Strongly Polyomial-Time and NC Algorithms for Detecting Cycles in Dynamic Graphs

Edith Cohen* and Nimrod Megiddo†

(Preliminary Version)

Abstract. This paper is concerned with the problem of recognizing, in a graph with rational vector-weights associated with the edges, the existence of a cycle whose total weight is the zero vector. This problem is known to be equivalent to the problem of recognizing the existence of cycles in dynamic graphs and to the validity of some systems of recursive formulas. It was previously conjectured that combinatorial algorithms exist for the cases of two- and three-dimensional vector-weights. The present paper gives strongly polynomial algorithms for any fixed dimension. Moreover, these algorithms also establish membership in the class NC. On the other hand, it is shown that when the dimension of the weights is not fixed, the problem is equivalent to the general linear programming problem under strongly polynomial and logspace reductions.

1. Introduction

Kosaraju and Sullivan [8] presented a polynomial-time solution to the following problem which was introduced[1] by Iwano and Steiglitz [5]:

Problem 1.1. Given is a digraph $G = (V, E, \boldsymbol{f})$ where $\boldsymbol{f} : E \rightarrow R^k$ associates with each edge of G a k-vector of rational numbers. Determine whether G contains a zero-cycle, i.e., a cycle whose edge vectors sum to the zero vector.

*Department of Computer Science, Stanford University, Stanford, CA 94305, and IBM Almaden Research Center. Research partially supported by NSF PYI Grant CCR-8858097, and ONR Contract AFOSR-86-0078.

†IBM Research, Almaden Research Center, San Jose, CA 95120-6099, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

[1] A one-dimensional version was earlier studied by Orlin [13].

Problem 1.1 is important since it has been shown in [5] that a "dynamic graph" has a cycle if and only if the "static graph" which generates it has a zero-cycle. The dynamic graph is well-defined when the weights are integral. Given the "static graph" $G = (V, E)$ with integral vector-weights $\boldsymbol{c}_{ij} \in R^k$, the corresponding dynamic graph is generated as follows. Place a copy of the vertex set of G at each point of the integral lattice in R^k. For every lattice point $\boldsymbol{z}$ and for every edge $(i, j) \in E$, connect the copy of vertex i that is located at $\boldsymbol{z}$ with the copy of vertex j that is located at $\boldsymbol{z} + \boldsymbol{c}_{ij}$. The problem on this infinite dynamic graph is to identify a cycle or conclude that none exists.

A much earlier reference on essentially the same problem is the paper by Karp, Miller and Winograd [7], where the problem is raised in the context of recursive definitions. Suppose we define n functions $F_1, \cdots, F_n$ on the k-dimensional integeral lattice by

$$F_i(\boldsymbol{z}) = \psi_i(F_1(\boldsymbol{z} - \boldsymbol{c}_{i1}), \cdots, F_n(\boldsymbol{z} - \boldsymbol{c}_{in})) \ ,$$

(the $\boldsymbol{c}_{ij}$'s are integral vectors) where the ψ_i's are specified, assuming for simplicity, e.g., $F(\boldsymbol{z}) = 0$ for $\boldsymbol{z} \notin R^n_+$. In order for the functions to be well defined it is necessary and sufficient that there is no cycle whose total vector weight is nonnegative. The problem of detecting a nonnegative cycle can be reduced to the problem of detecting a zero cycle by adding at each vertex i, k loops with weights $(-1, 0, \cdots, 0)$, $(0, -1, \cdots, 0)$, $\cdots$, $(0, \cdots, 0, -1)$.

The time complexity of the algorithm of [8] is $O(Zn \log n)$ where Z is the complexity of a certain linear programming problem with $2m$ variables and $m + n + k$ constraints, $n = |V|$, $m = |E|$. Also, for the cases $k = 2, 3$, $O(Z)$ algorithms are given in [8].

It was conjectured in [8] that combinatorial algorithms exist for the cases $k = 2, 3$. Although the notion of a combinatorial algorithm is not well-defined, we confirm the conjecture of [8] and in fact prove much more than what was expected. We show that not only for $k = 2, 3$, but also for any fixed k, the problem can be solved in strongly polynomial time, and indeed by "combinatorial" algorithms. Furthermore, our algorithms can be implemented on a parallel machine so that membership in the class NC

is established for any fixed k. To complement these results we also show that the general problem (where k is considered as part of the input) is as hard as the general linear programming problem in the following sense. We show that any linear programming problem can be reduced in strongly polynomial time and logspace to our problem with general k. In Section 2 we give some necessary definitions. In Section 3 we give an overview of the basic ideas underlying our algorithms. In Section 4 we introduce a problem called the parametric minimum cycle problem. The latter, which is an interesting problem in its own right, turns out to play a key role in the solution of the zero-cycle problem. In Section 5 we describe the algorithm for detecting zero-cycles. Concluding remarks are given in Section 6. Section 7 is an appendix containing some necessary geometric lemmas.

2. Preliminaries

Definition 2.1. Given a graph $G = (V, E)$, a *circulation* $\boldsymbol{x} = (x_{ij})$ $((i,j) \in E)$ is a solution of the system:

$$\sum_j (x_{ij} - x_{ji}) = 0 \qquad (i = 1, \cdots, n)$$

$$\boldsymbol{x} \geq \mathbf{0} \ .$$

Let $E(\boldsymbol{x})$ denote the set of *active* edges, i.e., edges (i,j) with $x_{ij} > 0$. Vertices incident on active edges are said to be *active in* $\boldsymbol{x}$. If the active edges form a connected subgraph of G, then we say that the circulation $\boldsymbol{x}$ is *connected*. If these edges form a simple cycle, then we say that $\boldsymbol{x}$ is a *simple cycle*, and with no ambiguity we continue to talk about the set of active vertices as a *simple cycle*.

Remark 2.2. Every circulation $\boldsymbol{x}$ is a sum of connected circulations, corresponding to the decomposition of $E(\boldsymbol{x})$ into strongly connected components. Moreover, it is also well known (and easy to see) that every circulation can be represented as a sum of simple cycles. If a connected circulation $\boldsymbol{x} = (x_{ij})$ consists of rational numbers, then it is proportional to an integral circulation. A connected integral circulation can be represented by a cycle $(u_0, u_1, \cdots, u_q = u_0)$ $((u_{i-1}, u_i) \in E)$, not necessarily simple, where x_{ij} is interpreted as the number of times the edge (i,j) is traversed throughout the cycle. It is easy to construct irrational circulations that cannot be interpreted this way.

Definition 2.3. Given vector weights

$$\boldsymbol{c}_{ij} = (c_{ij}^1, \cdots, c_{ij}^k)^T \quad ((i,j) \in E) \ ,$$

(i.e., using the notation of Problem 1.1, $\boldsymbol{c}_{ij} = \boldsymbol{f}(e)$ where $e = (i,j)$), a circulation $\boldsymbol{x} = (x_{ij})$ is called a *zero-circulation* if it satisfies the vector equation

$$\sum_{i,j} \boldsymbol{c}_{ij} x_{ij} = \mathbf{0} \ .$$

A rational and connected zero-circulation is called a *zero-cycle*.

3. An overview

We first present an informal overview of the basic ideas involved in the algorithms we describe later in this paper.

Suppose $G = (V, E, \boldsymbol{f})$ contains a vector zero-cycle C, i.e., the sum of the vector weights $\boldsymbol{c}_{ij}$ around C is equal to the zero vector. Obviously, for any $\boldsymbol{\lambda} \in R^k$, the sum of the scalar weights $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$ around C is zero. Since C can be decomposed into simple cycles, it follows that for every $\boldsymbol{\lambda} \in R^k$, there exists a simple cycle whose weight relative to the scalars $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$ is nonpositive. In other words, if there exists a $\boldsymbol{\lambda} \in R^k$ such that all the cycles are positive relative to $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$, then this $\boldsymbol{\lambda}$ certifies that there are no zero-cycles. On the other hand, it can be shown that if for every $\boldsymbol{\lambda} \neq \mathbf{0}$ there exists a negative cycle, then there exists a vector zero-cycle.

