SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1430–1447

FASTER ALGORITHMS FOR MINIMUM CYCLE BASIS IN DIRECTED GRAPHS[*]

RAMESH HARIHARAN[†], TELIKEPALLI KAVITHA[‡], AND 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 nonnegative 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. We present an $O(m^3n + m^2n^2\log n)$ algorithm. We obtain our algorithm 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 nonzero. We also present a simple $O(m^2n + mn^2\log n)$ Monte Carlo algorithm. The problem of computing a minimum cycle basis in an *undirected* graph has been well studied. In this problem a $\{0,1\}$ edge incidence vector is associated with each cycle and the vector space over $\mathbb{Z}_2$ generated by these vectors is the cycle space of the graph. The fastest known algorithm for computing a minimum cycle basis in an undirected graph runs in $O(m^2n + mn^2\log n)$ time and our randomized algorithm for directed graphs matches this running time.

Key words. cycle basis, fast matrix multiplication, randomization, shortest paths

AMS subject classifications. 68W20, 68W40

DOI. 10.1137/060670730

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.[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* 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 nonnegative 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.

[*]Received by the editors September 26, 2006; accepted for publication (in revised form) April 24, 2008; published electronically September 8, 2008. Preliminary versions of this work appeared in [14, 16].
http://www.siam.org/journals/sicomp/38-4/67073.html

[†]Strand Life Sciences, 237, Sir C. V. Raman Avenue, Sadashivnagar, Bangalore 560080, India (ramesh@strandls.com).

[‡]Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (kavitha@csa.iisc.ernet.in).

[§]Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, Saarbrücken 66123, Germany (mehlhorn@mpi-sb.mpg.de).

[1]Formally, the incidence vector is determined only up to a factor ± 1 as either direction (clockwise or counterclockwise) could be chosen to traverse a cycle.

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 $\mathbb{Z}_2$. The minimum cycle basis problem in an undirected graph G asks for the cycle basis of minimum weight in G.

The different possible settings for the minimum cycle basis problem are (i) the minimum cycle basis problem in undirected graphs, (ii) the minimum cycle basis problem in directed graphs (where the directions on the cycles are ignored), and (iii) the minimum *directed cycle* basis problem in directed graphs, where the cycle basis has to consist of cycles that traverse edges only along the direction of the edge.

We first compare problems (i) and (ii) and then discuss problem (iii). Problems (i) and (ii) are essentially different, since the first problem deals with computing a minimum weight spanning set of cycles that is linearly independent over $\mathbb{Z}_2$ while the second problem needs to compute a minimum weight spanning set of cycles that is linearly independent over $\mathbb{Q}$. Transforming cycles in a cycle basis of a directed graph by replacing both -1 and 1 by 1 does not necessarily yield a basis for the underlying undirected graph, since the given cycle basis could be linearly independent over $\mathbb{Q}$ but linearly dependent over $\mathbb{Z}_2$. 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 presented in [22], which we include in section 2. Thus, one cannot find a minimum cycle basis for a directed graph by simply working with the underlying undirected graph.

A *directed cycle* basis is a spanning set of cycles where the incidence vector of each cycle in this basis is a vector in $\{0,1\}^m$, that is, each edge in a cycle here is traversed in the right direction. Note that a directed graph need not admit a directed cycle basis. Berge [2] studied the question of when a directed graph G admits such a cycle basis. He showed that if G is strongly connected, then G admits a directed cycle basis. Conversely, he showed that if G admits a directed cycle basis, then each maximal weakly connected induced subgraph of G with no cut vertex has to be strongly connected or a single arc.

Efficient algorithms for computing minimum cycle bases in the above settings have several applications. The minimum cycle basis problem is a special case of the *null space problem*. The null space problem is defined as follows: given a field $\mathbb{F}$ and an $n \times m$ matrix A with $n \leq m$, rank r, and entries in $\mathbb{F}$, find a matrix with the fewest nonzeros, whose columns span the null space of A. The null space problem was studied by Coleman and Pothen [6, 7], and this problem is NP-hard in general. The minimum cycle basis problem in undirected graphs with unit weights on the edges arises when the underlying field is $\mathbb{Z}_2$ and A is the $\{0,1\}$ vertex-edge incidence matrix of an undirected graph $G(V,E)$. The null vectors of this A are the vectors $(x_e)_{e\in E} \in \mathbb{Z}_2^{|E|}$ such that for each $v \in V$ we have $\sum_{e\in\delta(v)} x_e = 0 \pmod{2}$, where $\delta(v)$ is the set of edges incident on v. Note that the set of such vectors $(x_e)_{e\in E}$ is the cycle space of G, and a solution to the null space problem is a minimum cycle basis of G.

Similarly, the minimum cycle basis problem in directed graphs with unit weights on the edges is an instance of the null space problem when the underlying field is $\mathbb{Q}$ and A is the $\{-1,0,1\}$ vertex-edge incidence matrix of a directed graph $G(V,E)$. The null vectors of such an A are the vectors $(x_e)_{e\in E} \in \mathbb{Q}^{|E|}$ such that for each $v \in V$ we have $\sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e = 0$, where $\delta^+(v)$ and $\delta^-(v)$ are the set of edges leaving v and entering v, respectively. It is easy to see that the set of such vectors $(x_e)_{e\in E}$ is the cycle space of G. The minimum cycle basis of G is a solution to this null space problem. Indeed, assume that B is a minimum solution to the null space problem. We may assume that the entries of B are integral, as multiplication by a suitable constant makes all entries integral and does not change the number of nonzero entries. So assume that B contains a column C whose entries are not in $\{0, \pm 1\}$. Since C belongs to the null space of the vertex-edge incidence matrix A, C decomposes into a set of simple cycles C_1 to C_k. Each C_i uses a subset (not necessarily proper) of the edges of C and $B \setminus C \cup C_i$ is a basis for some i. We conclude that there exists a solution with $0, \pm 1$ entries to this null space problem, which implies that a minimum cycle basis of the directed graph G is a solution to this null space problem.

