A Faster Deterministic Algorithm for Minimum Cycle Bases in Directed Graphs

Ramesh Hariharan[1], Telikepalli Kavitha[1,⋆], and Kurt Mehlhorn[2,⋆⋆]

[1] Indian Institute of Science, Bangalore, India
{ramesh, kavitha}@csa.iisc.ernet.in
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
mehlhorn@mpi-inf.mpg.de

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 minimum cycle basis is a set of cycles of minimum weight that span the cycle space of G. 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. We also slightly improve the running time of the current fastest randomized algorithm from $O(m^2n \log n)$ to $O(m^2n + mn^2 \log n)$.

1 Introduction

Let $G = (V, E)$ be a directed graph with m edges and n vertices. A *cycle* in G is actually a cycle in the underlying undirected graph, i.e., edges are traversable in both directions. Associated with each cycle is a $\{-1, 0, 1\}$ edge incidence vector: 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 is called a *cycle basis* if it forms a basis for this vector space. When G is connected, the cycle space has dimension $d = m - n + 1$. We assume that there is a weight function $w : E \to \mathbb{R}^{\geq 0}$, i.e., the edges of G have non-negative weights assigned to them. The weight of a cycle basis is the sum of the weights of its cycles. A *minimum cycle basis* of G is a cycle basis of minimum weight. We consider the problem of computing a minimum cycle basis in a given digraph.

⋆ This research was partially supported by a "Max Planck-India Fellowship" provided by the Max Planck Society.

⋆⋆ Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

M. Bugliesi et al. (Eds.): ICALP 2006, Part I, LNCS 4051, pp. 250–261, 2006.

A related problem pertains to undirected graphs, where a $\{0,1\}$ edge incidence vector is associated 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 $\mathbb{Z}_2$. 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. Examples of both phenomena were given in [16]. Thus, one cannot find a minimum cycle basis for a directed graph by simply working with the underlying undirected graph. Books by Deo [6] and Bollobás [3] have an in-depth coverage of cycle bases.

Motivation. Apart from its interest as a natural question, an efficient algorithm for computing a minimum cycle basis has several applications. A minimum cycle basis is primarily used 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 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]. The network graphs of interest are frequently 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. Cycle bases in strongly connected digraphs where cycles are forced to follow the direction of the edges were studied in [8]; such cycle bases are of particular interest in metabolic flux analysis.

Previous Work and Our Contribution. 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]. The first polynomial time algorithm for computing a minimum cycle basis in a directed graph had a running time of $\tilde{O}(m^4n)$ [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 was the current fastest deterministic algorithm. A faster randomized algorithm of Monte Carlo type with running time $O(m^2n \log n)$ exists [11].

We present an $O(m^3n + m^2n^2 \log n)$ deterministic algorithm for this problem and improve the running time of the randomized algorithm to $O(m^2n + mn^2 \log n)$. The running time of our deterministic algorithm is m times the running time of the fastest algorithm for computing minimum cycle bases in undirected graphs and we leave it as a challenge to close the gap. The increased complexity seems to stem from the larger base field. Arithmetic in $\mathbb{Z}_2$ suffices for undirected graphs. For directed graphs, the base field is $\mathbb{Q}$ and this seems to necessitate the handling of large numbers. Also, the computation of a shortest cycle that has a non-zero dot product with a given vector seems more difficult in directed graphs than in undirected graphs.

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 consists of d cycles $C_1, \ldots, C_d$. We describe cycles by their incidence vectors in $\{-1, 0, +1\}^m$. 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 3.

We can also assume that there are no multiple edges in G. 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 (call it $C(a)$) that contains the edge 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 $m \leq n^2$.

Framework. We begin with a structural characterization of a minimum cycle basis, which is simple to show. This framework was introduced by de Pina [5]; it uses auxiliary rational vectors $N_1, \ldots, N_d$ which serve as a scaffold for proving that $C_1, \ldots, C_d$ form a minimum cycle basis. We use $\langle v_1, v_2 \rangle$ to denote the standard inner product or dot product of the vectors v_1 and v_2.