The observation of the preceding paragraph suggests that one might first attempt to find a $\boldsymbol{\lambda}$ for which all the cycles are positive relative to the weights $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$. This task can be viewed as an extension of the well-known problem of detecting the existence of a negative cycle in a graph with scalar weights. The latter problem can be viewed as asking for an evaluation of a function at a given $\boldsymbol{\lambda}$, whereas our problem amounts to a search over the space of $\boldsymbol{\lambda}$'s. The evaluation problem can be solved by running an all pairs shortest paths algorithm. The search for $\boldsymbol{\lambda}$ as above can be formulated as an optimization problem over the $\boldsymbol{\lambda}$-space, where one seeks to maximize the minimum weight of any cycle relative to the $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$'s. However, there is a certain difficulty with this approach since the minimum is not well-defined when there are negative cycles. Note that we do not require the cycle to be simple, since the problem of finding a minimum simple cycle is NP-hard. However, we can instead consider one of the following quantities: (i) the minimum cycle-mean, i.e., the minimum of the average weight per edge of the cycle, or (ii) the minimum of the total weight of cycles (not necessarily simple) consisting of at most n edges. It is easy to see that the sign of the minimum cycle-mean (which is the same as the sign of quantity defined in (ii)) distinguishes the following three cases: 1. there exists a negative cycle, 2. there exists a zero cycle but no negative cycles, 3. all the cycles are positive.

If an algorithm for either (i) or (ii) of the preceding paragraph is given, which uses only additions and comparisons and multiplications by constants, then such an algorithm can be "lifted" to solve the optimization problem. Very roughly, the basic idea (which is explained in [9, 10, 11]) is to run the given algorithm simultaneously on a continuum of values of λ, while repeatedly restricting the set of these values until the optimum is found. Another interpretation of the lifted algorithm is that it operates on linear forms rather than constants. When it needs to compare two linear forms it first computes a hyperplane which cuts the space into two halfspaces, such that the outcome of the comparison is uniform throughout each of them. The algorithm then consults an "oracle" (whose details are given later) for selecting the correct halfspace, and moves on. The lifted algorithm maintains a polyhedron P which is the intersection of the correct halfspaces.

As noted above, if a λ is found such that all the cycles are positive then we are done. Otherwise, the lifted algorithm concludes that $\lambda = 0$ is an optimal solution, i.e., for every λ there exists a nonpositive cycle, so the choice of $\lambda = 0$ maximizes the weight of a minimum cycle. However, the zero vector itself does not convey enough information. Nonetheless, the algorithm actually computes a $\bar{\lambda} \neq 0$ (called a separating vector) in the relative interior of the set of optima[2], along with a "certificate" of optimality. The certificate consists of vector circulation values $c_1, \cdots, c_r$. These span in nonnegative linear combinations a suitable linear space, proving that there is no direction to move so that the minimum cycle becomes positive. The scalar weights $\bar{\lambda}^T c_{ij}$ then induce a decomposition of the graph, where two vertices are in the same component if they belong to the same scalar zero-cycle. It is then shown that a vector zero-cycle exists in the given graph if and only if such a cycle exists in one of the components. Also, if there is only one component (and the graph has more than one vertex) then there exists a zero-cycle. These observations suggest an algorithm which iteratively computes a separating vector, decomposes the graph accordingly, and works on the components independently. The depth of the decomposition tree is bounded by the dimension of the weights.

The part we have so far left open is the "oracle" which recognizes the correct halfspace. It turns out that, as in [12], the oracle can be implemented by recursive calls to the same algorithm in a lower dimension. This will be explained later in the paper.

We have outlined the general framework for establishing the qualitative result of strong polynomial time and membership in NC for any fixed dimension. However, to get more efficient algorithms, we perform more efficient multidimensional searches as in [12, 1, 3] which reduce the number of oracle calls and improve both the sequential and the parallel complexities. The design can be viewed as an integration of the techniques of [10] and [12] (and the further improvements of [1, 3]).

4. The problem of parametric minimum cycle

Let $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$ be a graph with two sets of vectors associated with the edges, i.e., for every $e \in E$, $\boldsymbol{f}(e) \in R^k$ and $\boldsymbol{w}(e) \in R^\ell$. We identify $\boldsymbol{f}(e)$ with the $(k-1)$-dimensional affine form:

$$\boldsymbol{f}(e) = f_1(e)\lambda_1 + \cdots + f_{k-1}(e)\lambda_{k-1} + f_k(e) \ .$$

The *weight* of an edge e is the $(\ell+1)$-tuple $(w_1(e), \ldots, w_\ell(e), \boldsymbol{f}(e))$.

Let R_1^k denote the set of vectors $\lambda = (\lambda_1, \cdots, \lambda_k)^T \in R^k$ such that $\lambda_k = 1$. When λ is assigned with specific numerical values, we denote by $\boldsymbol{f} \cdot \lambda$ the resulting set of scalar weights.

Definition 4.1. Given $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$, where for $e \in E$, $\boldsymbol{f}(e) = \mathbf{c}_e \in R^k$, we introduce the following definitions and notations:

(i) For $E' \subset E$, denote by $\boldsymbol{f}(E')$ the $(k-1)$-dimensional affine form $\sum_{e \in E'} \boldsymbol{f}(e)$.

(ii) A cycle C (not necessarily simple) is called *$\boldsymbol{w}$-minimal* if the value $\boldsymbol{w}(C) = \sum_{e \in C} \boldsymbol{w}(e)$ (where edges are counted as many times as they occur on the cycle) is minimal relative to the lexicographic order on R^ℓ.

(iii) Let $C = C(\lambda)$ denote a $\boldsymbol{w}$-minimal cycle of at most n edges which minimizes the total scalar weight $\lambda^T \mathbf{c}_e$. Denote $g(\lambda) = \boldsymbol{f}(C) \cdot \lambda$. Note that the first l coordinates of the vector weight of a minimum cycle (i.e., the values given by $\boldsymbol{w}$) are independent of λ; $g(\lambda)$ gives only the $(\ell+1)$st coordinate.

(iv) Denote by Λ the set of vectors $\lambda \in R_1^k$ where $g(\lambda)$ is maximized (possibly $\Lambda = \emptyset$). Denote by $\mathcal{K}$ the set of $\lambda \in R^k$ such that $g(\lambda) \geq 0$, and also $\mathcal{K}_1 = \mathcal{K} \cap R_1^k$.

Remark 4.2. Given a graph with scalar weights, the minimum among cycles of length less than or equal to n can be computed in one application of an all pairs shortest paths algorithm. This takes $O(|E| \cdot |V|)$ time sequentially, or $O(\log^2 n)$ time on n^3 processors in parallel. [3]

[2] There is also the possibility that the zero vector is the only optimal solution, so there is no separating vector. However, in this case, assuming strong connectivity of the graph, it can be shown that a zero-cycle exists.

[3] If the number of edges is $m = \Omega(n^3 \log n)$, we will need $m/\log^2 n$ processors to achieve the same $O(\log^2 n)$ parallel time bound.

Definition 4.3. Given $S \subset R^\ell$, denote by aff S the affine hull of S, (i.e., the smallest flat in R^ℓ that contains the set S).

Problem 4.4. [Extended Parametric Minimum Cycle] Given $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$, if $g(\boldsymbol{\lambda}) > 0$ for some $\boldsymbol{\lambda}$, then output any such $\boldsymbol{\lambda}$; otherwise, find $\boldsymbol{\lambda} \in \text{rel int}\,\Lambda$ and a collection $\mathcal{C} = \{C_1, \cdots, C_r\}$ of $\boldsymbol{w}$-minimal cycles, each of at most n edges, such that:[4]

$$\min_{1 \le i \le r} \boldsymbol{f}(C_i) \cdot \boldsymbol{\lambda} < 0 \quad \text{for} \quad \boldsymbol{\lambda} \notin \text{aff}\,\mathcal{K}_1 \ .$$

Below we propose Algorithm 4.17 for Problem 4.4, but before discussing it we introduce two subproblems. The first is to decide on which side of a given query hyperplane (in the $\boldsymbol{\lambda}$ space) $g(\boldsymbol{\lambda})$ is either maximized or unbounded. We refer to this problem as the *oracle problem.* The second is a multidimensional search problem. A call to the oracle is a costly operation, hence in the multidimensional search techniques one seeks to minimize the number of such calls.

Hyperplane queries

Remark 4.5. The function g is a concave piecewise linear mapping from R_1^k into R. Concave functions have the property that it can be effectively decided which side of a given hyperplane H contains the maximum of the function. The decision can be made by considering a neighborhood of the maximum of the function relative to H, searching for a direction of ascent from that point. This principle is explained in detail in [12], and is developed below for the special structure of our problem.