Some of the applications of minimum cycle bases arise from the above characterization of minimum cycle bases as the solutions of the null space problem in graphs. The cycle analysis of electrical networks [9] corresponding to Kirchoff's law is such an example. Applications of minimum cycle bases have also been shown in structural engineering [5], chemistry and biochemistry [11, 19], and surface reconstruction from point clouds [23]. The minimum directed cycle basis has applications in metabolic flux analysis [12]. A cycle basis of minimum weight in a directed graph whose $d \times m$ cycle-edge incidence matrix satisfies the constraint that all its regular $d \times d$ submatrices have determinant ± 1 has been found to be very useful in cyclic timetabling [20, 21]. Books by Deo [10] and Bollobás [4] have in-depth coverage of cycle bases.

Horton [15] designed the first polynomial time algorithm to compute a minimum cycle basis in an undirected graph and there are now several polynomial time algorithms for this problem [3, 9, 13, 18], the fastest running in $O(m^2 n + mn^2 \log n)$ time [18]. Gleiss, Leydold, and Stadler [12] used Berge's characterization of directed cycle bases and showed that a generalization of Horton's minimum cycle basis algorithm in undirected graphs computes a minimum directed cycle basis in strongly connected directed graphs. The first polynomial time algorithm for computing a minimum cycle basis in a directed graph had a running time of $\tilde{O}(m^4 n)$ [17]. Liebchen and Rizzi [22] gave an $\tilde{O}(m^{\omega+1} n)$ algorithm for this problem, where $\omega < 2.376$ [8] is the exponent of matrix multiplication; this was the fastest deterministic algorithm so far for this problem in directed graphs.

In this paper we present an $O(m^3 n + m^2 n^2 \log n)$ deterministic algorithm and an $O(m^2 n + mn^2 \log n)$ Monte Carlo algorithm to compute a minimum cycle basis in a directed graph G with m edges, n vertices, and nonnegative edge weights. 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, 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 $\mathbb{Q}$, which seems to necessitate the handling of large numbers. Also, the computation of a shortest cycle that has a nonzero dot product with a given vector seems more difficult in directed graphs than in undirected graphs.

The framework used in our algorithms was introduced by de Pina [9] and was also used in [3, 18, 17]: we compute cycles C_i and supporting vectors N_i so that each C_i is a 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 algorithms for computing the C_i's and N_i's rest on two ideas.

First, we show how to compute the vectors N_i efficiently using fast matrix multiplication and inversion. Our basic algorithm updates all vectors N_j with $j > i$ in iteration i, which results in an $\tilde{O}(m^4)$ algorithm. The improvement rests on an idea already used in [18] to delay the update of vectors with higher index and to perform these updates in bulk using matrix multiplication and inversion. However, this creates a problem since the numbers involved in the arithmetic get very large. We show two approaches to solving this problem, leading to an $O(m^2n + mn^2 \log n)$ randomized algorithm and an $O(m^3n + m^2n^2 \log n)$ deterministic algorithm. In the randomized algorithm we work over a finite field $\mathbb{Z}_p$ for a small prime p, which ensures that the numbers involved here do not get large. We show that we compute a minimum cycle basis with a probability of at least $3/4$ when p is a prime chosen uniformly at random from a set of d^2 small primes. In the deterministic algorithm, we could run into intermediate numbers whose bit size is $\tilde{\Theta}(m^2)$ when we use fast matrix inversion algorithms. This makes the running time of our algorithm $\tilde{\Theta}(m^{\omega+2})$, which is worse than the original iterative algorithm. We circumvent the problem by working over a suitable ring $\mathbb{Z}_R$. 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 now is $\tilde{O}(m^{\omega+1})$.

The second key step in our algorithm is a subroutine to compute a shortest cycle whose dot product with a given vector N_i is nonzero modulo a small number p. 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 vertices and the second is a shortest path whose residue class is different from the residue class of the first path. Thus this yields an $O(m^2n + mn^2 \log n)$ algorithm over $\mathbb{Z}_p$ for computing the d cycles. For the deterministic algorithm, the computation of each cycle can be reduced via the Chinese remainder theorem to computing a shortest cycle C_p whose inner product with N_i is nonzero modulo p for some $p \in \{p_1, \ldots, p_d\}$, which is a collection of small primes. This procedure is repeated for each $p \in \{p_1, \ldots, p_d\}$ yielding $O(m^2n+mn^2 \log n)$ time for each C_i and thus $O(m^3n + m^2n^2 \log n)$ time overall for all of the d cycles.

Organization of this paper. In section 2 we discuss some preliminaries and describe the examples from [22] that were mentioned in section 1. Our framework is given in section 3, and we present a simple deterministic algorithm from [17] that follows from this framework. Section 4 lays the approach for a faster scheme and shows the problem of large numbers that such an approach runs into. Section 5 describes the deterministic algorithm that overcomes this problem, and section 6 gives a randomized algorithm. Section 7 describes the subroutine for computing the required 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[2] of the cycle space of G. So a minimum cycle basis of G consists of d cycles $C_1, \ldots, C_d$. We describe cycles by their incidence vectors in $\{-1, 0, +1\}^m$.

A cycle basis of a directed graph need not project onto an undirected cycle

[2]Fix any spanning tree of G. For a nontree edge e, the fundamental cycle F_e induced by e consists of e plus the tree path connecting its endpoints. This set of cycles is clearly independent as every nontree edge is contained in a single cycle. We need to show that it spans all cycles. Consider any cycle $C = (x_e)_{e \in E}$ and define the sum $S = \sum_{e \text{ is a nontree edge}} x_e F_e$, in which every fundamental cycle is multiplied with the multiplicity of its defining edge in C. The vector S is in the cycle space and so is $Z = S - C$. The entries of Z corresponding to nontree edges are zero (by the definition of S) and Z satisfies the flow conservation constraints. Hence the entries of Z corresponding to tree edges must also be zero. Thus $Z = 0$ and the fundamental cycles form a basis. The number of fundamental cycles is exactly $m - n + 1$.

basis. Consider the following three 4-cycles in the directed graph in Figure 2.1: $C_1 = (e_1, e_2, e_3, e_4)$, $C_2 = (e_1, e_6, e_3, e_5)$, 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 $(0, 1, 0, -1, -1, 1)$. It is easy to see that these vectors 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 modulo 2 of the other two, so C_1, C_2, C_3 do not form a cycle basis for the undirected K_4.