Theorem 1. *Cycles $C_1, \ldots, C_d$ form a minimum cycle basis if there are vectors $N_1, \ldots, N_d$ in $\mathbb{Q}^m$ such that for all i, $1 \leq i \leq d$:*

1. **Prefix Orthogonality:** *N_i is orthogonal to all previous C_j, i.e., $\langle N_i, C_j \rangle = 0$ for all j, $1 \leq j < i$.*
2. **Non-Orthogonality:** *$\langle N_i, C_i \rangle \neq 0$.*
3. **Shortness:** *C_i is a shortest cycle with $\langle N_i, C_i \rangle \neq 0$.*

3 A Simple Deterministic Algorithm

We present the simple deterministic algorithm from [12], that computes N_i's and C_i's satisfying the criteria in Theorem 1.

The algorithm Deterministic-MCB:

1. Initialize the vectors $N_1, \ldots, N_d$ of $\mathbb{Q}^m$ to the first d vectors $e_1, \ldots, e_d$ of the standard basis of $\mathbb{Q}^m$. (The vector e_i has 1 in the i-th position and 0's elsewhere.)
2. For $i = 1$ to d do
   - compute C_i to be a shortest cycle such that $\langle C_i, N_i \rangle \neq 0$.

– for $j = i + 1$ to d do

$$\text{update } N_j \text{ as: } N_j = N_j - N_i \frac{\langle C_i, N_j \rangle}{\langle C_i, N_i \rangle}$$

$$\text{normalize } N_j \text{ as: } N_j = N_j \frac{\langle C_i, N_i \rangle}{\langle C_{i-1}, N_{i-1} \rangle}$$

(We take $\langle C_0, N_0 \rangle = 1$.) The above algorithm needs the vector N_i in the i-th iteration to compute the cycle C_i. Instead of computing N_i from scratch in the i-th iteration, it obtains N_i by update and normalization steps through iterations 1 to $i-1$. We describe how to compute a shortest cycle C_i such that $\langle C_i, N_i \rangle \neq 0$ in Section 5. Let us now show that the N_i's obey the prefix orthogonality property. Lemma 1, proved in [12], shows that and more.

Lemma 1. *For any i, at the end of iteration $i-1$, the vectors $N_i, \ldots, N_d$ are orthogonal to $C_1, \ldots, C_{i-1}$ and moreover, for any j with $i \leq j \leq d$, $N_j = \langle N_{i-1}, C_{i-1} \rangle (x_{j,1}, \ldots, x_{j,i-1}, 0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 occurs in the j-th coordinate and $\mathbf{x} = (x_{j,1}, \ldots, x_{j,i-1})$ is the unique solution to the set of equations:*

$$\begin{pmatrix} \tilde{C}_1 \\ \vdots \\ \tilde{C}_{i-1} \end{pmatrix} \mathbf{x} = \begin{pmatrix} -c_{1j} \\ \vdots \\ -c_{(i-1)j} \end{pmatrix}. \tag{1}$$

Here $\tilde{C}_k$, $1 \leq k < i$, is the restriction of C_k to its first $i-1$ coordinates and c_{kj} is the j-th coordinate of C_k.

Remark. Note that the i-th coordinate of N_i is non-zero. This readily implies that there is at least one cycle that has non-zero dot product with N_i, namely the fundamental cycle F_{e_i} formed by the edge e_i and the path in the spanning tree T connecting its endpoints. The dot product $\langle F_{e_i}, N_i \rangle$ is equal to the i-th coordinate of N_i, which is non-zero.

We next give an alternative characterization of these N_j's. This characterization helps us in bounding the running time of the algorithm Deterministic-MCB. Let M denote the $(i-1) \times (i-1)$ matrix of $\tilde{C}_k$'s in Equation (1) and b_j denote the column vector of $-c_{kj}$'s on the right. We claim that solving $M\mathbf{x} = \det(M) \cdot b_j$ leads to the same vectors N_j for all j with $i \leq j \leq d$. First, note that $\langle N_{i-1}, C_{i-1} \rangle = \det(M)$, it is easy to show this.

By Lemma 1, $(x_{j,1}, \ldots, x_{j,i-1})$ is the unique solution to $M\mathbf{x} = b_j$. Hence $\det(M)(x_{i,1}, \ldots, x_{i,i-1}, 1, 0, \ldots)$, which is N_i in the i-th iteration (that is when it is used in the algorithm to compute the cycle C_i), could have obtained directly in the i-th iteration by solving the set of equations $M\mathbf{x} = \det(M) b_i$ and appending $(\det(M), 0, \ldots, 0)$ to $\mathbf{x}$. However, such an algorithm would be slower - it would take time $\tilde{\Theta}(m^{\omega+2})$, where $\omega < 2.376$ is the exponent of matrix multiplication. The updates and normalizations in the algorithm Deterministic-MCB achieve the same result in a more efficient manner.

