde{O}(m^{\omega+1} n)$ algorithm for this problem, where $\omega < 2.376$ is the exponent of matrix multiplication. This is the current fastest deterministic algorithm known for this problem. But faster randomized algorithms, like the $O(m^2 n \log n)$ Monte Carlo algorithm in [11], are known.

Our Contribution. In this paper, we present an $O(m^3 n + m^2 n^2 \log n)$ deterministic algorithm to compute a minimum cycle basis in a directed graph G with m edges and n vertices. The framework used in the algorithm is standard [5, 2, 13, 12]: we compute cycles C_i and supporting vectors N_i so that each C_i is the shortest cycle *not* orthogonal to its corresponding N_i, and each N_i is orthogonal to all previous C_j, $j < i$. This collection of cycles C_i is known to be a minimum cycle basis. Our improved algorithm for computing the C_i, N_i's rests on two ideas.

• We show how to compute the vectors N_i efficiently using fast matrix multiplication and inversion. In contrast to [16], the sizes of the matrices we need to multiply reduce with recursion depth yielding better performance. Our starting algorithm performs some arithmetic iteratively to compute the vectors N_i and this takes $\tilde{O}(m^4)$ time. So we delay the arithmetic and perform it in bulk using matrix multiplication and inversion. This idea was used in [13] - however there is a problem with this approach here. We are no longer in a finite field as was the case in [13] and the intermediate numbers could start having bit size $\tilde{\Theta}(d^2)$. Thus each arithmetic operation no longer takes $O(1)$ time and this makes the running time of such an algorithm at least $\tilde{\Theta}(m^{\omega+2})$, which is worse than the original iterative algorithm.

This problem is circumvented by working over a suitable ring $\mathbb{Z}_R$. And the important point is that there is no fixed R over which we work during the entire algorithm: whenever we perform the matrix multiplication and inversion, we determine a suitable R so that the inverse of the matrix that we seek to invert exists in $\mathbb{Z}_R$. This leads us to the vectors N_i mod R and the number R will be large enough so that we can easily recover the original vectors N_i from N_i mod R. The total time needed here now is $\tilde{O}(m^{\omega+1})$.

• A key step in our algorithm is a subroutine to compute a shortest cycle whose inner product with a given vector N is non-zero. This can be reduced via the Chinese Remainder Theorem to computing a shortest cycle C_p whose inner product with N is non-zero modulo p for some $p \in \{p_1, \ldots, p_d\}$, which is a collection of small primes. Here we present an $O(mn + n^2 \log n)$ algorithm to compute such a cycle C_p. This algorithm is obtained by computing two types of paths between each adjacent pair of vertices. The first path is a shortest path between these two nodes and the second is a shortest path whose residue class is different from the residue class of the first path. This whole procedure is repeated for each $p \in \{p_1, \ldots, p_d\}$ yielding $O(m^2n + mn^2 \log n)$ time for each C and thus $O(m^3n + m^2n^2 \log n)$ overall for all the d cycles.

2 Preliminaries

We are given a digraph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Without loss of generality, the underlying undirected graph of G is connected. Then $d = m - n + 1$ is the dimension of the cycle space of G. The minimum cycle basis of G will comprise d vectors $C_1 \ldots C_d$, each of length m. These vectors will have entries $\pm 1, 0$. We assume that we have ordered the edges in the edge set $E = \{e_1 \ldots e_m\}$ so that edges $e_{d+1} \ldots e_m$ form the edges of a spanning tree T of the underlying undirected graph. This means that the first d coordinates each of $C_1 \ldots C_d$ correspond to edges outside the tree T and the last $n-1$ coordinates are the edges of T. This will be important in our proofs in Section 4.

We assume that there are no multiple edges in G without loss of generality. It is easy to see that whenever there are two edges from u to v, the heavier edge (call it a) can be deleted from E and the least weight cycle $C(a)$ containing a can be added to the minimum cycle basis computed on $(V, E \setminus \{a\})$. The cycle $C(a)$ consists of the edge a and the shortest path between u and v in the underlying undirected graph. All such cycles can be computed by an all-pairs-shortest-paths computation in the underlying undirected graph of G, which takes $\tilde{O}(mn)$ time. Hence we can assume that $m \leq n^2$.

The notation $\langle v_1, v_2 \rangle$ denotes the standard inner product or dot product of the vectors v_1 and v_2.

3 Framework

We begin with a structural characterization of a minimum cycle basis. This characterization uses auxiliary rational vectors $N_1 \ldots N_i$ which serve as the scaffold for proving properties of $C_1 \ldots C_d$, as described below.

Theorem 1 *If cycles $C_1 \ldots C_d$ and vectors $N_1 \ldots N_d$ in $\mathbb{Q}^m$ satisfy the properties listed below then $C_1 \ldots C_d$ is a minimum cycle basis of G.*

1. **Prefix Orthogonality:** *Each N_i is orthogonal to all previous C_j, i.e., $\langle N_i, C_j \rangle = 0$ for all i, j, $1 \leq j < i \leq d$.*

2. **Non-Orthogonality:** $\langle N_i, C_i \rangle \neq 0$, *for all i,* $1 \leq i \leq d$.

3. **Shortness:** *C_i is the shortest cycle such that* $\langle N_i, C_i \rangle \neq 0$, *for all i,* $1 \leq i \leq d$.

Proof: First, we show that $C_1 \ldots C_d$ is a cycle basis of G by showing that these are linearly independent over $\mathbb{Q}$ (the dimension of the cycle space is known to be d so we do not need to prove that $C_1 \ldots C_d$ span the cycle space). Suppose not. Then a rational linear combination of a subset of these cycles yields 0; let the cycle with largest index in this subset be C_i. By Properties 1 and 2 of the theorem, taking the dot product of this linear combination with N_i yields a non-zero value on the left and a 0 value on the right, a contradiction.