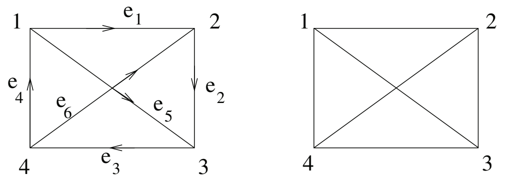

FIG. 2.1. *Directed K_4 and the underlying undirected graph.*

Further, there are directed weighted graphs in which the minimum cycle basis has lower weight than any cycle basis of the underlying undirected graph; such an example was given in [22]. Consider the generalized Petersen graph $P_{7,2}$ in Figure 2.2.

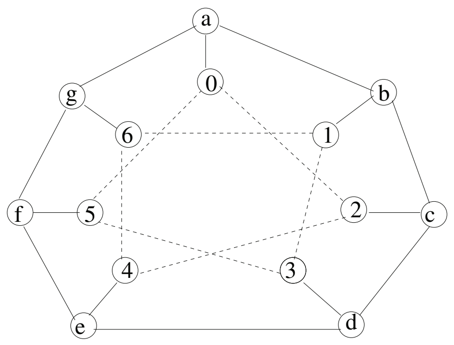

FIG. 2.2. *The generalized Petersen graph $P_{7,2}$.*

Call an edge (u, v) an *inner edge* if $\{u, v\} \subset \{0, 1, \ldots, 6\}$. Similarly call an edge (u, v) an *outer edge* if $\{u, v\} \subset \{a, \ldots, g\}$. The seven edges that remain are called *spokes*. Assign weight two to the seven inner edges and weight three to the outer edges and spokes. The shortest cycle in this graph has a weight of 14 and there are precisely eight cycles having a weight of 14, namely the cycle C_I consisting of only inner edges, and the seven cycles using one inner edge, two spokes, and two outer edges. Use C_i, $0 \leq i \leq 6$, to denote the cycle using the inner edge connecting i and $i + 2 \bmod 7$. Every other cycle has a length of at least 15.

Every edge of $P_{7,2}$ belongs to precisely two of the eight cycles with a weight of 14. Therefore in the undirected case, these $8 = m - n + 1$ cycles are not independent over $\mathbb{Z}_2$. Thus in the undirected case, every cycle basis has a weight of at least 113. In the directed case, these 8 cycles under any orientation of edges are linearly independent.[3] So there is a directed cycle basis of weight 112.

[3]Direct the inner edges clockwise, the spokes outward, and the outer edges counterclockwise. Then all eight cycles use their inner and outer edges in the forward direction. Assume $\alpha_I C_I + \sum_{0 \leq i \leq 6} \alpha_i C_i = 0$. The edge $(i, i + 2 \bmod 7)$ is used only by C_I and C_i, and hence we must have $\alpha_i = -\alpha_I$ for all i. But then the outer edges do not cancel out.

We will assume for the rest of this paper that the edges in $E = \{e_1, \ldots, e_m\}$ are ordered so that edges e_{d+1} to e_m form the edges of a spanning tree T of the underlying undirected graph. This means that the first d coordinates, of 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. So we will assume from now on that $m \leq n^2$.

3. Framework and a simple algorithm.

We begin with a structural characterization of a minimum cycle basis. This characterization uses auxiliary rational vectors $N_1, \ldots, N_d$ which serve as a scaffold for proving properties of $C_1, \ldots, C_d$, as described below. 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 3.1. *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: $\langle N_i, C_j \rangle = 0$ for all j, $1 \leq j < i$.*
2. *Nonorthogonality: $\langle N_i, C_i \rangle \neq 0$.*
3. *Shortness: C_i is a shortest cycle with $\langle N_i, C_i \rangle \neq 0$.*

Proof. First, we show that $C_1, \ldots, C_d$ is a cycle basis of G by showing that these are linearly independent over $\mathbb{Q}$ (recall that any set of d linearly independent cycles is a cycle basis). Suppose it is not. Then a rational linear combination of a subset of these cycles yields 0. Let the cycle with the 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 nonzero value on one side of this linear combination and a 0 on the other side, a contradiction.

Second, we show that $C_1, \ldots, C_d$ is a minimum cycle basis of G. Suppose it is not. Then consider the smallest $i \geq 1$ such that $C_1, \ldots, C_i$ are not in any minimum cycle basis. Then $C_1, \ldots, C_{i-1}$ belong to some minimum cycle basis; call this basis $\mathcal{K}$ (in the event that $i = 1$, let $\mathcal{K}$ be any minimum cycle basis). We will exhibit a cycle $K \in \mathcal{K}$ such that (i) $\langle N_i, K \rangle \neq 0$ and (ii) K can be written as a rational linear combination of C_i along with cycles in $\mathcal{K}/\{K\}$. Demonstrating such a cycle $K \in \mathcal{K}$ is easy: since $\mathcal{K}$ is a basis not containing C_i, the cycle C_i must be a rational linear combination 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. Therefore (i) holds. Further, (ii) follows by rewriting the above linear combination to switch the sides of C_i and K.

Property 1 of the theorem and condition (i) ensure that the cycle K is not one of $C_1, \ldots, C_{i-1}$. Condition (ii) implies that $\mathcal{K}/\{K\} \cup \{C_i\}$ is a cycle basis. Property 3 of the theorem and condition (i) 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. ◻

We present a simple deterministic algorithm from [17] that computes N_i's and C_i's satisfying the criteria in Theorem 3.1.

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 ith 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
          update $N_j$ as: $N_j = N_j - N_i \frac{\langle C_i, N_j\rangle}{\langle C_i, N_i\rangle}$,
          normalize $N_j$ 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 ith iteration to compute the cycle C_i. Instead of computing N_i from scratch in the ith 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 7. Let us now show that the N_i's obey the prefix orthogonality property. Lemma 3.2, proved in [17], shows this and more.