Let us now bound the running time of the i-th iteration of Deterministic-MCB. We will show in Section 5 that a shortest cycle C_i such that $\langle C_i, N_i \rangle \neq 0$ can be computed in $O(m^2 n + mn^2 \log n)$ time. Let us look at bounding the time taken for update and normalization steps. We take $O(m)$ arithmetic steps for updating and scaling each N_j since each N_j has m coordinates. Thus the total number of arithmetic operations in the i-th iteration is $O((d-i)m) = O(md)$ over all j, $i+1 \leq j \leq d$. We next estimate the cost of arithmetic. The coordinates of N_j are determined by the system $M\mathbf{x} = \det(M)b_j$ and hence are given by Cramer's rule. Fact 1, which follows from Hadamard's inequality, shows that each entry in N_j is bounded by $d^{d/2}$. Thus we pay $\tilde{O}(d)$ time per arithmetic operation. Thus the running time of the i-th iteration is $\tilde{O}(m^3)$ and hence the running time of Deterministic-MCB is $\tilde{O}(m^4)$.

Fact 1. *Since M is a $\pm 1, 0$ matrix of size $(i-1) \times (i-1)$ and b_j is a $\pm 1, 0$ vector, all determinants used in Cramer's Rule are bounded by $i^{i/2}$. Therefore, the absolute value of each entry in N_j, where $j \geq i$, is bounded by $i^{i/2}$.*

4 A Faster Deterministic Algorithm

The update and normalization steps form the bottleneck in Deterministic-MCB. We will reduce their cost from $\tilde{O}(m^4)$ to $\tilde{O}(m^{\omega+1})$.

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

The Scheme. The faster deterministic algorithm starts with the same configuration for the N_i's as before, i.e., N_i is initialized to the i-th unit vector, $1 \leq i \leq d$. 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, leaving $N_{\lfloor d/2 \rfloor + 1}, \ldots, N_d$ at their initial values. Second, it runs a bulk update step in which $N_{\lfloor d/2 \rfloor + 1}, \ldots, N_d$ are modified so that they become 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. Such a scheme was used earlier in [13,11].

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 $C_1, \ldots, C_{\lfloor d/2 \rfloor}$ that we achieved in the bulk update step? The trick lies in the fact that whenever we update any $N_j \in \{N_{\lfloor d/2 \rfloor + 1}, \ldots, N_d\}$ in the second recursive call, we do it as $N_j = \sum_{k=\lfloor d/2 \rfloor + 1}^{d} \alpha_k N_k, \alpha_k \in \mathbb{Q}$. That is, the updated N_j is obtained as a rational linear combination of $N_{\lfloor d/2 \rfloor + 1}, \ldots, N_d$. Since the bulk update step prior to the second recursive call ensures that $N_{\lfloor d/2 \rfloor + 1}, \ldots, N_d$ are all orthogonal to $C_1, \ldots, C_{\lfloor d/2 \rfloor}$ at the beginning of this step, the updated N_j's remain orthogonal

to $C_1, \ldots, C_{\lfloor d/2 \rfloor}$. 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 (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 5. As regards time complexity, the bulk update step will be shown to take $O(md^{\omega-1})$ arithmetic operations.

4.1 The Bulk Update Procedure

We describe the bulk update procedure in the recursive call that computes the cycles $C_\ell, \ldots, C_h$ for some h and ℓ with $h > \ell$. This recursive call works with the vectors $N_\ell, \ldots, N_h$: all these vectors are already orthogonal to $C_1, \ldots, C_{\ell-1}$. The recursive call runs as follows:

1. compute the cycles $C_\ell, \ldots, C_{mid}$ where $mid = \lceil (\ell + h)/2 \rceil - 1$, using the vectors $N_\ell, \ldots, N_{mid}$, recursively.
2. modify $N_{mid+1}, \ldots, N_h$, which are untouched by the first step, to make them orthogonal to $C_\ell, \ldots, C_{mid}$.
3. compute $C_{mid+1}, \ldots, C_h$ using these $N_{mid+1}, \ldots, N_h$, recursively.