Second, we show that $C_1 \ldots C_d$ is a minimum cycle basis of G. Suppose not and consider the smallest $i \geq 1$ such that $C_1 \ldots C_i$ is not in any minimum cycle basis. Then $C_1 \ldots C_{i-1}$ belong to some minimum cycle basis; consider this basis $\mathcal{K}$ (in the event that $i = 1$, let $\mathcal{K}$ be any minimum cycle basis). We will demonstrate a cycle $K \in \mathcal{K}$ satisfying the following properties:

A: $\langle N_i, K \rangle \neq 0$.

B: K can be written as a rational linear combination of C_i along with cycles in $\mathcal{K}/\{K\}$.

Property 1 of the theorem and [A] ensure that K is not one of $C_1 \ldots C_{i-1}$. [B] implies that $\mathcal{K}/\{K\} \cup \{C_i\}$ is a cycle basis. Property 3 of the theorem and [A] imply that C_i has weight at most that of K and therefore $\mathcal{K}/\{K\} \cup \{C_i\}$ is also a minimum cycle basis. $C_1 \ldots C_i$ belong to this minimum cycle basis, a contradiction.

It remains to demonstrate cycle K with the above properties. Since $\mathcal{K}$ is a basis not containing C_i, the cycle C_i must be a rational linear combination of some subset of cycles in $\mathcal{K}$. At least one of these cycles $K \in \mathcal{K}$ satisfies $\langle N_i, K \rangle \neq 0$, because $\langle N_i, C_i \rangle \neq 0$ by Property 2 in the theorem; [A] therefore holds. Further, [B] follows by rewriting the above linear combination to switch the sides of C_i and K. This completes the proof. ∎

4 A Simple Deterministic Algorithm

We show how to obtain N_i, C_i satisfying the criteria in Theorem 1 using a simple inductive deterministic algorithm. The algorithm starts with N_i initially set to 1 in the ith location and 0's elsewhere, for all i, $1 \leq i \leq d$. It then computes $C_1 \ldots C_d$ one by one in that sequence. The N_i's get modified as the C_i's are computed. The following invariants will hold after $C_1 \ldots C_{i-1}$ have been computed in the first $i-1$ iterations.

1. $N_1 \ldots N_i$ are frozen and will not be modified in the future.

2. For $j \in 1 \ldots i$, N_j can be non-zero only in the first j coordinates. Further N_j is necessarily non-zero in coordinate j.

3. For $j \in i \ldots d$, N_j can be non-zero only in the first $i-1$ coordinates and in coordinate j. Further N_j is necessarily non-zero in coordinate j.

4. For k, r such that $1 \leq r < k \leq i$, $\langle N_k, C_r \rangle = 0$.

5. For $j \in 1 \ldots i-1$, $\langle N_j, C_j \rangle \neq 0$. Further, C_j is the shortest cycle satisfying $\langle N_j, C_j \rangle \neq 0$.

6. $N_i \ldots N_d$ are orthogonal to $C_1 \ldots C_{i-1}$.

The *i*th Iteration. The ith iteration does the following:

- Computes C_i such that it is the shortest cycle satisfying $\langle N_i, C_i \rangle \neq 0$. This will take time $O(mn(m + n\log n))$.
- Updates $N_{i+1} \dots N_d$ so that these stay orthogonal to $C_1 \dots C_{i-1}$ and, in addition, become orthogonal to C_i. Updates to N_j, $i+1 \leq j \leq d$, affect only coordinates $1 \dots i$ in addition to coordinate j, which stays non-zero. This will take $O(md)$ arithmetic operations.

These changes are easily verified to maintain the invariants listed above.

Computing C_i. We call this problem the *non-orthogonal shortest cycle problem* and give an algorithm for this in Section 6. Here, we will focus only on showing that such a C_i exists. i.e., there is at least one cycle in G whose dot-product with N_i is non-zero. As an example, just take the fundamental cycle formed by edge i. Recall from Section 2 that edges have been ordered so the last $n-1$ edges are in a particular spanning tree and edges $1 \dots d$ are not in this spanning tree; the fundamental cycle of edge i with respect to this spanning tree is represented by a vector with a ± 1 in the ith coordinate and 0s in coordinates $1 \dots i-1$. By Invariant 2 above, N_i has a non-zero in coordinate i and 0s in $i+1 \dots m$; its dot-product with this fundamental cycle is therefore non-zero.

Updating $N_{i+1} \dots N_d$. We show below two methods (used in [12]) to update a particular N_j, $i+1 \leq j \leq d$; this procedure is then repeated for all j in this range. Method 1 is computationally expensive but shows that N_j is unique (modulo scaling) and also enables a bound on the norm of N_j. Method 2 is computationally fast and the one used by our algorithm. By virtue of uniqueness, the two methods yield the same results.

Method 1. To compute N_j, we solve the equations $\langle N_j, C_k \rangle = 0$, $k = 1 \dots i$ for N_j. We have i equations here, which we can solve for i variables. These variables will be the first i coordinates of N_j. Note that by Invariant 3 above applied to the end of iteration i instead of iteration $i-1$, we are allowed non-zero values in only the first i coordinates of N_j in addition to coordinate j. We will set the jth coordinate of N_j to 1 and solve the i equations for the first i coordinates. This can be captured in matrix form as $Ax = b$ where the kth row of A comprises the first i coordinates of C_k, $1 \leq k \leq i$, x comprises the first i coordinates of N_j, and b comprises the negated jth coordinates of $C_1 \dots C_i$.