LEMMA 3.2. *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 jth coordinate and the vector $\mathbf{x} = (x_{j,1}, \ldots, x_{j,i-1})$ is the unique solution to the set of equations:*

$$\begin{pmatrix} \tilde{C}_1^T \\ \vdots \\ \tilde{C}_{i-1}^T \end{pmatrix} \mathbf{x} = \begin{pmatrix} -c_{1j} \\ \vdots \\ -c_{(i-1)j} \end{pmatrix}. \tag{3.1}$$

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 jth coordinate of C_k.

Proof. The claim is certainly true after the 0th iteration, that is, at the beginning of the algorithm. So consider the ith iteration and assume that the claim is true at the end of iteration $i-1$. In iteration i, we determine C_i with $\langle C_i, N_i\rangle \neq 0$ and then update the N_j's for all j with $i+1 \leq j \leq n$. Consider any j with $i+1 \leq j \leq n$ and use N_j' to denote the updated value of N_j.

N_j is updated by subtracting a scalar multiple of N_i from it. Since N_j and N_i are orthogonal to C_l for $l < i$ by induction hypothesis, N_j' is orthogonal to C_l. The update step also guarantees orthogonality to C_i. Indeed,

$$\langle C_i, N_j'\rangle = \langle C_i, N_j\rangle - \langle C_i, N_i\rangle \frac{\langle C_i, N_j\rangle}{\langle C_i, N_i\rangle} = 0.$$

By induction hypothesis, N_j is of the form $(t_{j,1}, \ldots, t_{j,i-1}, 0, \ldots, t_{j,j}, 0, \ldots)$, where $t_{j,j} = \langle C_{i-1}, N_{i-1}\rangle$ and N_i has nonzero entries only in its first i coordinates. So N_j' has the form $(t_{j,1}', \ldots, t_{j,i}', 0, \ldots, t_{j,j}, 0, \ldots)$. After normalization, the jth coordinate of N_j' is

$$t_{j,j} \frac{\langle N_i, C_i\rangle}{\langle N_{i-1}, C_{i-1}\rangle} = \langle N_{i-1}, C_{i-1}\rangle \frac{\langle N_i, C_i\rangle}{\langle N_{i-1}, C_{i-1}\rangle} = \langle N_i, C_i\rangle.$$

Hence N_j' has the form

$$N_j' = \langle N_i, C_i\rangle (u_{j,1}, \ldots, u_{j,i}, 0, \ldots, 1, 0, \ldots, 0).$$

Since N'_j is orthogonal to $C_1, \ldots, C_i$ and $\langle N_i, C_i \rangle \neq 0$, $(u_{j,1}, \ldots, u_{j,i})$ is a solution to the following set of equations:

$$\begin{pmatrix} \tilde{C}_1^T \\ \vdots \\ \tilde{C}_i^T \end{pmatrix} \mathbf{x} = \begin{pmatrix} -c_{1j} \\ \vdots \\ -c_{ij} \end{pmatrix}, \tag{3.2}$$

where $\tilde{C}_k$, for $k = 1, \ldots, i$, is the restriction of the vector C_k to its first i coordinates and c_{kj} is the jth coordinate of the vector C_k. We show that the matrix of $\tilde{C}_k$'s is nonsingular, hence $(u_{j,1}, \ldots, u_{j,i})$ is the unique solution of (3.2). The proof of the nonsingularity of this matrix mimics the argument in Theorem 3.1. Consider any linear combination of the rows adding to the zero vector:

$$\sum_{k=1}^{i} \alpha_k \tilde{C}_k = \mathbf{0}. \tag{3.3}$$

Assume that one of the α_k's is nonzero and consider the largest ℓ such that $\alpha_\ell \neq 0$. We take the inner product of both sides of (3.3) with $\tilde{N}_\ell$, where $\tilde{N}_\ell$ is the restriction of the vector N_ℓ to its first i coordinates. Note that $\tilde{N}_\ell$ has all of the nonzero entries of N_ℓ since $\ell \leq i$ and only the first ℓ entries of N_ℓ may be nonzero. So $\langle \tilde{C}_k, \tilde{N}_\ell \rangle = \langle C_k, N_\ell \rangle$ for all $k \leq \ell$. Hence the left-hand side is $\sum_{k=1}^{\ell} \alpha_k \langle C_k, N_\ell \rangle = \alpha_\ell \langle C_\ell, N_\ell \rangle$ since $\langle C_k, N_\ell \rangle = 0$ for each k with $k < \ell$. Since α_ℓ and $\langle C_\ell, N_\ell \rangle$ are nonzero while the right-hand side is zero, we get a contradiction. Thus all of the α_k's in (3.3) are zero and so the matrix with $\tilde{C}_k$'s as its rows is nonsingular and the proof is complete. $\square$

Remark. Note that the ith coordinate of N_i is nonzero. This readily implies that there is at least one cycle that has nonzero 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 ith coordinate of N_i, which is nonzero.

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 (3.1), and let 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 $i \leq j \leq d$. We first show the following claim.

LEMMA 3.3. $\langle N_{i-1}, C_{i-1} \rangle = \det(M)$.

Proof. Let X be the $(i-1) \times (i-1)$ matrix with its kth column equal to N_k truncated to its first $i-1$ coordinates. We know from Lemma 3.2 that X is an upper triangular matrix with $X[k,k] = \langle N_{k-1}, C_{k-1} \rangle$. So $\det(X) = \prod_{k=1}^{i-1} \langle N_{k-1}, C_{k-1} \rangle$. The product MX has $\langle C_j, N_k \rangle$ as its (j,k)th element, so it is a lower triangular matrix by prefix orthogonality. Hence $\det(MX)$ is the product of its diagonal values, i.e, $\det(MX) = \prod_{k=1}^{i-1} \langle N_k, C_k \rangle$. Since $\det(M) \cdot \det(X) = \det(MX)$, $\langle N_0, C_0 \rangle = 1$, and $\langle N_k, C_k \rangle \neq 0$ for all k, the lemma follows. $\square$