Given a hyperplane H of R_1^k, we wish to decide on which side of H the set rel int Λ lies. If $\Lambda = \emptyset$, or if rel int Λ is either contained in H or extends into both sides of H, then our oracle will actually solve Problem 4.4. If $H \cap \mathcal{K} \neq \emptyset$, we find $\boldsymbol{\lambda} \in H$ such $g(\boldsymbol{\lambda}) > 0$. If $H \cap \text{rel int}\,\Lambda \neq \emptyset$, we find $\boldsymbol{\lambda} \in H \cap \text{rel int}\,\Lambda$ and the collection $\mathcal{C}$.

The subproblem defined on a hyperplane H is very similar to the original problem:

Problem 4.6. Given $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$ and a hyperplane H in the $\boldsymbol{\lambda}$-space,

(i) recognize whether there exists $\boldsymbol{\lambda} \in H$ such that $g(\boldsymbol{\lambda}) > 0$, and if so then output any such $\boldsymbol{\lambda}$; otherwise,

(ii) find $\boldsymbol{\lambda} \in H \cap \text{rel int}(\Lambda)$ and a collection $\mathcal{C}$ of $\boldsymbol{w}$-minimal cycles such that for every $\boldsymbol{\lambda}' \notin \text{aff}\,\mathcal{K}_1$, $\min_{C \in \mathcal{C}} (\boldsymbol{\lambda}')^T \boldsymbol{c} < 0$; if $H \cap \text{rel int}\,\Lambda = \emptyset$, then

(iii) recognize which of the two halfspaces determined by H either intersects rel int Λ, or has g unbounded on it.

A procedure for solving Problem 4.6 will be called an *oracle* and the hyperplane H will be called *the query hyperplane.*

Theorem 4.7. *Problem 4.6 can be solved by three recursive calls to the $(k-1)$-dimensional form of Problem 4.4. The complexity of the additional computation is dominated by the calls to Problem 4.4. The additional computation can be done sequentially in $C|E| + D$ time for some constants $C = O(k), D = D(k)$. With $O(m + n^3)$ processors in parallel, this can be done in constant $B = B(k)$ time.*

Proof: Consider Problem 4.4 subject to $\boldsymbol{a}^T\boldsymbol{\lambda} = \alpha$ (and $\boldsymbol{\lambda} \in R_1^k$). This is in fact a $(k-1)$-dimensional version of the original problem restricted to H. If g is unbounded on H, then this fact is detected; otherwise, suppose $\boldsymbol{\lambda}^{(0)}$ is in the relative interior of the set of maximizers of $g(\boldsymbol{\lambda})$ subject to $\boldsymbol{\lambda} \in H$, and we get the collection $\mathcal{C}^{(0)}$. Suppose $t^{(0)} = g(\boldsymbol{\lambda}^{(0)})$. We wish to recognize whether $\boldsymbol{\lambda}^{(0)}$ is also a relative interior point of the set of global maxima (i.e., relative to R_1^k). If not, then we wish to decide whether for all $\boldsymbol{\lambda}^* \in R_1^k$ such that $g(\boldsymbol{\lambda}^*) \ge g(\boldsymbol{\lambda}^{(0)})$, necessarily $\boldsymbol{a}^T\boldsymbol{\lambda}^* > \alpha$, or whether for all of them $\boldsymbol{a}^T\boldsymbol{\lambda}^* < \alpha$; these are all the possible cases. Consider $G' = (V, E, \boldsymbol{w}', \boldsymbol{f})$, where $\boldsymbol{w}' = (\boldsymbol{w}, \boldsymbol{f} \cdot \boldsymbol{\lambda}^{(0)})$. Notice that all the minimum cycles of G' correspond to minimum cycles with value $t^{(0)}$ at $\boldsymbol{\lambda}$ of G. We solve Problem 4.4 twice on G', where in one of the problems we also include the constraint $\boldsymbol{a}^T\boldsymbol{\lambda} = \alpha - 1$, and in the other we include the constraint $\boldsymbol{a}^T\boldsymbol{\lambda} = \alpha + 1$. Both problems can be solved as $(k-1)$-dimensional problems since one of the λ_i's can be eliminated. Denote the optimal values of these problems by $t^{(1)}$ and $t^{(-1)}$, and denote by $\mathcal{C}^{(1)}, \mathcal{C}^{(-1)}$ the corresponding sets of cycles. Only one of the optimal values can be greater than $t^{(0)}$. If this is the case, or if one of $t^{(1)}, t^{(-1)}$ equals $t^{(0)}$ and the other is smaller, then the side of the hyperplane that contains rel int Λ is determined. Otherwise, if both are less than or both are equal to $t^{(0)}$, then $t^{(0)}$ is the global optimal value. In the latter case we can construct $\mathcal{C} = \mathcal{C}^{(-1)} \cup \mathcal{C}^{(0)} \cup \mathcal{C}^{(1)}$. The base of the recursion is the 1-dimensional problem (with no parameters at all) where C would be any cycle with minimal value. The following proposition will conclude the proof. ∎

Proposition 4.8. *Consider the collection $\mathcal{C} = \mathcal{C}^{(-1)} \cup \mathcal{C}^{(0)} \cup \mathcal{C}^{(1)}$ of circulation values, and consider the piecewise*

[4] It can be shown that $2k$ simple cycles suffice. However, we do not need the cycles themselves for deciding the existence of a zero cycle.

linear mapping

$$f_C(\lambda) = \min_{c \in C} \lambda^T c \ .$$

(i) *For every* $\lambda \notin \mathrm{aff}(K_1)$, $f_C(\lambda) < 0$ *(as required in Problem 4.4).*

(ii) *There exists* $C' \subseteq C$ *such that* $|C'| \leq 2k$, *and* C' *satisfies* (i).

Multi-dimensional search

The following multi-dimensional search problem was defined and used in [12] for solving linear programming problems in fixed dimension:

Problem 4.9. Suppose there exists an unknown convex set $X \subseteq R^d$, and an oracle is available such that for any query hyperplane H in R^d, the oracle tells whether $X \cap H = \emptyset$; if so, then the oracle tells which of the open halfspaces determined by H contains X. Given m query hyperplanes, determine the location of X relative to each of the hyperplanes, or find any hyperplane (not necessarily one of the given ones) which intersects X.

The following theorem was proven in [12]:

Theorem 4.10. *For any fixed dimension* d, *the solution of Problem 4.9 relative to some* $m/2$ *of the given hyperplanes can be found within a constant number* $C = C(d)$ *of oracle calls. The additional computation can be done sequentially in* $O(m)$ *time, and on* m *parallel processors in* $O(\log m)$ *time.*

Corollary 4.11. *For any fixed dimension* d, *Problem 4.9 can be solved in* $O(\log m)$ *oracle calls plus* $O(m)$ *time (see [12]).*

Remark 4.12. The procedure described in [12] can be parallelized so that the additional computation (besides the $O(\log m)$ oracle calls) is done by m parallel processors in $O(\log^2 m)$ time.

Definition 4.13. We define a partial order on $R^k \setminus \{0\}$ as follows. For any pair of distinct vectors $a_1, a_2 \in R^k$, denote

$$H = H(a_1, a_2) = \{\lambda \in R_1^k : a_1^T \lambda = a_2^T \lambda\} \ .$$

If g is unbounded on $H(a_1, a_2)$ or if $H(a_1, a_2) \cap \mathrm{rel\,int}\,\Lambda \neq \emptyset$, then we write $a_1 <>_\Lambda a_2$. Otherwise, g can be unbounded on at most one of the open subspaces determined by H, and also $\mathrm{rel\,int}\,\Lambda$ can intersect at most one of these open halfspaces. We denote $a_1 <_\Lambda a_2$ (respectively, $a_1 >_\Lambda a_2$) if there exists a $\lambda \in \mathrm{rel\,int}\,\Lambda$ such that $a_1^T \lambda < a_2^T \lambda$ (respectively, $a_1^T \lambda > a_2^T \lambda$), in which case the same holds for all these λ's, or if g is unbounded on the halfspace determined by the inequality $a_1^T \lambda < a_2^T \lambda$ (respectively, $a_1^T \lambda < a_2^T \lambda$). We also use the notation $<_P$ for a similar partial order relative to any set P.