We claim that A is non-singular and therefore a unique solution for N_j exists with jth coordinate 1, rational values in the first i coordinates, and 0's elsewhere. The proof for A's non-singularity mimics the argument in the first paragraph of Theorem 1. If A is indeed singular, then a rational linear combination of a subset of its rows yields 0; let the row with largest index in this subset be row l. Note that N_l has non-zero values only in the first l coordinates and therefore dot products of $C_1 \dots C_i$ with N_l are determined purely by the first l coordinates, i.e., those coordinates captured in A. Then taking the dot product of the above linear combination with N_l yields a non-zero value on the left (by Invariant 4) and a 0 value on the right, a contradiction. Thus there is a unique solution to $Ax = b$, which leads us to the rational vector N_j. Now scale N_j as $\det(A).N_j$ to give us $N_j \in \mathbb{Z}^m$. Alternatively, solving $Ax = \det(A).b$ would have led us to this integral vector.

Fact 1 *Since A is a $\pm 1, 0$ matrix of size $i \times i$ and b is a $\pm 1, 0$ vector, all determinants used in Cramer's Rule are bounded by $i^{i/2}$ using Hadamard's inequality. Therefore, at the end of the i-th iteration the absolute value of each entry in N_j, $j > i$, is bounded by $i^{i/2}$.*

Method 2. The trick in Method 2 is to compute the new N_j as a rational linear combination of the current N_j and of N_i. Since both of these are orthogonal to $C_1 \dots C_{i-1}$ by Invariant 6, any linear combination will continue to be orthogonal as well. So it suffices to find a linear combination of the current N_j and N_i which is orthogonal to C_i. This linear combination is given by the following, which is well defined as $\langle N_i, C_i \rangle \neq 0$.

$$N_j - \frac{\langle N_j, C_i \rangle}{\langle N_i, C_i \rangle} N_i \tag{1}$$

Note that the second term only modifies the first i coordinates of N_j; the j-th coordinate therefore stays non-zero. And at the end of the i-th iteration, N_j is orthogonal to $C_1,\ldots,C_i$ and can have non-zero coordinates only in its first i coordinates and j-th coordinate. However these coordinates could have very large numerators and denominators unless we keep canceling their gcds. Since gcd computation is expensive, we will instead scale N_j as:

$$N_j = N_j \frac{\langle N_i, C_i\rangle}{\langle N_{i-1}, C_{i-1}\rangle} \tag{2}$$

(take $\langle N_0, C_0\rangle = 1$)

So an update step i.e., (1) is always followed by a scaling step, i.e., (2). This makes the jth coordinate of N_j equal to $\Pi_{k=1}^{i}\langle N_k, C_k\rangle / \langle N_{k-1}, C_{k-1}\rangle = \langle N_i, C_i\rangle$. And since N_j is orthogonal to $C_1,\ldots,C_i$, the first i coordinates of N_j satisfy $Ax = \langle N_i, C_i\rangle b$. And from the following claim we see that both Method 1 and Method 2 give us the same vector N_j.

Claim. $\langle N_i, C_i\rangle = \det(A)$.

Multiply the matrix A with the $i \times i$ matrix $\mathcal{S}$ where the k-th column of $\mathcal{S}$ is N_k truncated to its first i coordinates. $\mathcal{S}$ is an upper triangular matrix. We have $\mathcal{S}[k,k] = \langle N_{k-1}, C_{k-1}\rangle$ because of scaling, so $\det(\mathcal{S}) = \Pi_{k=1}^{i}\langle N_{k-1}, C_{k-1}\rangle$. The product matrix $A\mathcal{S}$ has its (j,k)-th element as $\langle C_j, N_k\rangle$, so it is a lower triangular matrix. And $\det(A\mathcal{S}) = \Pi_{k=1}^{i}\langle N_k, C_k\rangle$. Since $\det(A\mathcal{S}) = \det(A).\det(\mathcal{S})$, we get

$$\det(A)\Pi_{k=1}^{i}\langle N_{k-1}, C_{k-1}\rangle = \Pi_{k=1}^{i}\langle N_k, C_k\rangle.$$

Thus the claim follows.

The time taken for updating and scaling N_j is $O(m)$ arithmetic steps. Thus the total number of arithmetic operations in the i-th iteration is $\tilde{O}((d-i)m) = O(md)$ over all j, $i+1 \le j \le d$, as required.

5 A Faster Deterministic Algorithm

The algorithm above modifies or updates each of $N_{i+1}\ldots N_d$ when C_i is computed; this takes $\tilde{O}(md^2) = \tilde{O}(m^3)$ number of arithmetic operations (over all i). Now arithmetic operations no longer cost us $O(1)$ time since we are dealing with numbers as large as $d^{d/2}$. Hence we pay $\tilde{O}(d)$ time per arithmetic operation and this results in an $\tilde{O}(m^4)$ running time of Method 2. Instead we would like to bound the total running time of all the update steps by $\tilde{O}(m^{\omega+1})$. So we generalize Method 2 above in two ways.

- First, we delay updates until after several new cycles C_i have been generated. For instance, we update $N_{d/2+1}\ldots N_d$ not after each new cycle but in bulk after *all* of $C_1, C_2, \ldots, C_{d/2}$ are computed.
- We use a fast matrix multiplication method to do the updates for all of $N_{d/2+1}\ldots N_d$ together, and not individually as before.

The Scheme. The faster deterministic algorithm starts with the same configuration for the N_i's as before (N_i initially set to 1 in the ith location and 0's elsewhere, for all i, $1 \le i \le d$). It then executes 3 steps. First, it computes $C_1 \ldots C_{\lfloor d/2\rfloor}$ and $N_1 \ldots N_{\lfloor d/2\rfloor}$ recursively, This is done while ignoring $N_{\lfloor d/2\rfloor+1}\ldots N_d$. Second, it runs a bulk update step in which $N_{\lfloor d/2\rfloor+1}\ldots N_d$ are modified so they are all orthogonal to $C_1 \ldots C_{\lfloor d/2\rfloor}$. And third, $C_{\lfloor d/2\rfloor+1}\ldots C_d$ are computed recursively modifying $N_{\lfloor d/2\rfloor+1}\ldots N_d$ in the process.