We know by Lemma 3.2 that $(x_{j,1}, \ldots, x_{j,i-1})$ is the unique solution to $M\mathbf{x} = b_j$. In the ith iteration of the algorithm, we could have directly computed the vector $N_i = \langle N_{i-1}, C_{i-1} \rangle (x_{i,1}, \ldots, x_{i,i-1}, 1, 0, \ldots)$, which is $\det(M)(x_{i,1}, \ldots, x_{i,i-1}, 1, 0, \ldots)$ (by Lemma 3.3), 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 ith iteration of deterministic-MCB. We will show in section 7 that a shortest cycle C_i such that $\langle C_i, N_i\rangle \neq 0$ can be computed in $O(m^2n+mn^2\log n)$ time. Let us look at bounding the time taken for the 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 ith iteration is $O((d-i)m) = O(md)$ over all j, $i+1 \leq j \leq d$. We next estimate the cost of the 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 below 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 ith 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 $\pm1,0$ matrix of size $(i-1)\times(i-1)$ and b_j is a $\pm1,0$ vector, all determinants used in Cramer's rule in solving $M\mathbf{x} = \det(M)b_j$ are bounded by $i^{i/2}$ using Hadamard's inequality. Therefore, the absolute value of each entry in N_j in the ith iteration, where $j \geq i$, is bounded by $i^{i/2}$.*

4. A faster scheme.

The update and normalization steps form the bottleneck in the algorithm 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 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 ith unit vector, $1 \leq i \leq d$. It then executes three 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.

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 $N_j = \sum_{k=\lfloor d/2\rfloor+1}^{d}\alpha_k N_k$, where $\alpha_k \in \mathbb{Q}$. 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_ℓ, N_ℓ) in which case the algorithm simply retains N_ℓ as it is and computes C_ℓ using the algorithm in section 7. With regards to time complexity, the bulk update step will be shown to take $O(md^{\omega-1})$ arithmetic operations.

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 of 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:[4]

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

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, for all i, j, where $\ell \leq i \leq mid$ and $mid+1 \leq j \leq h$, we want

$$\frac{\langle N_{mid}, C_{mid}\rangle}{\langle N_{\ell-1}, C_{\ell-1}\rangle} \langle C_i, N_j\rangle + \sum_{t=\ell}^{mid} \alpha_{tj} \langle C_i, N_t\rangle = 0. \tag{4.1}$$

Rewriting the above relations in matrix form, we get

$$A \cdot \mathcal{N}_d \cdot D = -A \cdot \mathcal{N}_u \cdot X, \tag{4.2}$$

where (let $k = mid - \ell + 1$)

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

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 $A \cdot \mathcal{N}_d \cdot D$ using $O(k^\omega)$ 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 $O(mk^{\omega-1})$.

The total number of arithmetic operations required for the bulk update step is $O(mk^{\omega-1})$; however, each arithmetic operation is quite expensive since we deal with large numbers here. In this algorithm, 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

[4]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).

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.

5. A fast deterministic algorithm.

In the approach described in the previous section, we saw that entries in the matrix $(A \cdot \mathcal{N}_u)^{-1}$ (refer to (4.2)) could be as large as $d^{\Theta(d^2)}$. Thus we were not able to use the faster scheme for updating the vectors N_j, since each arithmetic operation could cost us up to $\tilde{\Theta}(d^2)$ time. But observe that the 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 $d^{d/2}$ (see Fact 1), since we know $N_j = (y_1, \ldots, y_{mid}, 0, \ldots, 0, \langle N_{mid}, C_{mid}\rangle, 0, \ldots, 0)$, 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 the 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 $\mathbb{Z}_{p_1}, \mathbb{Z}_{p_2}, \ldots, \mathbb{Z}_{p_s}$ 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 $\mathbb{Z}_p$ takes $O(1)$ time and we thus spend $O(s \cdot mk^{\omega-1})$ 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 $\ell \le i \le mid$, then we cannot invert $A \cdot \mathcal{N}_u$ in the field $\mathbb{Z}_p$. Since each number $\langle N_i, C_i\rangle$ could be as large as $d^{d/2}$, 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 with respect to 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 $\mathbb{Z}_q$ where q is a large prime. If $q > d^{d/2}$, then q can never be a divisor of any $\langle N_i, C_i\rangle$, so we can carry out all of our arithmetic in $\mathbb{Z}_q$ since the matrix $A \cdot \mathcal{N}_u$ will be nonsingular in $\mathbb{Z}_q$. Arithmetic in $\mathbb{Z}_q$ 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 of 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; recall 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 $N_j \bmod R$, because R is much larger than any coordinate of N_j.

What properties of R do we need? The integer R must be relatively prime to the numbers: $\langle N_\ell, C_\ell\rangle$, $\langle N_{\ell+1}, C_{\ell+1}\rangle, \ldots, \langle N_{mid}, C_{mid}\rangle$ so that the triangular matrix $A \cdot \mathcal{N}_u$ 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 $\langle N_{mid}, C_{mid}\rangle / \langle N_{\ell-1}, C_{\ell-1}\rangle$ 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 $\langle N_{\ell-1}, C_{\ell-1}\rangle$, $\langle N_\ell, C_\ell\rangle, \langle N_{\ell+1}, C_{\ell+1}\rangle, \ldots,$ and $\langle N_{mid}, C_{mid}\rangle$ by doing the following.

1. Right at the beginning of the algorithm, compute d^2 primes $p_1, \ldots, p_{d^2}$, 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, 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 $\mathcal{L}$. Let $R = p^d$.