Step 2 is the bulk update step. We wish to update each N_j, $mid+1 \leq j \leq h$, to a rational linear combination of $N_\ell, \ldots, N_{mid}$ and N_j as follows[1]:

$$N_j = \frac{\langle N_{mid}, C_{mid} \rangle}{\langle N_{\ell-1}, C_{\ell-1} \rangle} N_j + \sum_{t=\ell}^{mid} \alpha_{tj} N_t$$

where the α_{tj}'s are to be determined in a way which ensures that N_j becomes orthogonal to $C_\ell, \ldots, C_{mid}$. That is, we want for all i, j, where $\ell \leq i \leq mid$ and $mid + 1 \leq j \leq h$,

$$\frac{\langle N_{mid}, C_{mid} \rangle}{\langle N_{\ell-1}, C_{\ell-1} \rangle} \langle C_i, N_j \rangle + \sum_{t=\ell}^{mid} \alpha_{tj} \langle C_i, N_t \rangle = 0. \tag{2}$$

Rewriting this in matrix form, we get $A \cdot \mathcal{N}_d \cdot D = -A \cdot \mathcal{N}_u \cdot X$, where (let $k = mid - \ell + 1$)

- A is a $k * m$ matrix, the i-th row of which is $C_{\ell+i-1}$,
- $\mathcal{N}_d$ is an $m * (h-k)$ matrix, the j-th column of which is N_{mid+j},
- D is an $(h-k) * (h-k)$ diagonal matrix with $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ in the diagonal,
- $\mathcal{N}_u$ is an $m * k$ matrix, the t-th column of which is $N_{\ell+t-1}$,
- X is the $k * (h-k)$ matrix of variables α_{tj}, with t indexing the rows and j indexing the columns.

[1] Note that the coefficient $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ for N_j is chosen so that the updated vector N_j here is exactly the same vector N_j that we would have obtained at this stage using the algorithm Deterministic-MCB (Section 3).

To compute the α_{tj}'s, we solve for $X = -(A \cdot \mathcal{N}_u)^{-1} \cdot A \cdot \mathcal{N}_d \cdot D$. Using fast matrix multiplication, we can compute $A \cdot \mathcal{N}_u$ and $A \cdot \mathcal{N}_d$ in $O(mk^{\omega-1})$ time, by splitting the matrices into d/k square blocks and using fast matrix multiplication to multiply the blocks. Multiplying each element of $A \cdot \mathcal{N}_d$ with the scalar $\langle N_{mid}, C_{mid}\rangle / \langle N_{\ell-1}, C_{\ell-1}\rangle$ gives us $A \cdot \mathcal{N}_d \cdot D$. Thus we compute the matrix $A \cdot \mathcal{N}_d \cdot D$ with $O(mk^{\omega-1})$ arithmetic operations. Next, we find the inverse of $A \cdot \mathcal{N}_u$ with $O(k^\omega)$ arithmetic operations (this inverse exists because $A \cdot \mathcal{N}_u$ is a lower triangular matrix whose diagonal entries are $\langle C_i, N_i\rangle \neq 0$). Then we multiply $(A \cdot \mathcal{N}_u)^{-1}$ with $A \cdot \mathcal{N}_d \cdot D$ with $O(k^\omega)$ arithmetic operations. Thus we obtain X. Finally, we obtain $N_{mid+1}, \ldots, N_d$ from X using the product $\mathcal{N}_u \cdot X$, which we can compute in $O(mk^{\omega-1})$ arithmetic operations, and adding $\mathcal{N}_d \cdot D$ to $\mathcal{N}_u \cdot X$. The total number of arithmetic operations required for the bulk update step is thus $O(mk^{\omega-1})$.

What is the cost of the arithmetic? In the algorithm presented above, the entries in $(A \cdot \mathcal{N}_u)^{-1}$ could be very large. The elements in $A \cdot \mathcal{N}_u$ have values up to $d^{\Theta(d)}$, which would result in the entries in $(A \cdot \mathcal{N}_u)^{-1}$ being as large as $d^{\Theta(d^2)}$. So each arithmetic operation then costs us up to $\tilde{\Theta}(d^2)$ time and the overall time for the outermost bulk update step would be $\tilde{\Theta}(m^{\omega+2})$ time, which makes this approach slower than the algorithm Deterministic-MCB.