A crucial point to note about the second recursive call is that it modifies $N_{\lfloor d/2\rfloor+1}\ldots N_d$ while ignoring $C_1 \ldots C_{\lfloor d/2\rfloor}$ and $N_1 \ldots N_{\lfloor d/2\rfloor}$; how then does it retain the orthogonality of $N_{\lfloor d/2\rfloor+1}\ldots N_d$ with $C_1 \ldots C_{\lfloor d/2\rfloor}$ obtained in the bulk update step? The trick lies in the fact that the new values of $N_{\lfloor d/2\rfloor+1}\ldots N_d$ resulting from the second recursive call are obtained as rational linear combinations of $N_{\lfloor d/2\rfloor+1}\ldots N_d$ themselves;

since the latter are already orthogonal to $C_1 \dots C_{\lfloor d/2 \rfloor}$, so are the former. This property allows the second recursive call to work strictly in the bottom half of the data without looking at the top half.

The base case for the recursion is a subproblem of size 1 (without loss of generality, let this subproblem involve C_i, N_i) in which case the algorithm simply retains N_i as it is and computes C_i using the algorithm in Section 6. As regards time complexity, the bulk update step will be shown to take $O(m * d^{\omega-1})$ arithmetic operations.

The Bulk Update Procedure. We describe this procedure say, in the recursive call that computes the cycles $C_\ell, \dots, C_h$. That is, the cycles $C_\ell, \dots, C_{mid}$ have already been computed, where $mid = \lceil (\ell + h)/2 \rceil - 1$ and $N_{mid+1} \dots N_h$ are to be modified so that they are all orthogonal to $C_\ell \dots C_{mid}$. We do this by setting each N_j, $mid + 1 \leq j \leq h$, to a rational linear combination of $N_\ell \dots N_{mid}$ and of the current N_j itself, the latter having coefficient $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ in the linear combination. This coefficient is chosen because N_j currently has a coefficient of $\langle N_{\ell-1}, C_{\ell-1} \rangle$ in its jth coordinate and after the update, we want that coordinate to become $\langle N_{mid}, C_{mid} \rangle$. More precisely, N_j is set to $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle N_j + \sum_{k=\ell}^{mid} \alpha_{j,k} N_k$, where the $\alpha_{j,k}$'s are to be determined in a way which ensures that N_j becomes orthogonal to $C_\ell \dots C_{mid}$, i.e.,

$$\frac{\langle N_{mid}, C_{mid} \rangle}{\langle N_{\ell-1}, C_{\ell-1} \rangle} \langle C_i, N_j \rangle + \sum_{k=\ell}^{mid} \alpha_{j,k} \langle C_i, N_k \rangle$$

is an all 0s vector for all $i, \ell \leq i \leq mid$, and all j, $mid + 1 \leq j \leq h$. Rewriting this in matrix form, we get $C \cdot N_d \cdot D = -C \cdot N_u \cdot X$, where (taking $k = mid - \ell + 1$) we have

- C is a $k * m$ matrix, the ith row of which is C_i.
- N_d is a $m * (h - k)$ matrix, the ith column of which is N_{mid+i}.
- N_u is a $m * k$ matrix, the ith column of which is $N_{\ell+i-1}$.
- D is an $(h - k) * (h - k)$ scalar matrix where each diagonal element is $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$.
- X is the $k * (h - k)$ matrix of variables $\alpha_{j,k}$, with j going along the columns and k along the rows.

To compute the $\alpha_{j,k}$'s, we solve for $X = -(C \cdot N_u)^{-1} \cdot C \cdot N_d \cdot D$. Using fast matrix multiplication, we can compute $C \cdot N_u$ and $C \cdot N_d$ in $O(m * k^{\omega-1})$ time, by splitting the matrices into d/k square blocks and using fast matrix multiplication to multiply the blocks. Multiplying each element of $C \cdot N_d$ with a scalar $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ gives us $C \cdot N_d \cdot D$. Then find the inverse of $C \cdot N_u$ in $O(k^\omega)$ time (this inverse exists because $C \cdot N_u$ is a lower triangular matrix whose diagonal entries are $\langle C_i, N_i \rangle \neq 0$). Subsequently, we can multiply $(C \cdot N_u)^{-1}$ with $C \cdot N_d \cdot D$ in $O(k^\omega)$ time. Finally, we obtain $N_{mid+1} \dots N_d$ from X using the product $N_u \cdot X$, which takes $O(m * k^{\omega-1})$ time. The time required for the bulk update step is thus $O(m * k^{\omega-1})$.

However, this time actually refers to the number of rational arithmetic operations and not to the running time itself. We would like to perform each arithmetic operation in $\tilde{O}(m)$ time, then the actual time required for the bulk update step is $\tilde{O}(m^2 * k^{\omega-1})$. This would lead to an overall time complexity of $T(d) = 2T(d/2) + \tilde{O}(m^2 * d^{\omega-1})$ and $T(1) = O(m^2 n + mn^2 \log n)$. This yields $T(d) = O(m^3 n + m^2 n^2 \log n + m^{\omega+1} poly(\log m))$, which is $O(m^3 n + m^2 n^2 \log n)$, because $m \leq n^2$ implies $\tilde{O}(m^{\omega+1})$ is always $o(m^3 n)$.

