Strongly Polynomial-Time and NC Algorithms for Detecting Cycles in Periodic Graphs

Edith Cohen* Nimrod Megiddo†

December 17, 1991

Abstract

This paper[1] 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 periodic (dynamic) graphs and to the validity of systems of recursive formulas. It was previously conjectured that combinatorial algorithms exist for the cases of two- and three-dimensional vector-weights. We show that strongly polynomial algorithms exist for any fixed dimension d. Moreover, these algorithms also establish membership in the class $\mathcal{NC}$. 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. The algorithms presented here solve the cycle detection problem by reducing it to instances of the parametric minimum cycle problem. In the latter, we consider graphs with edge-weights which are linear functions of d parameters. The goal, roughly, is to find an assignment of the parameters such that the value of the minimum weight cycle is maximized. The technique we used in order to obtain strongly polynomial algorithms for the parametric minimum cycle problem is a general tool applicable to parametric extensions of a variety of other problems.

1. Introduction

This paper is concerned with the following problem:

---

*Currently at: AT&T Bell Laboratories, 600 Mountain Av. rm. 2D-146, Murray Hill, NJ 07974. Work on this paper was done while the first author was at Stanford University and IBM Almaden Research Center. Research partially supported by NSF PYI Grant CCR-8858097, and ONR Contract N00014-88-K-0166.

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

[1]Extended abstract appeared in STOC 89 [3]

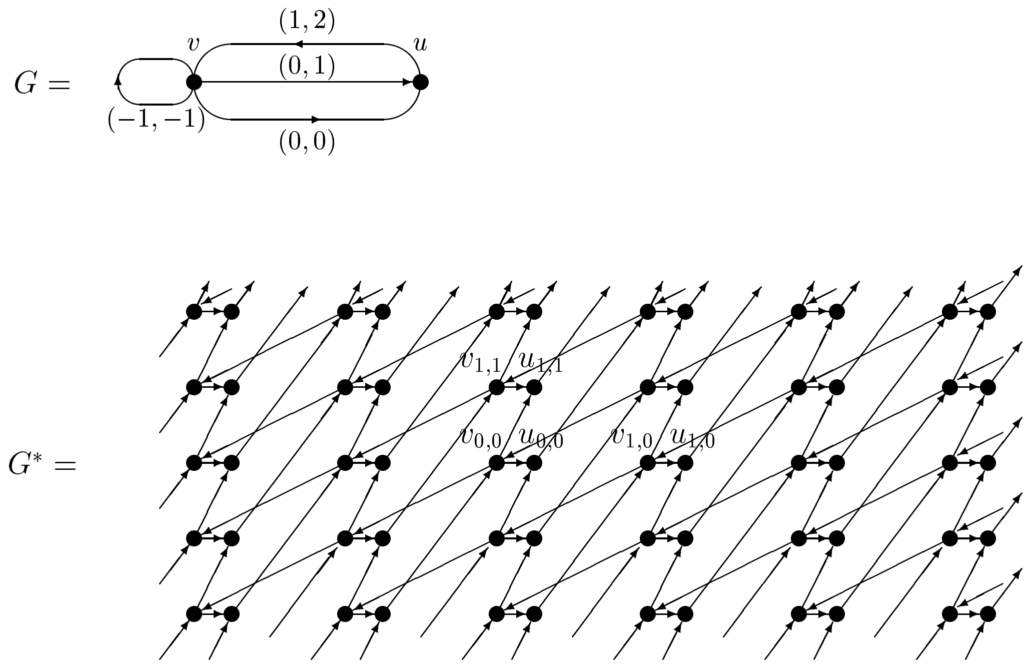


Figure 1: A two-dimensional dependence graph G and the periodic graph G^*

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

When the edge vectors are integral, G can be viewed as a representation of a *periodic graph*. It has been shown by Iwano and Steiglitz [13, 15] that the periodic graph generated by G contains a directed cycle if and only if G has a zero-cycle.

Periodic graphs are infinite digraphs with a regular structure. For a given finite digraph $G = (V, E)$ with integral vector-weights $\boldsymbol{c}_{ij} \in R^d$ (called the *dependence graph*), the corresponding periodic graph is generated as follows. Place a copy of the vertex set of G at each point of the integral lattice in R^d. 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 periodic graph is to identify a cycle or conclude that none exists. See figure 1 for an example of a dependence graph and the corresponding periodic graph.