The good news is that the numbers α_{tj}'s are just *intermediate* numbers in our computation. That is, they are the coefficients in

$$\sum_{t=\ell}^{mid} \alpha_{tj} N_t + \frac{\langle N_{mid}, C_{mid}\rangle}{\langle N_{\ell-1}, C_{\ell-1}\rangle} N_j.$$

Our final aim is to determine the updated coordinates of N_j which are at most $d^{d/2}$ (refer Fact 1), since we know $N_j = (y_1, \ldots, y_{mid}, 0, \ldots, \langle N_{mid}, C_{mid}\rangle, \ldots, 0)$, where $\mathbf{y} = (y_1, \ldots, y_{mid})$ is the solution to the linear system: $M\mathbf{y} = \det(M) b_j$; M is the $mid \times mid$ matrix of $C_1, \ldots, C_{mid}$ truncated to their first mid coordinates and b_j is the column vector of negated j coordinates of $C_1, \ldots, C_{mid}$. Since the final coordinates are bounded by $d^{d/2}$ while the intermediate values could be much larger, this suggests the use of modular arithmetic here. 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 $N_j \bmod p_1, \ldots, N_j \bmod p_s$, which is possible (by the Chinese Remainder Theorem) if $s \approx d/2$. Arithmetic in $\mathbb{F}_p$ takes $O(1)$ time and we thus spend $O(smk^{\omega-1})$ 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 $A \cdot \mathcal{N}_u$ in the field $\mathbb{F}_p$. Since each number $\langle N_i, C_i\rangle$ could be as large as $d^{d/2+1}$, it could be a multiple of up to $\Theta(d)$ primes which are in the range $d, \ldots, d^2$. So in order to be able to determine d primes which are relatively prime to each of $\langle N_\ell, C_\ell\rangle, \ldots, \langle N_{mid}, C_{mid}\rangle$, we might in the worst case have to test about $(mid - \ell + 1) \cdot d = kd$ primes. Testing kd primes for divisibility w.r.t. k d-bit numbers costs us $k^2 d^2$ time. 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 $\mathbb{F}_q$ without any problem. 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 which will result in a total time of $\tilde{O}(m^{\omega+1})$ for all the update steps, which is our goal. But computing such a large prime q is a difficult problem.

The solution is to work over a suitable ring instead of over a field; note that fast matrix multiplication algorithms work over rings. 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 much larger than any coordinate of N_j.

What properties do we need of R? The integer R must be relatively prime to the numbers: $\langle N_\ell, C_\ell\rangle$, $\langle N_{\ell+1}, C_{\ell+1}\rangle$, ..., $\langle N_{mid}, C_{mid}\rangle$ so that that triangular matrix $A \cdot \mathcal{N}_d$ 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 $\mathbb{Z}_R$. Once we determine such an R, we will work in $\mathbb{Z}_R$. We stress the point that such an R is a number used only in this particular bulk update step - in another bulk update step of another recursive call, we need to compute another such large integer.

It is easy to see that the number R determined below is a large number that is relatively prime to $\langle N_{\ell-1}, C_{\ell-1}\rangle$, $\langle N_\ell, C_\ell\rangle$, $\langle N_{\ell+1}, C_{\ell+1}\rangle$, ..., and $\langle N_{mid}, C_{mid}\rangle$.

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: $P_1 = p_1 \cdots p_d$, $P_2 = p_1 \cdots p_{2d}$, $P_3 = p_1 \cdots p_{3d}$, ..., $P_d = p_1 \cdots p_{d^2}$.
2. Then during our current update step, compute 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 $s \geq 0$ such that P_{s+1} does not divide $\mathcal{L}$.
4. Determine a $p \in \{p_{sd+1}, \ldots, p_{sd+d}\}$ that does not divide $\mathcal{L}$. Compute $R = p^d$.

Cost of computing R: The value of $\pi(r)$, the number of primes less than r, is given by $r/6 \log r \leq \pi(r) \leq 8r/\log r$ [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 using a sieving algorithm. The products $P_1, \ldots, P_d$ are computed just once in a preprocessing step. 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. So computing $P_1, \ldots, P_d$ takes $\tilde{O}(d^3)$ preprocessing time.