However, in the algorithm as presented above, the entries in $(C \cdot N_u)^{-1}$ could be very large. The elements in $C \cdot N_u$ have values up to $d^{\Theta(d)}$, which would result in the entries in $(C \cdot N_u)^{-1}$ being as large as $d^{\Theta(d^2)}$. So each arithmetic operation then costs us up to d^2 time and the overall time through fast matrix multiplication costs us $\tilde{\Theta}(m^{\omega+2})$ time, which is worse than Method 2.

The good news is that these numbers $\alpha_{j,k}$ are just *intermediate* numbers in our computation. That is, they are the coefficients in

$$\sum_{k=1}^{mid} \alpha_{j,k} N_k + \frac{\langle N_{mid}, C_{mid} \rangle}{\langle N_{\ell-1}, C_{\ell-1} \rangle} N_j.$$

Our final aim is to determine the actual coordinates of N_j which are at most $d^{d/2}$ (refer Fact 1).

This motivates us to do modular arithmetic. We could work over the finite fields $\mathbb{F}_{p_1}, \mathbb{F}_{p_2}, \ldots, \mathbb{F}_{p_s}$ where $p_1, \ldots, p_s$ are small primes (say, in the range d to d^2) and try to retrieve N_j from N_j mod $p_1, \ldots, N_j$ mod p_s, which is possible if $s \approx d$. Arithmetic in $\mathbb{F}_p$ takes $O(1)$ time and we thus spend $O(s * m * k^{\omega-1})$ time for the update step now. However, if it is the case that some p is a divisor of some $\langle N_i, C_i \rangle$ where $\ell \leq i \leq mid$, then we cannot invert $C \cdot N_u$ in the field $\mathbb{F}_p$. Since each number $\langle N_i, C_i \rangle$ could be as large as $d^{d/2+1}$, in order to be able to determine d primes which are relatively prime to all of $\langle N_\ell, C_\ell \rangle, \ldots, \langle N_{mid}, C_{mid} \rangle$ we need to spend at least $k^2 d^2$ time (to test kd primes w.r.t. k d-bit numbers). We cannot afford so much time per update step.

Another idea is to work over just one finite field $\mathbb{F}_q$ where q is a large prime. If $q > d^{d/2+1}$, then it can never be a divisor of any $\langle N_i, C_i \rangle$, so we can always carry out our arithmetic in $\mathbb{F}_q$ and never be in trouble. Arithmetic in $\mathbb{F}_q$ costs us $\tilde{\Theta}(d)$ time if $q \approx d^d$. Then our update step takes $\tilde{O}(m^2 * k^{\omega-1})$ time as we desired. But computing such a large prime q is a very difficult problem.

Idea. Fast matrix multiplication algorithms work over *rings*. So we do not have to restrict ourselves to fields. Let us do the above computation modulo a large integer R, say $R \approx d^d$. Then intermediate numbers do not grow more than R and we can retrieve N_j directly from N_j mod R, because R is large.

What properties do we need of R? The integer R must be relatively prime to $\langle N_\ell, C_\ell \rangle, \langle N_{\ell+1}, C_{\ell+1} \rangle, \ldots, \langle N_{mid}, C_{mid} \rangle$ so that that triangular matrix $C \cdot N_d$ which has these elements along the diagonal is invertible in $\mathbb{Z}_R$. And R must also be relatively prime to $\langle N_{\ell-1}, C_{\ell-1} \rangle$ so that the number $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ is defined in $\mathbb{Z}_R$. So our aim now is to determine a large integer R with this relatively prime property.

We do this in the following manner. It is easy to see that this number R has the properties that we want.

1. Right at the beginning of the algorithm, compute d^2 primes $p_1, ..., p_{d^2}$, where each of these primes is at least d. Then form the d products: $P_1 = p_1 \cdots p_d; P_2 = p_1 \cdots p_{2d}; P_3 = p_1 \cdots p_{3d}; \ldots P_d = p_1 \cdots p_{d^2}$.
2. Then during our current update step, form the product

   $\mathcal{L} = \langle N_{\ell-1}, C_{\ell-1} \rangle \langle N_\ell, C_\ell \rangle \cdots \langle N_{mid}, C_{mid} \rangle$.
3. By doing a binary search on $P_1, \ldots, P_d$, determine the smallest $t \geq 0$ such that P_{t+1} does not divide $\mathcal{L}$.
4. Determine a $p_r \in \{p_{td+1}, \ldots, p_{td+d}\}$ that does not divide $\mathcal{L}$. Compute $R = p_r^d$.

Cost of computing R. We will always perform arithmetic on large integers using Schönhage-Strassen multiplication, so that it takes $\tilde{O}(d)$ time to multiply two d-bit numbers. Whenever we perform a sequence of multiplications, we will perform it using a tree so that d numbers (each of bit size $\tilde{O}(d)$) can be multiplied in $\tilde{O}(d^2)$ time. The value of $\pi(s)$, the number of primes less than s, is given by $s/6 \log s \leq \pi(s) \leq 8s/\log s$ [1]. So all the the primes $p_1, \ldots, p_{d^2}$ are $\tilde{O}(d^2)$, and computing them takes $\tilde{O}(d^2)$ time. The products $P_1, \ldots, P_d$ are computed just once in the whole course of the algorithm. That takes $\tilde{O}(d^3)$ preprocessing time.

In the update step, we compute $\mathcal{L}$, which takes $\tilde{O}(d^2)$ time. The product $p_{td+1} \cdots p_{td+d}$ is found in $\tilde{O}(d^2)$ time by binary search. Determine an $p_r \in \{p_{td+1}, \ldots, p_{td+d}\}$ that does not divide $\mathcal{L}$ by testing which of the two products: $p_{td+1} \cdots p_{td+d/2}$ or $p_{td+d/2+1} \cdots p_{td+d}$ does not divide $\mathcal{L}$ and recurse on the product that does not divide $\mathcal{L}$. Thus R can be computed in $\tilde{O}(d^2)$ time.