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 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 use 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 $P_{s+1} = p_{sd+1} \cdots p_{sd+d}$ is found in $\tilde{O}(d^2)$ time by binary search in the set $\{P_1, \ldots, P_d\}$. Determine a p in the set $\{p_{sd+1}, \ldots, p_{sd+d}\}$ 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 greatest common divisor (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 (see (4.1)), 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 analysis presented in section 4 that the time required for all of 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 (assuming Lemma 7.4 from section 7, which shows that the base case $T(1)$ takes $O(m^2n + mn^2 \log n)$ time)

$$T(k) = \begin{cases} 2T(k/2) + \tilde{O}(m^2 k^{\omega-1}) & \text{if } k > 1, \\ m^2 n + mn^2 \log n & \text{if } k = 1. \end{cases}$$

The recurrence solves to $T(k) = O(k(m^2n + mn^2 \log n) + k^{\omega-1}m^2 \cdot \text{poly}(\log m))$, and hence $T(d) = O(m^3n + m^2n^2 \log n) + \tilde{O}(m^{\omega+1})$, which is $O(m^3n + m^2n^2 \log n)$, because $m \leq n^2$ implies $\tilde{O}(m^{\omega+1})$ is always $o(m^3n)$. Thus we have shown the following theorem.

THEOREM 5.1. *A minimum cycle basis in a weighted directed graph with m edges and n vertices and nonnegative edge weights can be computed in $O(m^3n + m^2n^2 \log n)$ time.*

6. Faster arithmetic via randomization.

Suppose we could perform each arithmetic operation in $O(1)$ time; then our recurrence relation for $T(k)$, $k > 1$ will be given by:

$$T(k) = 2T(k/2) + O(mk^{\omega-1}).$$

We would also like to show that $T(1)$ is $O(mn + n^2 \log n)$. Then this yields a running time of $O(n^2 d \log n + nmd + m^\omega)$ or $O(m^2n + mn^2 \log n)$, since m^ω is $o(m^2n)$. Such a running time matches the complexity of minimum cycle basis computation in undirected graphs.

Now we will show that we *can* perform each arithmetic operation in $O(1)$ time. This will require working modulo a randomly chosen prime p. We will perform all arithmetic over the finite field $\mathbb{Z}_p$ and not over the rationals. The danger now is that the results of working in this field could be different from those obtained by working over rationals. Fortunately, the following theorem claims that the results remain the same, provided the prime p chosen satisfies certain properties. In the description below, let C_i, N_i denote the results obtained by the fast deterministic algorithm working over the rationals, and let C_i', N_i' be the counterparts obtained by the fast deterministic algorithm working over $\mathbb{Z}_p$.

THEOREM 6.1. *If $\langle C_i, N_i \rangle \neq 0 \pmod{p}$ for all i, $1 \leq i \leq d$, then $C_i' = C_i$ and $N_i' = N_i \pmod{p}$.*

Proof. Recall that the fast deterministic algorithm has a recursive structure. We use an inductive argument that mimics this recursion. At the very beginning, $N_i' = N_i \pmod{p}$ for all i, $1 \leq i \leq d$, as the N_i's are $\{0, 1\}$ vectors. Each recursive subproblem (when working with rationals) then takes a contiguous subset $N_\ell, N_{\ell+1}, \ldots, N_h$ and computes $C_\ell, C_{\ell+1}, \ldots, C_h$ using only $N_\ell, N_{\ell+1}, \ldots, N_h$, modifying the latter in the process. We claim that if this recursive subproblem began with $N_\ell', N_{\ell+1}', \ldots, N_h'$ instead of $N_\ell, N_{\ell+1}, \ldots, N_h$, where

$$N_\ell' = N_\ell \pmod{p},\ N_{\ell+1}' = N_{\ell+1} \pmod{p}, \ldots, N_h' = N_h \pmod{p},$$

and subsequently worked modulo p, then it would still produce the same cycles $C_\ell, \ldots, C_h$ and in addition, after modification, the relations $N_\ell' = N_\ell \pmod{p}$, $N_{\ell+1}' = N_{\ell+1} \pmod{p}, \ldots, N_h' = N_h \pmod{p}$ will continue to hold. We will use induction on $\ell - h + 1$ to prove this claim. We will explicitly show the base case when $\ell - h + 1 = 1$. We will then assume that the claim is true for all recursive subproblems with $1 \leq \ell - h + 1 < t$ and then prove it for $\ell - h + 1 = t$.

Consider a subproblem of size 1 ($\ell - h + 1 = 1$, i.e., $\ell = h$). When working over rationals, N_ℓ does not change and C_ℓ is defined as the shortest cycle such that $\langle C_\ell, N_\ell \rangle \neq 0$. When working (mod p), N'_ℓ does not change and C'_ℓ is defined as the shortest cycle such that $\langle C'_\ell, N'_\ell \rangle \neq 0 \pmod p$. Assuming $N'_\ell = N_\ell \pmod p$ at the beginning of this subproblem, and given that p satisfies $\langle C_\ell, N_\ell \rangle \neq 0 \pmod p$, it follows that $C'_\ell = C_\ell$. This shows the base case.

Next, consider the three-step process used in the fast deterministic algorithm. First, consider the first recursive step which computes $C_\ell, \dots, C_{mid}$. By the induction hypothesis, at the end of this step, we have $C'_\ell = C_\ell, \dots, C'_{mid} = C_{mid}$ and $N'_\ell = N_\ell \pmod p), \dots, N'_{mid} = N_{mid} \pmod p$. The vectors $N'_{mid+1}, \dots, N'_h$ are untouched by this step and by virtue of the initial assignment, stay identical to $N_{mid+1}, \dots, N_h$. Second, consider the bulk update step. This step modifies $N'_{mid+1} \dots N'_h$ as a function of $N'_\ell, \dots, N'_{mid}$ and $C'_\ell, \dots, C'_{mid}$. Since $N'_\ell = N_\ell \pmod p), \dots, N'_{mid} = N_{mid} \pmod p$ and $C'_1 = C_1, \dots, C'_{mid} = C_{mid}$, it follows that $N'_{mid+1} = N_{mid+1} \pmod p), \dots, N'_h = N_h \pmod p$ after the bulk update step. This sets up the necessary initial condition for the second recursive step which computes $C'_{mid+1}, \dots, C'_h$ from $N'_{mid+1}, \dots, N'_h$ alone. Applying the induction hypothesis again proves the theorem. $\square$

Selecting the number p. It remains to show how p is chosen. Consider a pool of d^2 primes, each of which is bigger than d^2, and suppose we choose a prime at random from this pool. Lemma 6.3 shows that it satisfies the conditions of Theorem 6.1 with probability of at least 3/4. Let us first make the following definition.