In the update step, we compute $\mathcal{L}$, which takes $\tilde{O}(d^2)$ time. The product $p_{sd+1} \cdots p_{sd+d}$ is found in $\tilde{O}(d^2)$ time by binary search. Determine a $p \in \{p_{sd+1}, \ldots, p_{sd+d}\}$ that does not divide $\mathcal{L}$ by testing which of the two products: $p_{sd+1} \cdots p_{sd+\lfloor d/2 \rfloor}$ or $p_{sd+\lfloor d/2 \rfloor+1} \cdots p_{sd+d}$ 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 $A \cdot \mathcal{N}_u$ in the ring $\mathbb{Z}_R$. Recall that this matrix is lower triangular. Computing the inverse of a lower triangular matrix is easy. If

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

Hence to invert $A \cdot \mathcal{N}_u$ 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 $A \cdot \mathcal{N}_u$ is inverted via fast matrix multiplication and once we compute $(A \cdot \mathcal{N}_u)^{-1}$, the matrix X, that consists of all the coordinates α_{tj} that we need (refer Equation (2)), can be easily computed in $\mathbb{Z}_R$ as $-(A \cdot \mathcal{N}_u)^{-1} \cdot A \cdot \mathcal{N}_d \cdot D$ by fast matrix multiplication. Then we determine all $N_j \bmod R$ for $mid + 1 \leq j \leq h$ from $\mathcal{N}_u \cdot X + \mathcal{N}_d \cdot D$. It follows from the discussion presented at the beginning of Section 4.1 that the time required for all these operations is $\tilde{O}(m^2 k^{\omega-1})$ since each number is now bounded by d^d.

Retrieving the actual N_j. 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 $N_j \bmod R$ 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 $N_j \bmod R$ in $O(d^2)$ time. The time complexity for the update step, which includes matrix operations, gcd computations and other arithmetic, is $\tilde{O}(m^2 k^{\omega-1} + d^2 k)$ or $\tilde{O}(m^2 k^{\omega-1})$. Thus our recurrence becomes $T(k) = 2T(k/2) + \tilde{O}(m^2 k^{\omega-1})$ when $k > 1$. We shall show the following lemma in the next section.

Lemma 2. *A shortest cycle C_i such that $\langle C_i, N_i \rangle \neq 0$ can be computed in $O(m^2 n + mn^2 \log n)$ time.*

Thus $T(1) = O(m^2 n + mn^2 \log n)$. Our recurrence solves to $T(k) = O(k(m^2 n + mn^2 \log n) + k^\omega m^2 \cdot \text{poly}(\log m))$ and hence $T(d) = O(m^3 n + m^2 n^2 \log n) + \tilde{O}(m^{\omega+1})$, 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 $o(m^3 n)$. We can 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 $O(m^3 n + m^2 n^2 \log n)$ time.*

5 Computing Non-orthogonal Shortest Cycles

Now we come to the second key routine required by our algorithm - given a directed graph G with non-negative edge weights, compute a shortest cycle in G whose dot product with a given vector $N \in \mathbb{Z}^m$ is non-zero. We will first consider the problem of computing a shortest cycle C_p such that $\langle C_p, N \rangle \neq 0 \pmod p$ for a number $p = O(d \log d)$. Recall that C_p can traverse edges of G in both forward and reverse directions; the vector representation of C_p 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 dot products with N. The weight of C_p itself is simply the sum of the weights of the edges in the cycle. We show how to compute C_p in $O(mn + n^2 \log n)$ time.

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 dot product between the vector representation of this path or cycle 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 $O(mn + n^2 \log n)$ time. 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 a 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 3. *Let $C = u_0u_1 \dots u_ku_0$ be a shortest cycle whose residue class is non-zero modulo p and whose shortest edge is u_0u_1. Then the path $u_1u_2 \dots u_ku_0$ has a residue class different from the residue class of the edge u_1u_0 and the length of the path $u_1u_2 \dots u_ku_0$ equals $s_{u_1u_0}$ and the length of the edge u_0u_1 equals $f_{u_1u_0}$.*

Proof. First, we show that the path $u_1u_2 \dots u_ku_0$ and the edge u_1u_0 have different residue classes. Let x denote the residue class of the path and y denote the residue class of the edge u_0u_1. Since C is in a non-zero residue class, $x + y \not\equiv 0 \pmod{p}$, so $x \not\equiv -y \pmod{p}$. Since the incidence vector corresponding to u_1u_0 is the negation of the incidence vector corresponding to u_0u_1, the residue class of the edge u_1u_0 is $-y$. Thus 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 $f_{u_1u_0}$). Combining π with u_1u_0 yields a cycle and combining π with $u_1u_2 \dots u_ku_0$ yields another cycle. These cycles are in distinct residue classes and are shorter than C. This contradicts the definition of C. Therefore, the edge u_1u_0 has length $f_{u_1u_0}$.