Computation in $\mathbb{Z}_R$. We need to invert the matrix $C \cdot N_u$ in the ring $\mathbb{Z}_R$. Recall that this matrix is lower triangular. Computing the inverse of a lower triangular matrix is easy. If

$$C \cdot N_u = \begin{pmatrix} W & 0 \\ Y & Z \end{pmatrix}, \text{ then we have } (C \cdot N_u)^{-1} = \begin{pmatrix} W^{-1} & 0 \\ -Z^{-1}YW^{-1} & Z^{-1} \end{pmatrix}.$$

Hence to invert $C \cdot N_u$ in $\mathbb{Z}_R$ we need the multiplicative inverses of only its diagonal elements: $\langle C_\ell, N_\ell \rangle, \ldots, \langle C_{mid}, N_{mid} \rangle$ in $\mathbb{Z}_R$. Using Euclid's gcd algorithm each inverse can be computed in $\tilde{O}(d^2)$ time since each of the numbers involved here and R have bit size $\tilde{O}(d)$. The matrix $C \cdot N_u$ is inverted via fast matrix multiplication and once we compute $(C \cdot N_u)^{-1}$, X can be easily computed in $\mathbb{Z}_R$ as $-(C \cdot N_u)^{-1} \cdot C \cdot N_d \cdot D$ by fast matrix multiplication. Then we determine all N_j mod R for $mid + 1 \leq j \leq h$ from the product $N_u \cdot X$.

Each entry of N_j can have absolute value at most $d^{d/2}$ (from Fact 1). The number R is much larger than this, $R > d^d$. So if any coordinate, say n_l in N_j mod R is larger than $d^{d/2}$, then we can retrieve the original n_l as $n_l - R$. Thus we can retrieve N_j from N_j mod R in $O(kd)$ time. The time complexity for the update step, which includes matrix operations, gcd computations and other arithmetic, is $\tilde{O}(m^2k^{\omega-1} + d^2k)$ or $\tilde{O}(m^2k^{\omega-1})$. Thus our overall recurrence becomes $T(d) = 2T(d/2) + \tilde{O}(d^{\omega+1})$. As described earlier, this yields a running time of $T(d) = O(m^3n + m^2n^2 \log n)$ under the assumption that $T(1) = O(m^2n + mn^2 \log n)$. The bound on $T(1)$ will be shown in the next section. We now conclude with the following theorem.

Theorem 2 *A minimum cycle basis in a weighted directed graph, with m edges and n vertices and non-negative edge weights, can be computed in $O(m^3n + m^2n^2 \log n)$ time.*

6 Solving for Non-Orthogonal Shortest Cycles

Given a directed graph G with non-negative edge weights, we consider the problem of computing a shortest cycle C in G whose inner product with a given vector N is non-zero modulo a number $p = O(d \log d)$. Recall that C can traverse edges of G in forward or reverse direction; the vector representation of C has a 1 for every forward edge in the cycle, a -1 for every reverse edge, and a 0 for edges not present at all in the cycle. This vector representation is used for computing inner products with N. The weight of C itself is simply the sum of the cycle edge weights. We show how to compute C in time $O(n^2 \log n + mn)$.

Definitions. To compute shortest paths and cycles, we will work with the undirected version of G. Directions will be used only to compute the *residue class* of a path or cycle, i.e., the inner product between the vector representation of this cycle or path and N modulo p. Let p_{uv} denote a shortest path between vertices u and v and let f_{uv} denote its length and r_{uv} its residue class. Let s_{uv} be the length of a shortest path, if any, between u and v in a residue class distinct from r_{uv}. Observe, that the value of s_{uv} is independent of the choice of p_{uv}.

We will show how to compute f_{uv} and s_{uv} for all pairs of vertices u, v in time $O(n^2 \log n + mn)$; as is standard, we will also compute paths realizing these lengths in addition to computing the lengths themselves. The following claim tells us how these paths can be used to compute the shortest non-orthogonal cycle: simply take each edge uv and combine it with s_{vu} to get a cycle; the shortest of all these cycles having a non-zero residue-class is our required cycle.

Lemma 1 *Consider the shortest cycle $C = u_0u_1 \ldots u_ku_0$ whose residue class is non-zero modulo p and whose shortest edge is u_0u_1. Then the length of the path $u_1u_2 \ldots u_ku_0$ equals $s_{u_1u_0}$ and the length of the edge u_0u_1 equals $f_{u_1u_0}$. Further, the path $u_1u_2 \ldots u_ku_0$ has a distinct residue class than the edge u_1u_0.*

Proof: First, we show that the path $u_1u_2 \ldots u_ku_0$ has a distinct residue class than the edge u_1u_0. Let x denote the residue class of the path $u_1u_2 \ldots u_ku_0$ and y denote the residue class of the edge u_0u_1. Since

$u_0u_1 \ldots u_ku_0$ is in a non-zero residue class, $x+y \not\equiv 0(\bmod p)$, so $x \not\equiv -y(\bmod p)$. From the definition of the vector representation of path above, the residue class of the edge u_1u_0 is $-y$ (because the direction of traversal of the edge is reversed). The claim follows.

Now, if the length of u_1u_0 is strictly greater than $f_{u_1u_0}$, then consider any shortest path π between u_1 and u_0 (which of course has length $f_{u_1u_0}$). It is easily shown that combining π with u_1u_0 and with $u_1u_2 \ldots u_ku_0$ respectively yields 2 cycles which are shorter than C and which are in distinct residue classes; this contradicts the definition of C. Therefore, the edge u_1u_0 has length $f_{u_1u_0}$.