DEFINITION 6.2. *Call a prime p* good *if* $\langle C_i, N_i \rangle \neq 0 \pmod p$ *for each* $i \in \{1, \dots, d\}$. *Call a prime p* bad *if it is not good.*

LEMMA 6.3. *Let P be a set of d^2 primes, each of which is at least d^2. Then at least $3/4$th of the set P is good.*

Proof. We will use Lemma 3.3 here. Lemma 3.3 shows that for all $1 \leq j \leq d$, $\langle C_j, N_j \rangle = \det(M)$, where M is the $j \times j$ matrix of $\tilde{C}_k$'s on the right-hand side of (3.1). Hadamard's inequality tells us that the absolute value of $\det(M)$ is at most $d^{d/2}$, since the entries in M are $0, \pm 1$. Hence for all $1 \leq i \leq d$, we have $0 \neq |\langle C_i, N_i \rangle| \leq d^{d/2}$. Since each prime in P is at least d^2, at most $d/4$ elements in P can be divisors of $\langle C_i, N_i \rangle$. So the number of primes in P that can divide at least one of $\langle C_1, N_1 \rangle, \langle C_2, N_2 \rangle, \dots, \langle C_d, N_d \rangle$ is at most $d^2/4$. Hence the fraction of bad primes in P is at most $d^2/4d^2 \leq 1/4$. $\square$

We now need to show how to compute the pool P of d^2 primes, each bigger than d^2. As mentioned earlier, the value of $\pi(r)$, the number of primes less than r, is given by $r/6 \log r \leq \pi(r) \leq 8r/\log r$. So the elements in P can be bounded by $100 d^2 \log d$. Using sieving, we can compute the set of primes in the first $100 d^2 \log d$ numbers in $O(d^2 \log^2 d)$ time. So the set P can be determined in $O(d^2 \log^2 d) = O(m^2 \log^2 n)$ time. Note that this term does not appear in the final complexity of $O(m^2 n + mn^2 \log n)$ because it is completely dominated by the $m^2 n$ term.

Arithmetic modulo p. Under the assumption that arithmetic on $O(\log m)$ bits takes unit time, it follows that addition, subtraction, and multiplication in $\mathbb{Z}_p$ can be implemented in unit time since p is $O(d^2 \log d) = O(m^2 \log m)$. However, we also need to implement division efficiently. Once p is chosen, we will compute the multiplicative inverses of all elements in $\mathbb{Z}_p^*$ by the extended Euclid's gcd algorithm by solving $ax = 1(\text{mod} p)$ for each $a \in \mathbb{Z}_p^*$. This takes time $O(\log p)$ for each element and hence $O(p \log p) = O(d^2 \log^2 d)$ for all of the elements. Thereafter, division in $\mathbb{Z}_p$ gets implemented as multiplication with the inverse of the divisor. The $O(d^2 \log^2 d)$

term does not appear in the final complexity of $O(m^2n + mn^2 \log n)$ because it is completely dominated by the m^2n term.

Lemma 7.3, proved in section 7, shows that $T(1)$, the time taken to compute the shortest C_i such that $\langle C_i, N_i\rangle \neq 0(\bmod p)$, is $O(mn + mn^2 \log n)$. Hence Theorem 6.4 follows from the recurrence relation for $T(\cdot)$ (refer to the beginning of this section).

THEOREM 6.4. *A minimum cycle basis in a directed graph with* n *vertices and* m *edges, with nonnegative weights on its edges, can be computed with a probability of at least* $3/4$ *in* $O(m^2n + mn^2 \log n)$ *time.*

7. Computing nonorthogonal shortest cycles.

Now we come to the second key routine required by our algorithm—given a directed graph G with nonnegative edge weights, compute a shortest cycle in G whose dot product with a given vector $N_i \in \mathbb{Z}^m$ is nonzero. We will first consider the problem of computing a shortest cycle C_p such that $\langle C_p, N_i\rangle \neq 0 \pmod p$ for a number $p = O(d^2 \log d)$. 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_i. 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_i 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 nonorthogonal cycle—simply take each edge uv and combine it with s_{vu} to get a cycle. The shortest of all these cycles having a nonzero residue class is our required cycle.

LEMMA 7.1. *Let* $C = u_0u_1 \ldots u_ku_0$ *be a shortest cycle whose residue class is nonzero modulo* p *and whose shortest edge is* u_0u_1*. Then the path* $u_1u_2 \ldots u_ku_0$ *has a residue class different from the residue class of the edge* u_1u_0*, 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* $f_{u_1u_0}$*.*

Proof. First, we show that the path $u_1u_2 \ldots u_ku_0$ and the edge u_1u_0 have different residue classes. Let x denote the residue class of the path, and let y denote the residue class of the edge u_0u_1. Since C is in a nonzero 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 $u_1u_2 \ldots u_ku_0$ 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 $u_1u_2 \ldots u_ku_0$ has a different residue class from the edge u_1u_0, the length of $u_1u_2 \ldots u_ku_0$ cannot be smaller than $s_{u_1u_0}$, by the very definition of $s_{u_1u_0}$. Suppose, for a contradiction, that the length $u_1 u_2 \dots u_k u_0$ is strictly larger than $s_{u_1 u_0}$. Then combining the path between u_1 and u_0 which realizes the length $s_{u_1 u_0}$ along with the edge $u_1 u_0$ yields a cycle which is shorter than C and which has a nonzero 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 (among 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 a 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 7.2. *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 that realizes 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}, then 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, then we set it to ∞.

The update step. Suppose we have just removed w from the priority queue. We consider the u-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 7.2. 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 the s_{uv}'s for all $u, v \in V$. We have thus shown the following lemma.