Problem 4.14. Given are finite sets $A_1, \cdots, A_r$ of nonzero vectors. We wish either to find a minimal element, with respect to the partial order $<_\Lambda$, in each of the sets A_i, or (if we encounter two incomparable elements) to reduce the problem to a lower dimension. More specifically, if $A_i = \{a_1^i, \cdots, a_{s_i}^i\}$ $(a_j^i \in R^k)$ and $s = \sum s_i$, then we need to do one of the following:

(i) report indices $m_1, \cdots, m_r$ such that for every $j \neq m_i$, $a_{m_i}^i <_\Lambda a_j^i$, and a collection of closed halfspaces[5] whose intersection P contains $\mathrm{rel\,int}\,\Lambda$, and for every $\lambda \in P$ and i, $i = 1, \cdots, r$, $(a_{m_i}^i)^T \lambda \leq (a_j^i)^T \lambda$ for all j (which we denote by $a_{m_i}^i <_P a_j^i$), or

(ii) report a hyperplane H such that either g is unbounded on H or $H \cap \mathrm{rel\,int}\,\Lambda \neq \emptyset$.

Proposition 4.15. *In any fixed dimension, Problem 4.14 can be solved within either of the following complexities:*

(i) $O(\log s)$ *oracle calls and* $O(\log^2 s)$ *parallel time on* $O(s)$ *processors.*

(ii) $O(\log s)$ *oracle calls and* $O(s)$ *sequential time.*

Proof: The underlying algorithm may be viewed as an extension of the multidimensional search procedure mentioned in Theorem 4.11. Here the set X is either $\mathrm{rel\,int}\,\Lambda$ or (if the latter is empty) a domain where the function g is unbounded. Thus, the case where X intersects the query hyperplane corresponds to the case where either g is unbounded on the query hyperplane, or the latter intersects $\mathrm{rel\,int}\,\Lambda$; the case where X is contained in one of the open halfspaces corresponds to the case where either $\mathrm{rel\,int}\,\Lambda$ is contained in the halfspace, or g is unbounded on the halfspace (but bounded on the hyperplane).

We first explain how to recognize for a given pair of distinct vectors a_i, a_j whether $a_1 <_\Lambda a_2$, $a_2 <_\Lambda a_1$ or $a_1 <>_\Lambda a_2$. Consider the hyperplane $H = H(a_i, a_j) \subset R_1^k$ (see Definition 4.13). Suppose H is the query hyperplane presented to an oracle which recognizes the location of the set $\mathrm{rel\,int}\,\Lambda$ relative to H (in the sense of Problem 4.6). In particular, if $H \cap \mathrm{rel\,int}\,\Lambda \neq \emptyset$, then the oracle discovers this fact and returns a $\lambda \in H \cap \mathrm{rel\,int}\,\Lambda$. Similarly, if g is unbounded on H, then the oracle reports this fact and provides a λ such that

[5] We discuss below bounds on the cardinality of such a collection.

$g(\boldsymbol{\lambda}) > 0$. In the remaining cases the oracle reports either $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$ or $\boldsymbol{a}_2 <_\Lambda \boldsymbol{a}_1$.

The multidimensional search algorithm computes, adaptively, $O(\log s)$ hyperplane queries. If any of these hyperplanes either intersects $\operatorname{relint}\Lambda$ or has g unbounded on it, then this fact is reported, and the present problem is considered solved. Otherwise each of the hyperplanes determines a closed halfspace such that the intersection P of all these halfspaces has the following property: either g is unbounded on the interior of P, or $\operatorname{relint}\Lambda$ is nonempty and contained in the interior of P. Moreover, r vectors $\boldsymbol{a}_{m_i}$, $i = 1, \cdots, r$, are found such that for every i, we have $\boldsymbol{a}^i_{m_i} <_P \boldsymbol{a}^i_j$, for all $1 \le j \le s_i, j \ne m_i$.

We implement the algorithm as follows. It was shown in [1] and [3] that with a constant number of oracle calls (which however, grows exponentially with the dimension) one can locate X relative to at least half of the hyperplanes. A similar scheme can be applied here. We run in $O(\log s)$ phases. First, for every i, $i = 1, \cdots, r$, we match the members of A_i into $s_i/2$ arbitrary pairs. This is done with at most $s/2$ processors. We then calculate the corresponding (at most $s/2$) hyperplanes $H(\boldsymbol{a}_i, \boldsymbol{a}_j)$ (see Definition 4.13). In a constant number of oracle calls and $O(\log s)$ time we can locate $\operatorname{relint}\Lambda$ relative to half of these $s/2$ hyperplanes; unless one of these hyperplanes turns out to be a valid output (in the sense of (ii) in Problem 4.14). We now drop one vector from every pair for which the location relative to $\operatorname{relint}\Lambda$ has been found. The same is repeated with the remaining $3s/4$ vectors, and so on. Altogether, we run in $O(\log s)$ phases, each of which takes $O(\log s)$ time on $O(s)$ processors, and $O(1)$ oracle calls. Sequentially, the total time is $O(s)$ plus $O(\log s)$ oracle calls. On $O(s)$ processors the total time is $O(\log^2 s)$ plus $O(\log s)$ oracle calls. ∎

The algorithm for extended parametric minimum cycle

The algorithm described below solves Problem 4.4. It finds a vector $\boldsymbol{\lambda} \in \operatorname{relint}\Lambda$, unless $g(\boldsymbol{\lambda}) > 0$ for some $\boldsymbol{\lambda}$, in which case the algorithm outputs such a $\boldsymbol{\lambda}$. It also returns a collection $\mathcal{C}$ of $\boldsymbol{w}$-minimal cycles such that the lower envelope of $\boldsymbol{f}(C)$ is negative outside $\operatorname{aff}\mathcal{K}_1$. The number of cycles in $\mathcal{C}$ is bounded by a function of k. In fact, a collection of only $2k$ cycles can be computed.

Definition 4.16. Given any scalar minimum cycle algorithm, where the only primitive operations are additions and comparisons, we define the corresponding *lifted algorithm* as follows. We generalize the weight of an edge e to be an $(\ell+1)$-tuple $(w_1(e), \cdots, w_\ell(e), \boldsymbol{f}(e))$ where $\boldsymbol{f}(e) \in R^k$ is viewed as a $(k-1)$-dimensional affine form. Denote $\boldsymbol{w}(e) = (w_1(e), \cdots, w_\ell(e))$. We extend the minimum cycle algorithm to such edge weights. The extension of the addition operation is straightforward, namely, given weights $(\boldsymbol{w}_1, \boldsymbol{f}_1)$, $(\boldsymbol{w}_2, \boldsymbol{f}_2)$ their sum is $(\boldsymbol{w}_1 + \boldsymbol{w}_2, \boldsymbol{f}_1 + \boldsymbol{f}_2)$. Comparisons are made with respect to a lexicographic partial order on the $(\ell+1)$-tuples. It is only a partial order since in the $(\ell+1)$st coordinate we only have the partial order $<_\Lambda$ (see Definition 4.13). To compare two $(\ell+1)$-tuples $(\boldsymbol{w}_1, \boldsymbol{f}_1)$ and $(\boldsymbol{w}_2, \boldsymbol{f}_2)$, we first compare lexicographically the first ℓ scalar coordinates. If the comparison is not resolved there, we need to compare the affine forms $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$. For this purpose the lifted algorithm computes the hyperplane $H(\boldsymbol{f}_1, \boldsymbol{f}_2)$ and presents it to an oracle which is described below. The lifted algorithm maintains a set $\mathcal{H}$ of halfspaces, where initially $\mathcal{H} = \emptyset$. The oracle decides whether the vectors are comparable. If they are, it decides whether $\boldsymbol{f}_1 <_\Lambda \boldsymbol{f}_2$. If $\boldsymbol{f}_1 <>_\Lambda \boldsymbol{f}_2$ then the lifted algorithm halts since the oracle call resulted in a solution to Problem 4.4. Otherwise, the oracle has found which of the halfspaces defined by $H(\boldsymbol{f}_1, \boldsymbol{f}_2)$ contains the set $\operatorname{relint}\Lambda$, so this hyperplane is added to $\mathcal{H}$.

Algorithm 4.17.

Step 1. Run the lifted minimum cycle algorithm, collecting into $\mathcal{H}$ all the halfspaces resulting from oracle calls where comparisons are resolved. If this algorithm halts then some comparison is not resolved but then a global solution is found, so STOP. Otherwise, denote by C_M the minimum cycle found.