Next, by virtue of the definition of $s_{u_1u_0}$, and since $u_1u_2 \ldots u_ku_0$ has a distinct residue class than the edge u_1u_0, $u_1u_2 \ldots u_ku_0$ cannot be shorter than $s_{u_1u_0}$. Suppose, for a contradiction that $u_1u_2 \ldots u_ku_0$ is strictly longer than $s_{u_1u_0}$. A simple argument shows that combining the path which realizes $s_{u_1u_0}$ with the edge u_1u_0 yields a cycle which is shorter than C and which has a non-zero residue-class, thus contradicting the definition of C. The lemma follows. ∎

Computing f_{uv} and s_{uv}. We first find any one shortest path (amongst possibly many) between each pair of vertices u and v; this gives us the p_{uv}'s, f_{uv}'s, and r_{uv}'s. The time taken using Dijkstra's algorithm is $O(n^2 \log n + mn)$. For each pair u, v, we now need to find a shortest path between u, v with residue-class distinct from r_{uv}; the length of this path will be s_{uv}. Use q_{uv} to denote any such path. We show how a modified Dijkstra search can compute these paths in the $O(n^2 \log n + mn)$. The following lemma shows the key prefix property of the s_{uv} paths needed for a Dijkstra-type algorithm.

Lemma 2 *For any u and v, the path q_{uv} can be chosen from the set $\{p_{uw} \circ wv, q_{uw} \circ wv\ ;\ wv \in E\}$. Here $p \circ e$ denotes the path p extended by the edge e.*

Proof: Consider any path π between u and v realizing the value s_{uv} (it has length s_{uv} and residue class distinct from r_{uv}). Let w denote the penultimate vertex on this path and let π' be the prefix path from u to w. We first show that π' has either length f_{uw} with residue-class r_{uw} or length s_{uw} and residue class distinct from r_{uw}. The claim follows from the following five cases.

• The length of π' is shorter than f_{uw}. This is not possible by the definition of f_{uw}.

• The length of π' equals f_{uw} but the residue-class is not r_{uw}. By the definition of s_{uw}, $s_{uw} = f_{uw}$ and the claim follows.

• The length of π' is strictly between f_{uw} and s_{uw}. By the definition of s_{uw}, the residue-class of π' must be r_{uw}. Extending p_{uw} by the edge from w to v gives us a path in the same residue class as π and shorter than π, a contradiction.

• The length of π' equals s_{uw} and the residue class is r_{uw}. Then, if $f_{uw} < s_{uw}$, extending p_{uw} via edge wv gives us a path in the same residue class as π and shorter than π, a contradiction. So $f_{uw} = s_{uw}$. The claim follows.

• The length of π' is strictly greater than s_{uw}. Consider paths between u and w realizing f_{uw} and s_{uw}, respectively. Extending each of these using the edge wv gives us two paths from u to v which have distinct residue-classes. One of these paths must have a residue-classes distinct from r_{uv}; let this path be denoted by π''. Since π'' is shorter than π, the length of π cannot be s_{uv}, a contradiction.

If π' has length f_{uw} and residue class r_{uw}, $p_{uw} \circ vw$ has the same length and residue class as π and we are done. So assume that π' has length s_{uw} and residue class distinct from r_{uw}. Then $q_{uw} \circ wv$ has the same length as π. If its residue class is also distinct from r_{uv}, we are done. If its residue class is equal to r_{uv}, $p_{uw} \circ vw$ has a residue class distinct from r_{uv}. Also, its length is at most the length of π and we are done. ∎

We now show to compute the s_{uv}'s for any fixed u in time $O(n \log n + m)$ with a Dijkstra-like algorithm. Repeating this for every source gives the result. The algorithm differs from Dijkstra's shortest path algorithm only in the initialization and update steps, which we describe below. We use the notation key_{uv} to denote the key used to organize the priority heap; key_{uv} will finally equal s_{uv}.

Initialization. We first set key_{uv} for all vertices v to ∞. Next, we consider each vertex w in turn. For each edge wv incident on w, we extend p_{uw} via the edge wv and update key_{uv} to the length of this path provided its residue class is distinct from r_{uv}.

The Update Step. Suppose we have just removed w from the priority queue. We consider the u to w path of length key_{uw} which was responsible for the current key value of w. For each edge wv incident on w, we extend this path via the edge wv. We update key_{uv} to the length of this path provided its residue class is distinct from r_{uv}.

Correctness. We need to show that key_{uv} is set to s_{uv} in the course of the algorithm (note that one doesn't need to worry about the residue-class since any path that updates key_{uv} in the course of the algorithm has residue-class distinct from r_{uv}). This follows immediately from the Lemma above. If s_{uv} is defined by $p_{uw} \circ wv$ for some neighbor w, key_{uv} is set to s_{uv} in the initialization step. If s_{uv} is defined by $q_{uw} \circ wv$ for some neighbor w, key_{uv} is set to s_{uv} in the update step.

The original problem. Our original problem was to compute a shortest cycle whose inner product with N is non-zero. But any cycle which satisfies $\langle C', N \rangle \neq 0$ should satisfy $\langle C', N \rangle \neq 0 (\bmod p)$ for some $p \in \{p_1, \ldots, p_d\}$ where $p_1, \ldots, p_d$ are distinct primes, each of which is at least d. This follows from the isomorphism of ring $\mathbb{Z}_{\pi p_i}$ to $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_d}$. So any non-zero element whose absolute value is less than $\pi_{i=1}^{d} p_i$ is mapped to a tuple of its residues that is not all 0s. And $|\langle C', N \rangle| \leq \|N\|_1 \leq d.d^{d/2} < \pi_{i=1}^{d} p_i$. Hence, $T(1)$, the time taken to compute C_i using N_i is $O(d * (mn + mn^2 \log n))$ or $O(m^2 n + mn^2 \log n)$.