LEMMA 7.3. *A shortest cycle C_p in G, whose dot product with N_i is nonzero modulo p, can be computed in $O(mn + n^2 \log n)$ time.*

We now consider the complexity of computing a shortest cycle C_i whose dot product with N_i is nonzero, instead of the condition that the dot product is nonzero modulo p. But any cycle C_i which satisfies $\langle C_i, N_i \rangle \neq 0$ satisfies $\langle C_i, N_i \rangle \neq 0 \pmod{p}$ for some $p \in \{p_1, \dots, p_{d/2}\}$, where $p_1, \dots, p_{d/2}$ are distinct primes, each of which is at least d. This follows from the isomorphism of the ring $\mathbb{Z}_{\prod p_i}$ to the ring $\mathbb{Z}_{p_1} \times$

$\mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_{d/2}}$. So any nonzero element whose absolute value is less than $\prod_{i=1}^{d/2} p_i$ is mapped to a tuple of its residues that is not the zero vector.

We have $|\langle C_i, N_i \rangle| \leq d^{d/2}$ (from Lemma 3.3 and Hadamard's inequality). Thus $|\langle C_i, N_i \rangle| < \prod_{i=1}^{d/2} p_i$. So if $\langle C_i, N_i \rangle$ is nonzero, then it is a nonzero element in $\mathbb{Z}_{\prod p_i}$, and so it satisfies $\langle C_i, N_i \rangle \neq 0 \pmod{p}$ for some p in $\{p_1, \ldots, p_{d/2}\}$. Thus a shortest cycle C_i such that $\langle C_i, N_i \rangle \neq 0$ is the shortest among all the cycles C_p, $p \in \{p_1, \ldots, p_{d/2}\}$, where C_p is a shortest cycle such that $\langle C_p, N_i \rangle \neq 0 \pmod{p}$. Hence, by Lemma 7.3, the time taken to compute C is $O(d \cdot (mn + n^2 \log n))$ or $O(m^2n + mn^2 \log n)$. Thus we have shown Lemma 7.4.

LEMMA 7.4. *A shortest cycle C_i in G, whose dot product with N_i is nonzero, can be computed in $O(m^2n + mn^2 \log n)$ time.*

8. Conclusions.

We considered the minimum cycle basis problem in directed graphs with nonnegative edge weights. We presented an $O(m^3n + m^2n^2 \log n)$ deterministic algorithm and an $O(m^2n + mn^2 \log n)$ randomized algorithm for this problem, where m is the number of edges and n is the number of vertices. These algorithms use the framework of computing cycles $C_1, \ldots, C_d$ and their supporting vectors $N_1, \ldots, N_d$ using fast matrix multiplication. However, this approach leads to large intermediate numbers and the cost of arithmetic becomes high. We overcome this problem in the randomized algorithm by working over a finite field $\mathbb{Z}_p$ for a small random prime p and by working over suitable rings $\mathbb{Z}_R$ in the deterministic algorithm. We also presented an efficient algorithm, based on Dijkstra's algorithm, to compute a shortest cycle whose dot product with a function on its edge set is nonzero.

REFERENCES

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

[2] C. BERGE, *Graphs*, North–Holland, Amsterdam, The Netherlands, 1985.

[3] F. BERGER, P. GRITZMANN, AND S. DE VRIES, *Minimum cycle bases for network graphs*, Algorithmica, 40 (2004), pp. 51–62.

[4] B. BOLLOBÁS, *Modern Graph Theory*, in Graduate Texts in Mathematics 184, Springer-Verlag, New York, 1998.

[5] 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 (1976), pp. 61–70.

[6] T. F. COLEMAN AND A. POTHEN, *The null space problem* I. *Complexity*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 527–537.

[7] T. F. COLEMAN AND A. POTHEN, *The null space problem* II. *Algorithms*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 544–563.

[8] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplications via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[9] J. C. DE PINA, *Applications of Shortest Path Methods*, Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands, 1995.

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

[11] P. M. GLEISS, *Short Cycles: Minimum Cycle Bases of Graphs from Chemistry and Biochemistry*, Ph.D. thesis, Universität Wien, Wein, Germany, 2001.

[12] P. M. GLEISS, J. LEYDOLD, AND P. F. STADLER, *Circuit bases of strongly connected digraphs*, Discuss. Math. Graph Theory, 23 (2003), pp. 241–260.

[13] A. GOLYNSKI AND J. D. HORTON, *A polynomial time algorithm to find the minimum cycle basis of a regular matroid*, in Proceedings of SWAT, Lecture Notes in Comput. Sci. 2368, Springer, Berlin, 2002, pp. 200–209.

[14] R. HARIHARAN, T. KAVITHA, AND K. MEHLHORN, *A faster deterministic algorithm for minimum cycle basis in directed graphs*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 4051, Springer, Berlin, 2006, pp. 250–261.

[15] J. D. Horton, *A polynomial-time algorithm to find a shortest cycle basis of a graph*, SIAM J. Comput., 16 (1987), pp. 358–366.

[16] T. Kavitha, *An* $\tilde{O}(m^2n)$ *randomized algorithm to compute a minimum cycle basis of a directed graph*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 3580, Springer, Berlin, 2005, pp. 273–284.

[17] T. Kavitha and K. Mehlhorn, *Algorithms to compute minimum cycle bases in directed graphs*, Theory of Comput. Syst., 40 (2007), pp. 485–505.

[18] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch, *A faster algorithm for minimum cycle bases of graphs*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 846–857.

[19] J. Leydold and P. F. Stadler, *Minimal cycle bases of outerplanar graphs*, Electron. J. Combin., 5 (1998), pp. 1–14.

[20] C. Liebchen, *Finding short integral cycle bases for cyclic timetabling*, in Proceedings of ESA, Lecture Notes in Comput. Sci. 2832, 2003, pp. 715–726.

[21] C. Liebchen and L. Peeters, *On Cyclic Timetabling and Cycles in Graphs*, Technical report 761/2002, TU Berlin, Berlin, Germany.

[22] C. Liebchen and R. Rizzi, *A greedy approach to compute a minimum cycle basis of a directed graph*, Inform. Process. Lett., 94 (2005), pp. 107–112.

[23] G. Tewari, C. Gotsman, and S. J. Gortler, *Meshing genus-*1 *point clouds using discrete one-forms*, Computers and Graphics, 30 (2006), pp. 917–926.