Step 2. Denote by P the intersection of the halfspaces in $\mathcal{H}$ (see Proposition 4.18 for some properties of P).

(i) Compute $\boldsymbol{\lambda} \in \operatorname{relint} P$. This amounts to a linear programming problem with $k-1$ variables and $|\mathcal{H}|$ constraints, and hence it can be solved in $O(|\mathcal{H}|)$ sequential time see [12].[6]

(ii) If $\boldsymbol{f}(C_M)$ is not a constant affine form, i.e., not all the coefficients $f_1(C_M), f_2(C_M), \cdots, f_{k-1}(C_M)$ equal zero, then $\Lambda = \emptyset$ and g is unbounded (see Remark 4.19). Otherwise,

(iii) Consider $g(\boldsymbol{\lambda}) = f_k(C_M)$.

- If $g(\boldsymbol{\lambda}) > 0$, then stop.
- If $g(\boldsymbol{\lambda}) = 0$, then $\mathcal{C} = \emptyset$ (since $\operatorname{aff}\mathcal{K}_1 = R^k_1$) and $\boldsymbol{\lambda} \in \operatorname{relint}\Lambda$.
- If $g(\boldsymbol{\lambda}) < 0$, then g is negative everywhere, so $\operatorname{aff}\mathcal{K}_1 = \emptyset$ and $\mathcal{C} = \{C_M\}$ (see Remark 4.19).

Proposition 4.18. *The polyhedron P is of full dimension. Moreover, $g(\boldsymbol{\lambda})$ is linear on P and either $P = \Lambda$ or g is unbounded on P (See Proposition 4.20). If $\Lambda \ne \emptyset$ then* $\dim\Lambda = k-1$.

[6] The size of $\mathcal{H}$ is bounded by the number of oracle calls.

Remark 4.19. Suppose $\dim \Lambda = k-1$. For any w-minimal cycle C, if for some $\lambda \in \operatorname{relint} \Lambda$, $f(C)\cdot\lambda = g(\Lambda)$, then $f(C)$ is constant, i.e., the first $(k-1)$ coordinates of the vector $f(C)$ are zero. In particular, this is true for C_M. Therefore, if $g(\Lambda) < 0$, then $\operatorname{aff} \mathcal{K}_1 = \emptyset$, so we can choose $\mathcal{C} = \{C_M\}$. If $g(\Lambda) = 0$, then $\operatorname{aff} \mathcal{K}_1 = R_1^k$, so $\mathcal{C} = \emptyset$.

Correctness

If an oracle call results in a solution in Step 1 of Algorithm 4.17, then correctness follows by induction on the dimension (see also the discussion under Hyperplane Queries). We now assume that no oracle call resulted in a solution in Step 1. In this case a collection $\mathcal{H}$ of halfspaces is obtained. It is easy to verify that the conditions of the following proposition hold for P, the intersection of the halfspaces in $\mathcal{H}$. This implies Proposition 4.18.

Proposition 4.20. *Suppose $P \subset R_1^k$ is such that $P \supseteq \Lambda$, and if $\Lambda = \emptyset$ then g is unbounded on P. Suppose that for all the pairs a_1, a_2 compared by the lifted minimum cycle algorithm, either $a_1 <_\Lambda a_2$ or $a_2 <_\Lambda a_1$, and also either $a_1 <_P a_2$ or $a_2 <_P a_1$, respectively. Under these conditions,*

(i) $\dim P = k-1$, *i.e., P is full-dimensional.*
(ii) *The function g is linear on P.*
(iii) *If $\Lambda \neq \emptyset$ then $P = \Lambda$; otherwise, g is unbounded on P.*

Proof: It follows that under our assumptions the algorithm finds the unique minimal element (vector cycle value). Denote this minimal element by m. It follows that $m <_\Lambda f(C)$ for any w-minimal cycle C. The uniqueness of m implies that g is linear on P and that Λ is full dimensional. For if not, there must exist at least two incomparable w-minimal cycles. Consider the restriction of g to P. If the latter is not constant, then $\Lambda = \emptyset$ and hence g is unbounded. Otherwise (if g is constant on P), it is obvious that $P = \Lambda$. ∎

Complexity

The complexity of the algorithm is strongly related to the number of oracle calls. We would like to execute only poly-logarithmically many calls. Thus, it is advantageous to group together many comparisons that could be done "in parallel" and employ the multi-dimensional search techniques discussed in Proposition 4.15. Consider the polygon P, the intersection of the halfspaces resulted from oracle calls during multi-dimensional search. From the proof of Proposition 4.15 we can see that the conditions of Proposition 4.20 still hold for P. This implies correctness of the algorithm.

Let $m = |E|$ and $n = |V|$.

Theorem 4.21. *Algorithm 4.17 can be implemented with complexity as follows.*

(i) $O(\log^{2k} n + \log^k m)$ *parallel time on* $O(n^3 + m)$ *processors.*
(ii) $O(m(\log^{2k} n + \log^k m))$ *sequential time, when* $m = \Omega(n^3 \log n)$.
(iii) $O(\log^{2k} n(n^3 + m))$ *sequential time, when* $m = O(n^3 \log n)$ *and* $m = \Omega(n^2)$.
(iv) $O(n^3 \log^{2(k-2)} n + nm \log^{2(k-1)} n)$ *sequential time, when* $m = O(n^2)$.

Proof: It is well-known that the problem of all pairs shortest paths can be solved on n^3 processors in $O(\log^2 n)$ time. The algorithm for this problem (essentially transitive closure) runs in $O(\log n)$ phases. In the first phase the minimal among all the parallel edges is determined for each pair of vertices which are linked with at least one edge. In general, the minimal value in a set is computed for $O(n^2)$ sets, each with $O(n)$ elements. More precisely, during phase ℓ, for each ordered pair (i,j) of vertices we find d_{ij}^ℓ, the length of shortest path from i to j consisting of at most 2^ℓ edges. We use the relation $d_{ij}^{\ell+1} = \min\{d_{ij}^\ell, \min_k\{d_{ik}^\ell + d_{kj}^\ell\}\}$. To find a minimum cycle, we run one more phase where we compute a minimum of the diagonal elements in the distance matrix. The complexity of this last phase is dominated by the other phases. Each phase can be implemented in one application of Problem 4.14, with $s = m$ for the first phase, and $s = n^3$ for the remaining $O(\log n)$ phases. The complexity is analyzed in Proposition 4.15.

Denote by T_k and R_k, respectively, the sequential time complexity and the parallel time complexity with $n^3 + m$ processors, of the k-dimensional problem.

Recall from Theorem 4.7 and Remark 4.12 that the time complexity of one oracle call is $O(T_{k-1})$ on a single processor and $O(R_{k-1})$ on $O(n^3 + m)$ processors. When $k = 1$, an oracle call can be implemented simply by a scalar minimum cycle algorithm. To derive an expression for the parallel complexity R_1, note that the problem can be solved by employing n^3 processors for $O(\log^2 n)$ time plus m processors for $O(\log m)$ time. Thus, on $O(n^3 + \min\{m, m\log m/\log^2 n\})$ processors we have $R_1 = O(\log^2 n + \log m)$. Notice that all the oracle calls are assumed to executed sequentially. It follows that

$$R_k = \log^3 n + \log^2 m + (\log^2 n + \log m)R_{k-1}$$

which proves (i).

The sequential complexity for $k = 1$ is $T_1 = O(\min\{nm, m + n^3\})$, and (ii)-(iv) follow from the recursion,

$$T_k = n^3 \log n + m + (\log^2 n + \log m) \cdot T_{k-1}$$

∎

5. Detecting zero-cycles

In this section we develop the procedure which decides the existence of a nontrivial zero-cycle (and computes one if one exists), using the parametric minimum cycle algorithm of Section 4. We continue to use the notation of the preceding section, and when $\boldsymbol{w}$ is the null vector we write $(V, E, \boldsymbol{f})$ instead of $(V, E, \boldsymbol{w}, \boldsymbol{f})$.