References

[1] T. M. Apostol. *Introduction to Analytic Number Theory*. Springer-Verlag, 1997.

[2] F. Berger, P. Gritzmann, and S. de Vries. Minimum Cycle Bases for Network Graphs. *Algorithmica*, 40(1): 51-62, 2004.

[3] B. Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*, Springer, Berlin, 1998.

[4] A. C. Cassell and J. C. Henderson and K. Ramachandran. Cycle bases of minimal measure for the structural analysis of skeletal structures by the flexibility method *Proc. Royal Society of London Series A*, 350: 61-70, 1976.

[5] J.C. de Pina. *Applications of Shortest Path Methods*. PhD thesis, University of Amsterdam, Netherlands, 1995.

[6] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, 1982.

[7] P. M. Gleiss. *Short cycles: minimum cycle bases of graphs from chemistry and biochemistry*. PhD thesis, Universität Wien, 2001.

[8] P. M. Gleiss, J. Leydold, and P. F. Stadler. Circuit bases of strongly connected digraphs. *Discussiones Math. Graph Th.*, 23: 241-260, 2003.

[9] Alexander Golynski and Joseph D. Horton. A polynomial time algorithm to find the minimum cycle basis of a regular matroid. In *8th Scandinavian Workshop on Algorithm Theory*, 2002.

[10] J. D. Horton. A polynomial-time algorithm to find a shortest cycle basis of a graph. *SIAM Journal of Computing*, 16:359–366, 1987.

[11] T. Kavitha. An $\tilde{O}(m^2 n)$ Randomized Algorithm to compute a Minimum Cycle Basis of a Directed Graph. *In Proc. of ICALP*, LNCS 3580: 273-284, 2005.

[12] T. Kavitha and K. Mehlhorn. Algorithms to compute Minimum Cycle Basis in Directed Graphs Full version submitted to special issue, preliminary version in *Proc. of STACS*, LNCS 3404: 654-665, 2005.

[13] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. A faster algorithm for Minimum Cycle Basis of graphs. *In Proc. of ICALP*, LNCS 3142: 846-857, 2004.

[14] Christian Liebchen. Finding Short Integral Cycle Bases for Cyclic Timetabling. *In Proc. of ESA*, LNCS 2832: 715-726, 2003.

[15] C. Liebchen and L. Peeters. On Cyclic Timetabling and Cycles in Graphs. Technical Report 761/2002, TU Berlin.

[16] C. Liebchen and R. Rizzi. A Greedy Approach to compute a Minimum Cycle Basis of a Directed Graph. Technical Report 2004/31, TU Berlin.

7 Some Examples

As mentioned in Section 1, every cycle basis of a directed graph need not project onto an undirected cycle basis. The following example (Figure 1) shows this.

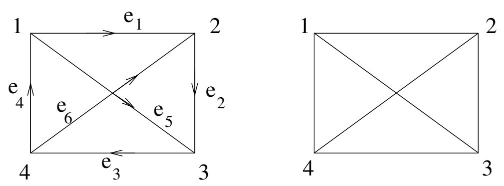

Figure 1: Directed K_4 and the underlying undirected graph

The three 4-cycles in the above directed graph in Figure 1: $C_1 = (e_1,e_2,e_3,e_4)$, $C_2 = (e_1,e_6,e_3,e_5)$ and $C_3 = (e_2,e_5,e_4,e_6)$ given by the vectors $(1,1,1,1,0,0),(1,0,-1,0,-1,-1)$, and $(0,1,0,-1,-1,1)$ are linearly independent over $\mathbb{Q}$. Hence they form a cycle basis for the directed K_4. But in the underlying undirected graph, each of these cycles is equal to the sum of the other two modulo 2, so C_1,C_2,C_3 do not form a cycle basis for undirected K_4.

Further, there are directed graphs in which the minimum cycle basis has lower weight than any undirected cycle basis. Such an example was given in [16]:

Consider the generalized Petersen graph $P_{7,2}$ in Figure 2. Call an edge (u,v) an *inner edge* if $\{u,v\} \subset \{0,1,\dots,6\}$. Similarly call an edge (u,v) an *outer edge* if $\{u,v\} \subset \{a,\dots,g\}$. The seven edges that remain are called *spokes*. Assign a weight function w by assigning weight two to the seven inner edges and weight three to the outer edges and spokes. It can be shown that $(P_{7,2},w)$ has girth 14 and there are precisely eight cycles having weight 14.

Every edge of $P_{7,2}$ belongs to precisely two of the eight cycles with weight 14. Therefore in the undirected case, these $8 = m-n+1$ cycles are not independent over $\mathbb{F}_2$. Thus in the undirected case, every cycle basis of $(P_{7,2},w)$ has weight at least 113. In the directed case, these 8 cycles under any orientation of edges can be shown to be linearly independent. So there is a directed cycle basis of weight 112.

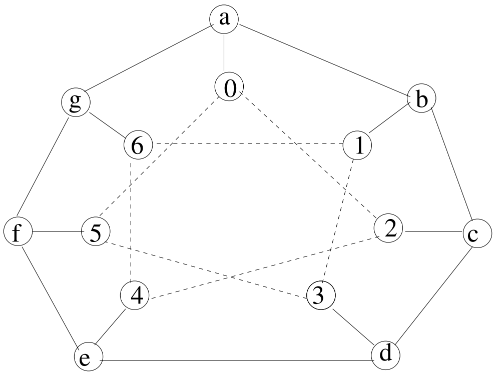

Figure 2: The generalized Petersen graph $P_{7,2}$.