Since $u_1u_2 \dots u_ku_0$ has a different residue class from the edge u_1u_0, the length of $u_1u_2 \dots u_ku_0$ cannot be smaller than $s_{u_1u_0}$, by the very definition of $s_{u_1u_0}$. Suppose, for a contradiction that the length $u_1u_2 \dots u_ku_0$ is strictly larger than $s_{u_1u_0}$. Then combining the path between u_1 and u_0 which realizes the length $s_{u_1u_0}$ along with the edge u_1u_0 yields a cycle which is shorter than C and which has a non-zero residue class modulo p. This contradicts 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 by Dijkstra's algorithm; this gives us p_{uv}, f_{uv}, and r_{uv}, for each pair u, v. The time taken is $O(mn + n^2 \log n)$. 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 $O(mn + n^2 \log n)$ time. The following lemma shows the key prefix property of the q_{uv} paths needed for a Dijkstra-type algorithm.

Lemma 4. *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}, i.e., it has length s_{uv} and residue class distinct from r_{uv}. Let w be the penultimate vertex on this path and let π' be the prefix path from u to w. Clearly, π cannot be shorter than $p_{uw} \circ wv$. Hence, if the residue class of $p_{uw} \circ wv$ is distinct from r_{uv}, we are done. So assume that $p_{uw} \circ wv$ has residue class r_{uv}. Then π' must have a residue class distinct from p_{uw} and hence q_{uw} exists. Also, the length of π' must be at least the length of q_{uw} and the residue class of $q_{uw} \circ wv$ is distinct from the residue class of $p_{uw} \circ wv$ and hence distinct from r_{uv}. Thus $q_{uw} \circ wv$ realizes s_{uv}. □

We now show how to compute the s_{uv}'s for any fixed u in time $O(m+n \log n)$ with a Dijkstra-type 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 set key_{uv} to the minimal length of any path $p_{uw} \circ wv$ with residue class distinct from r_{uv}. If there is no such path, we set it to ∞.

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 different 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 does not need to worry about the residue class since any path that updates key_{uv} in the course of the algorithm has residue class different from r_{uv}). This follows immediately from Lemma 4. If s_{uv} is realized by the path $p_{uw} \circ wv$ for some neighbor w, then key_{uv} is set to s_{uv} in the initialization step. If s_{uv} is realized by the path $q_{uw} \circ wv$ for some neighbor w, then key_{uv} is set to s_{uv} in the update step. This completes the proof of correctness.

Thus we have given an $O(mn + n^2 \log n)$ algorithm to compute a shortest cycle C_p whose dot product with N is non-zero modulo p. A slower algorithm with running time $O(mn \log n)$, which computes a layered graph, was given in [11] to compute such a cycle C_p. Using the algorithm described here instead of this slower algorithm results in a randomized algorithm with running time $O(m^2 n + mn^2 \log n)$ for the minimum cycle basis problem in directed graphs. We state this result as the following theorem.

Theorem 3. *A minimum cycle basis in a directed graph G can be computed with probability at least $3/4$ in $O(m^2 n + mn^2 \log n)$ time.*

The original problem. Our original problem here was to compute a shortest cycle C such that $\langle C, N \rangle \neq 0$. Any cycle C which satisfies $\langle C, N \rangle \neq 0$ satisfies

$\langle C, N\rangle \neq 0 \pmod{p}$ 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 $\mathbb{Z}_{\prod p_i}$ and $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_d}$. We have $|\langle C, N\rangle| \leq \|N\|_1 \leq d \cdot d^{d/2} < \prod_{i=1}^{d} p_i$. So if $\langle C, N\rangle$ is non-zero, then it is a non-zero element in $\mathbb{Z}_{\prod p_i}$ and so it satisfies $\langle C, N\rangle \neq 0 (\bmod p)$ for some p in $\{p_1, \ldots, p_d\}$. Thus a shortest cycle C such that $\langle C, N\rangle \neq 0$ is the shortest among all the cycles C_p, $p \in \{p_1, \ldots, p_d\}$, where C_p is a shortest cycle such that $\langle C_p, N\rangle \neq 0 \pmod{p}$. Hence the time taken to compute C is $O(d \cdot (mn + n^2 \log n))$ or $O(m^2 n + mn^2 \log n)$. This completes the proof of Lemma 2.

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, 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 Bases in Directed Graphs. Full version to appear in special issue of *TOCS*, 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 Bases 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. Information Processing Letters, 94(3): 107-112, 2005.