Proposition 5.1. *A graph $G = (V, E, \boldsymbol{f})$ with vector weights (see Definition 2.1) has a zero-cycle (see Definition 2.3) if and only if it has a connected zero-circulation.*

Proof: The proof follows by observing that existence of any connected zero-circulation implies existence of a *rational* connected one. ∎

Definition 5.2. Given a vector-weighted graph $G = (V, E, \boldsymbol{f})$, we use the following definitions and notation:

(i) Let $\mathcal{K}$ denote the cone of vectors $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_k)^T$ for which the scalar-weighted graph $(V, E, \boldsymbol{f} \cdot \boldsymbol{\lambda})$ has no negative cycles.

(ii) A nonzero vector $\boldsymbol{\lambda} \in \operatorname{relint} \mathcal{K}$ (the relative interior of $\mathcal{K}$) is called *a separating vector for G.*

(iii) A separating vector $\boldsymbol{\lambda}$ for which the scalar-weighted graph $(V, E, \boldsymbol{f} \cdot \boldsymbol{\lambda})$ has only positive cycles is called a *witness for G.*[7]

Remark 5.3. The cone $\mathcal{K}$ can be described as the projection on the $\boldsymbol{\lambda}$-space (R^k) of a cone in R^{n+k} (the space of $(\pi_1, \cdots, \pi_n, \boldsymbol{\lambda})$) which is characterized by the inequalities:

$$\pi_i - \pi_j + \boldsymbol{\lambda}^T \boldsymbol{c}_{ij} \geq 0 \quad ((i,j) \in E) \ .$$

Definition 5.4.

(i) Given $G = (V, E, \boldsymbol{f})$, denote by $\mathrm{CIRC}(G)$ the set of all circulation values $\boldsymbol{c} = \sum_{i,j} \boldsymbol{c}_{ij} x_{ij}$ (where $\boldsymbol{x} = (x_{ij})$ is a circulation in G).

(ii) Given a separating vector $\boldsymbol{\lambda} \neq \boldsymbol{0}$ (i.e., $\boldsymbol{\lambda} \in \operatorname{relint} \mathcal{K}$), denote by $\mathrm{ORTH}(G, \boldsymbol{\lambda})$ the set of vectors $\boldsymbol{c} \in \mathrm{CIRC}(G)$ which are orthogonal to $\boldsymbol{\lambda}$.

Note that $\mathrm{CIRC}(G)$ is a convex polyhedral cone.

Proposition 5.5. *The set $\mathcal{K}$ is precisely the set of vectors $\boldsymbol{\lambda}$ such that $\boldsymbol{\lambda}^T \boldsymbol{c} \geq 0$ for all $\boldsymbol{c} \in \mathrm{CIRC}(G)$.*

Proof: For any circulation $\boldsymbol{x}$ and any set of scalars π_i,

$$\sum_{i,j} (\pi_i - \pi_j) x_{ij} = 0 \ .$$

If the (vector) value of $\boldsymbol{x}$ is $\boldsymbol{c}$, then

$$\boldsymbol{\lambda}^T \boldsymbol{c} = \sum_{i,j} (\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}) x_{ij} \ ,$$

so by Remark 5.3, if $\boldsymbol{\lambda} \in \mathcal{K}$ then $\boldsymbol{\lambda}^T \boldsymbol{c} \geq 0$. Conversely, if $\boldsymbol{\lambda}^T \boldsymbol{c} \geq 0$ for all $\boldsymbol{c} \in \mathrm{CIRC}(G)$, then obviously there are no negative cycles in $(V, E, \boldsymbol{f} \cdot \boldsymbol{\lambda})$, so $\boldsymbol{\lambda} \in \mathcal{K}$. ∎

Theorem 5.6.

(i) $\mathrm{ORTH}(G, \boldsymbol{\lambda})$ *is independent of $\boldsymbol{\lambda}$, and hence will be denoted by* $\mathrm{ORTH}(G)$. *(Also, if $\mathcal{K} = \{\boldsymbol{0}\}$, define $\mathrm{ORTH}(G)$ to be the entire R^k.)*

(ii) $\mathrm{ORTH}(G) = (\operatorname{lin} \mathcal{K})^{\perp}$, *i.e., the orthogonal complement of the linear subspace spanned by $\mathcal{K}$ (hence it is a linear subspace).*

Proof: The proof is based on a geometric analysis which is given in the Appendix. ∎

Proposition 5.7. *Given (vector) circulation values $\boldsymbol{c}^1, \cdots, \boldsymbol{c}^r$, the following two conditions are equivalent:*

(i) *For every $\boldsymbol{\lambda} \notin \operatorname{lin} \mathcal{K}$,*

$$\min\{\boldsymbol{\lambda}^T \boldsymbol{c}^1, \cdots, \boldsymbol{\lambda}^T \boldsymbol{c}^r\} < 0 \ .$$

(ii) $\mathrm{cone}\{\boldsymbol{c}^1, \cdots, \boldsymbol{c}^r\} = \mathrm{ORTH}(G)$.

Proof: The proof follows from Theorem 5.6 part (ii) and Farkas' Lemma. We omit the details. ∎

The following problem constitutes one step of the algorithm we describe below:

Problem 5.8. Given $G = (V, E, \boldsymbol{f})$, find a witness for G (see Definition 5.2) if one exists; otherwise, find a separating vector $\boldsymbol{\lambda}$ or conclude that no such vector exists,[8] and provide a collection C of circulations with vector-values $\boldsymbol{c}^1, \cdots, \boldsymbol{c}^r$ such that

$$\mathrm{cone}\{\boldsymbol{c}^1, \cdots, \boldsymbol{c}^r\} = \mathrm{ORTH}(G) \ .$$

[7] A witness proves the nonexistence of nontrivial zero circulations. Although for this purpose the vector does not have to be in $\operatorname{relint} \mathcal{K}$, we add this as a requirement which is helpful in the recursion.

[8] Note that $\mathcal{K} \neq \emptyset$ since $\boldsymbol{0} \in \mathcal{K}$; a separating vector exists if and only if $\mathcal{K} \neq \{\boldsymbol{0}\}$.

Proposition 5.9. *Problem 5.8 can be solved using three applications of the parametric minimum cycle algorithm on G with $(k-1)$-dimensional affine forms.*

Proof: Without loss of generality, we can restrict our attention to vectors $\lambda \in R^k$ where $\lambda_k \in \{-1, 0, 1\}$. Denote

$$\mathcal{K}^{(\delta)} = \mathcal{K} \cap \{\boldsymbol{y} : y_k = \delta\} \quad (\delta \in \{-1, 0, 1\}) .$$

We use the parametric minimum cycle algorithm[9] on G as defined in Problem 4.4 to find a witness for G if one exists. Otherwise, we find nonzero vectors $\lambda^{(\delta)} \in \operatorname{rel\,int} \mathcal{K}^{(\delta)}$, $\delta = -1, 0, 1$ (if they exist), as well as collections $\mathcal{C}^{(\delta)}$ of cycles with values such that for $\delta = -1, 0, 1$, $\min_{\boldsymbol{c} \in \mathcal{C}^{(\delta)}} \{\lambda^T \boldsymbol{c}\} < 0$ for every $\lambda \notin \operatorname{aff} \mathcal{K}^{(\delta)}$ such that $\lambda_k = \delta$. If either $\mathcal{K}^{(1)}$ or $\mathcal{K}^{(-1)}$ is nonempty, then we choose λ to be either $\lambda^{(1)}$ or $\lambda^{(-1)}$, and then (for $\delta \in \{-1, 1\}$)

$$\lambda = \lambda^{(\delta)} \in \operatorname{rel\,int} \mathcal{K}^{(\delta)} \subset \operatorname{rel\,int} \mathcal{K} .$$

If $\mathcal{K}^{(-1)} = \mathcal{K}^{(1)} = \emptyset$ and $\mathcal{K}^{(0)} \neq \{\boldsymbol{0}\}$, we choose λ to be a nonzero vector $\lambda^{(0)} \in \operatorname{rel\,int} \mathcal{K}^{(0)}$, and so

$$\lambda = \lambda^{(0)} \in \operatorname{rel\,int} \mathcal{K}^{(0)} \subseteq \operatorname{rel\,int} \mathcal{K} .$$

The remaining case is $\mathcal{K}^{(-1)} = \mathcal{K}^{(1)} = \emptyset$ and $\mathcal{K}^{(0)} = \{\boldsymbol{0}\}$. Here we conclude that $\mathcal{K} = \{\boldsymbol{0}\}$ so there is no separating vector. It is easy to verify, using Proposition 5.7, that $\mathcal{C} = \mathcal{C}^{(-1)} \cup \mathcal{C}^{(0)} \cup \mathcal{C}^{(1)}$ is such that $\operatorname{cone} \mathcal{C} = \operatorname{ORTH}(G)$. ∎

The following proposition is mentioned in [8].

Proposition 5.10. *If $G = (V, E, \boldsymbol{d})$ is a scalar-weighted graph with no negative cycles, then with one application of the all pairs shortest paths algorithm we can find the partition of V into $V_1, \cdots, V_q$, where u and v are in the same V_i if and only if there exists a (scalar) zero-cycle of G in which both u and v are active.*

Proof: We find values π_i such that for every edge (i, j),

$$\delta_{ij} \equiv \pi_i - \pi_j + d_{ij} \geq 0 .$$

Then, u and v are in the same component if and only if the distance from u to v and the distance from v to u are both zero relative to the δ_{ij}'s. ∎

Remark 5.11. Each component of the partition in Proposition 5.10 contains a zero cycle where all the vertices of the component are active. This zero cycle can be easily constructed.

[9] We use three $(k-1)$-dimensional affine forms $f \cdot \lambda_\delta$ where $\lambda_\delta = (\lambda_1, \cdots, \lambda_{k-1}, \delta)^T$ for $\delta = -1, 0, 1$.

Proposition 5.12. *If $\operatorname{CIRC}(G)$ contains a nontrivial linear subspace then a nontrivial (vector) zero-circulation exists in G.*

Proof: The proof is immediate. ∎

Proposition 5.13. *For every fixed ℓ, if $\mathcal{C} = \{\boldsymbol{c}_1, \cdots, \boldsymbol{c}_r\} \subset R^\ell$ then for any $\boldsymbol{v} \in R^\ell$, it takes $O(|\mathcal{C}|)$ time to find nonnegative rational constants $\alpha_1, \cdots, \alpha_r$ such that $\boldsymbol{v} = \sum \alpha_i \boldsymbol{c}_i$, or to recognize that no such coefficients exist.*[10]

Proof: The problem can be solved as a linear programming problem with a fixed number of variables [12]. ∎

Proposition 5.14. *For every separating vector λ of $G = (V, E, \boldsymbol{f})$, if the partition of $(V, E, \boldsymbol{f} \cdot \lambda)$ (see Proposition 5.10) is such that $q = 1$, then G has a (vector) zero-cycle if and only if $(V, E, \boldsymbol{f} \cdot \lambda)$ has a (scalar) zero-cycle.*

Proof: The 'only if' part is trivial. We now prove the 'if' part. If G has one component relative to $\boldsymbol{f} \cdot \lambda$, then there exists a scalar zero-cycle in $(V, E, \boldsymbol{f} \cdot \lambda)$ in which all the vertices are active. Thus, there exists a value $\boldsymbol{c} \in \operatorname{ORTH}(G)$ which is attained at a circulation where all the vertices are active, so this circulation is connected. By Theorem 5.6 and Proposition 5.12, there exists a circulation, not necessarily connected, whose value is $-\boldsymbol{c}$. Now, by combining the connected circulation supporting $\boldsymbol{c}$ with the one supporting $-\boldsymbol{c}$, we obtain a connected (nontrivial) zero-circulation, i.e., a zero-cycle of G. ∎

Remark 5.15. Given a set $\mathcal{C}$ of (vector) cycle values such that $\operatorname{cone} \mathcal{C} = \operatorname{ORTH}(G)$, we can construct the vector zero-cycle of Proposition 5.14, since Remark 5.11 gives us the circulation with value $\boldsymbol{c}$, and by Proposition 5.13 we can construct the circulation with value $-\boldsymbol{c}$.

Remark 5.16. If there is no separating vector for G, then $\operatorname{ORTH}(G) = R^k$. Given a set $\mathcal{C}$ as above, we can find a zero circulation as follows. Since G is strongly connected, we can find a cycle with value $\boldsymbol{c}$ in which all the vertices are active. By Proposition 5.11 we can find a circulation with value $-\boldsymbol{c}$. We then combine the two to get a (nontrivial) zero circulation.

Proposition 5.17. *Suppose λ is separating and let $V_1, \cdots, V_q$ denote the components defined in Proposition 5.10 relative to the scalar weights $\boldsymbol{d} = \boldsymbol{f} \cdot \lambda$. (By definition, there are no negative cycles.) Denote by $G_i =$*

[10] In our application $|\mathcal{C}|$ is also a constant (proportional to ℓ) so this problem is solved in constant time.

$(V_i, E_i, \boldsymbol{f}_i)$ *the vector-weighted subgraph induced on* V_i. *Under these conditions, a (vector) zero-cycle exists in* G *if and only if a (vector) zero-cycle exists in one of the* G_i*'s.*

Proof: The 'if' part is trivial. For the 'only if' part, suppose $\boldsymbol{x}$ is a (vector) zero-cycle of $G = (V, E, \boldsymbol{f})$. Then $\boldsymbol{x}$ is a scalar zero-cycle of $(V, E, \boldsymbol{f} \cdot \boldsymbol{\lambda})$. By the definition in Proposition 5.10, all the vertices active in $\boldsymbol{x}$ are in the same component G_i and hence $\boldsymbol{x}$ is a scalar zero-cycle of G_i. By Proposition 5.14, G_i has a vector zero-cycle. ∎

Proposition 5.18. *A witness for* G *exists if and only if* G *does not have a non-trivial zero circulation.*

Proof: The proof is immediate ∎

Algorithm 5.19. [Zero-Cycle Detection]

(i) Use an algorithm for Problem 5.8 on G (see Proposition 5.9). If a witness for G is found then stop. Otherwise, find a collection $\mathcal{C}$ of circulation values such that $\text{cone}\,\mathcal{C} = \text{ORTH}(G)$, and either find a separating vector $\boldsymbol{\lambda}$ or conclude that none exists. In the latter case conclude that a connected zero-circulation, and hence a zero-cycle, exist in G. (See Remark 5.16 for how to find the zero-cycle.) Otherwise,

(ii) Construct the partition of G which is defined in Propositions 5.10 and 5.17.

(iii) If the partition is trivial (i.e., $q = 1$), then apply the all pairs shortest paths algorithm to check whether the scalar-weighted graph $(V, E, \boldsymbol{f} \cdot \boldsymbol{\lambda})$ has a zero-cycle. By Proposition 5.14, $G = (V, E, \boldsymbol{f})$ has a zero-cycle if and only if $(V, E, \boldsymbol{f} \cdot \boldsymbol{\lambda})$ has one. (See Remark 5.15 for how to find the zero-cycle.)

(iv) Apply (recursively) the Zero-Cycle Detection algorithm to $G_1, \cdots, G_q$; if a graph has only one vertex then it has no zero-circulation.

In the rest of the present section we prove the correctness and analyze the complexity of Algorithm 5.19.

Proposition 5.20. *If* G *is partitioned into* $G_1, \cdots, G_q$ *(see Proposition 5.17) and for some* G_i,

$$\dim(\text{ORTH}(G_i)) = \dim(\text{ORTH}(G)) ,$$

then G_i *will not be partitioned any further by the algorithm.*

Proof: Since $\text{ORTH}(G_i) \subseteq \text{ORTH}(G)$, equality of dimension implies equality of the sets, so a separating vector for G would be a separating vector for G_i ∎

Corollary 5.21. *Algorithm 5.19 terminates after at most* $k - 1$ *phases of partitioning.*

Theorem 5.22. *The complexity of the Zero-Cycle Detection algorithm for a graph* $G = (V, E, \boldsymbol{f})$ *(where* $|V| = n$ *and* $\boldsymbol{f}$ *is* k*-dimensional) is essentially the complexity of* k *applications of the* $(k-1)$*-dimensional minimum cycle algorithm of section 4.*

Proof: Consider the recursion tree of Algorithm 5.19. By Corollary 5.21 it has k levels. At each level we have a collection of subgraphs $G_1, \cdots, G_q$ with total number of n vertices. The algorithm will solve Problem 5.8 for each G_i and then partition G_i (unless done) as in Proposition 5.17. Since the complexity of these operations is super linear, the time and processor complexities of each level equals the sum of the complexity of the partitioning operation (which by Proposition 5.10 amounts to an all pairs shortest paths computation) and the complexity of Problem 5.8. The dominant term is the complexity of finding a separating vector. This was analyzed in Proposition 5.9 to be the complexity of the $(k-1)$-dimensional minimum cycle problem. ∎

6. Concluding remarks

The obvious open question that arises in this paper is whether Problem 1.1, where the dimension k is part of the input, can be solved in strongly polynomial time, and whether it is in the class NC. It is interesting to note the following:

Proposition 6.1. *The problem of detecting a zero cycle (Problem 1.1) is P-complete, and also the general linear programming problem is reducible to it in strongly polynomial time.*

Proof: The general linear programming problem is equivalent (in strongly polynomial time and NC reducibilities) to the problem of solving the following system:

$$(S) \qquad \begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{0} \\ \boldsymbol{0} \neq \boldsymbol{x} &\geq \boldsymbol{0} , \end{aligned}$$

where $\boldsymbol{A} \in R^{m \times n}$. Consider a network consisting of m parallel[11] edges from vertex s to vertex t and one edge from t to s. Associate with the i'th edge the weight-vector given by the i'th row of $\boldsymbol{A}$, and associate with the reverse edge the zero vector of weights. The existence of a nontrivial zero circulation in this network is equivalent to the existence of a solution to the given system (S). This establishes our claim. ∎

[11] It is a trivial matter to avoid parallel edges if this is desired.

In view of Proposition 6.1 the questions stated in the beginning of this section are equivalent to two famous and difficult open questions.

Recall that in this paper cycles were not necessarily simple. Unfortunately, if simplicity of the cycle is added to the requirements, the problem becomes NP-complete. Moreover, even the problem of recognizing whether a graph with scalar weights has a simple cycle with zero total weight is NP-complete, since the knapsack problem can be reduced to the detection of a simple zero-cycle in a graph whose edges form a ring where two consecutive vertices are connected with two parallel edges.

7. Appendix

In this Appendix we give the necessary lemmas which establish the proof of Theorem 5.6. The reader is referred to [4] for background.

For any subset C of R^d, denote by $\operatorname{lin} C$ the linear subspace spanned by C. Denote

$$C^+ = \{v : (\forall u \in C)(v^T u \geq 0)\} .$$

Recall that a cone which does not contain a nontrivial linear space is said to be *pointed.*

Proposition 7.1.

(i) *Every cone C is a direct sum, $C = L \oplus C_p$, of a linear subspace L and a pointed cone C_p.*

(ii) *C_p is contained in the orthogonal complement of L in* $\operatorname{lin} C$.

(iii) $\dim(C_p) = \dim(C) - \dim(L)$.

Proof: Well known. ∎

Proposition 7.2. *If C is a pointed cone then C^+ is full-dimensional.*

Proof: The following claim is a consequence of the duality theorem: For any finite set of vectors $u^1, \cdots, u^r$, if there does not exist $\alpha = (\alpha_1, \cdots, \alpha_r)^T \geq 0$, $\alpha \neq 0$, such that $\sum \alpha_i u^i = 0$, then there exists a vector v such that $v^T u^i \geq 1$, $i = 1, \cdots, r$. From this it can be shown that if C is a pointed cone (not necessarily polyhedral), then there exists a vector v such that for every unit-vector $u \in C$, $v^T u \geq 1$. It follows that $v \in C^+$ and there exists a ball B centered at v such that for every $w \in B$ and $u \in C$ $(u \neq 0)$, $w^T u > 0$. This implies $B \subset \mathcal{K}$. ∎

Let $C = \mathrm{CIRC}(G)$ (see Definition 5.4), let L and C_p be as in Proposition 7.1, and let $L^\perp$ denote the orthogonal complement of L in $\operatorname{lin} C$. Recall from Proposition 5.5 that $\mathcal{K}$ is the set of all vectors λ such that $\lambda^T c \geq 0$ for all $c \in C$. In other words, $\mathcal{K} = C^+$.

Proposition 7.3. *The cone $\mathcal{K}$ is contained in $L^\perp$, and hence $\mathcal{K} = C_p^+ \cap L^\perp$.*

Proof: The proof follows from the fact that if D is a linear subspace and $y \in D^+$ then $y^T d = 0$ for all $d \in D$. ∎

Proposition 7.4.

$$\dim(\mathcal{K}) = \dim(L^\perp) .$$

Proof: We use arguments similar to those given in the proof of Proposition 7.2. First, since C_p is orthogonal to L and is pointed, for any two sets of vectors $\{u^1, \cdots, u^r\} \subset C_p$ and $\{w^1, \cdots, w^s\} \subset L$, there do not exist vectors $\alpha = (\alpha_1, \cdots, \alpha_r)^T \geq 0$ $(\alpha \neq 0)$ and $(\beta_1, \cdots, \beta_s)^T$ such that

$$\sum_{i=1}^{r} \alpha_i u^i + \sum_{i=1}^{s} \beta_i w^i = 0 .$$

By the duality theorem it follows that there exists a vector v such that for every unit-vector $u \in C_p$, $v^T u \geq 1$, and for every $w \in L$, $v^T w = 0$. Thus $v \in C_p^+ \cap L^\perp = \mathcal{K}$. Moreover, $\mathcal{K}$ is full-dimensional relative to $L^\perp$. ∎

Proposition 7.5. *If $\lambda \in \operatorname{rel\,int}(\mathcal{K})$ then for all $c \in C_p$ $(c \neq 0)$, $\lambda^T c > 0$.*

Proof: From the proof of Proposition 7.4 we know that there exists a vector v such that for every unit-vector $u \in C_p$, $v^T u \geq 1$, and for every $w \in L$, $v^T w = 0$, and $\mathcal{K}$ is full-dimensional relative to $L^\perp$. This implies that if $\lambda^T c = 0$ for some $c \in C_p$ $(c \neq 0)$, then $\lambda \notin \operatorname{rel\,int} \mathcal{K}$. ∎

Corollary 7.6. *For every $\lambda \in \operatorname{rel\,int}(\mathcal{K})$, the set* $\mathrm{ORTH}(G, \lambda)$ *is equal to the linear subspace L.*

Proof: By Proposition 7.5, if $\lambda \in \operatorname{rel\,int} \mathcal{K}$ and $c \in C$ are such that $\lambda^T c = 0$, then necessarily $c \in L$. On the other hand, since $\mathcal{K} \subseteq L^\perp$ (see Proposition 7.3), if $c \in L$ and $\lambda \in \mathcal{K}$ then $\lambda^T c = 0$. ∎

References

[1] K. L. Clarkson, "Linear programming in $O(n \times 3^{d^2})$ time," *Information Processing Letters* **22** (1986) 21–27.

[2] L. Danzer, B. Grünbaum and V. Klee, "Helly's theorem and its relatives," in: *Proc. Symp. Pure Math.* **7**, American Mathematical Society, 1963, pp. 101–180.

[3] M. E. Dyer, "On a multidimensional search technique and its application to the Euclidean one-center problem," *SIAM J. Comput.* **15** (1986) 725–738.

[4] B. Grünbaum, *Convex polytopes*, Interscience-Wiley, London, 1967.

[5] K. Iwano and K. Steiglitz, "Testing for cycles in infinite graphs with periodic structure," *Proceedings of the 19th ACM Annual Symposium on Theory of Computing (1987)*, pp. 46–53.

[6] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics* **23** (1978) 309–311.

[7] R. M. Karp, R. E. Miller and S. Winograd, "The organization of computations for uniform recurrence equations," *J. Assoc. Comput. Mach.* **14** (1967) 563–590.

[8] S. R. Kosaraju and G. F. Sullivan, "Detecting cycles in dynamic graphs in polynomial time," in: *Proceedings of the 20th ACM Annual Symposium on Theory of Computing (1988)*, pp. 398–406.

[9] N. Megiddo, "Combinatorial optimization with rational objective functions," *Mathematics of Operations Research* **4** (1979) 414–424.

[10] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. Assoc. Comput. Mach.* **30** (1983) 337–341.

[11] N. Megiddo, "Towards a genuinely polynomial algorithm for linear programming," *SIAM Journal on Computing* **12** (1983) 347–353.

[12] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. Assoc. Comput. Mach.* **31** (1984) 114–127.

[13] J. B. Orlin, "Some problems in dynamic/periodic graphs," in: *Progress in in Combinatorial Optimization*, W. R. Pullyblank, Ed., Academic Press, Orlando, Florida, 1984, pp. 273–293.