The problem was first introduced in a paper by Karp, Miller and Winograd [17], where it was raised in the context of recursive definitions. They gave an algorithm which amounts to solving polynomially many linear programs. They stated the problem of validity of recursive definitions as detecting a cycle in a periodic graph. Consider n functions $F_1, \ldots, F_n$ on the d-dimensional integral lattice defined by

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

(the $\boldsymbol{c}_{ij}$'s are integral vectors) where the ψ_i's are specified (assume for simplicity the boundary conditions $F(\boldsymbol{z}) = 0$ for $\boldsymbol{z} \notin R^n_+$). The corresponding dependence graph has n vertices $v_1, \ldots, v_n$, and edges (v_i, v_j) of weight $\boldsymbol{c}_{ij}$ if $F_i(\boldsymbol{z})$ depends on $F_j(\boldsymbol{z} - \boldsymbol{c}_{ij})$. In order for the functions to be well defined it is necessary and sufficient that there will be 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, d loops with weights $(-1, 0, \ldots, 0)$, $(0, -1, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, -1)$. As an example, consider the recurrence relation $F(z) = F(z-1)+F(z-2)$. The corresponding dependence graph consists of a single vertex and two loops of weights -1 and -2.

The problem was re-introduced by Iwano and Steiglitz [13, 15] following Orlin [25]. Orlin studied properties of one-dimensional periodic graphs which included computing strongly connected components. Kosaraju and Sullivan [18] presented a polynomial-time algorithm. The time complexity of the latter is $O(Zn \log n)$, where Z is the complexity of a certain linear programming problem with $2m$ variables and $m + n + k$ constraints ($n = |V|$, $m = |E|$). They also presented $O(Z)$ algorithms for the cases $d = 2, 3$.

It was conjectured in [18] that combinatorial algorithms exist for the cases $d = 2, 3$. Although the notion of a combinatorial algorithm is not well-defined, we believe we have confirmed this conjecture and have in fact proven more than what was expected. We show that not only for $d = 2, 3$, but also for any fixed d, 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 $\mathcal{NC}$ is established for any fixed d. To complement these results, we also show that the general problem (where d is considered part of the input) is as hard as the general linear programming problem in a sense as follows. We show that any linear programming problem can be reduced in strongly polynomial time and logspace to our problem with general d.

Consider an instance of Problem 1.1 where $m = |E|$ is the number of edges and $n = |v|$ is the number of vertices. We show that when d is fixed the problem can be solved within the following bounds:

i. $O(\log^{2d} n + \log^d m)$ parallel time on $O(n^3 + m)$ processors.

ii. $O(m(\log^{2d} n + \log^d m))$ sequential time, when $m = \Omega(n^3 \log n)$.

iii. $O((n^3 + m) \log^{2d} n)$ sequential time, when $m = O(n^3 \log n)$ and $m = \Omega(n^2)$.

iv. $O(n^3 \log^{2(d-2)} n + nm \log^{2(d-1)} n)$ sequential time, when $m = O(n^2)$.

The constant factors hidden in the above bounds are of the order of $O(3^{d^2})$, and arise from the multi-dimensional search algorithm [1, 8, 23].

It is worth mentioning that other properties of periodic graphs were considered in the literature. The computation of strongly connected components [25] (see Section 6.2), scheduling the computation of systems of recursive definitions [17, 27] (see Section 6.1), planarity testing [12], computing connected components, recognizing bipartiteness [6, 14, 25], and computing a minimum average cost spanning tree [6, 25].

Our algorithm for Problem 1.1 is based on recursively solving instances of either one of the parametric minimum cycle or the parametric minimum cycle-mean problems. The time complexity is dominated by that of solving $O(d)$ instances of the parametric minimum cycle problem with $d-1$ parameters, m edges and n nodes. The parametric minimum cycle (resp., cycle-mean) problem is defined as follows. Consider a digraph where weights, which are linear functions of d variables ("parameters"), are associated with the edges. Each set of values for the parameters corresponds to a set of scalar weights associated with the edges of the graph. The goal, roughly, is to find the set of values for which the weight of the minimum-weight cycle (resp., minimum cycle-mean) is maximized.

The parametric extensions of the minimum cycle and the minimum cycle-mean problems, which are interesting in their own right, can be solved by a sequence of linear programming problems. We present algorithms that perform the computation in strongly polynomial time and in $\mathcal{NC}$, when the number of parameters is fixed. The method is based on an integration of techniques from [21, 23]. It is a general tool for achieving strongly polynomial bounds. The method used to maximize the function that maps sets of parameter values to the value of the minimum-weight cycle can actually be applied to maximize (resp., minimize) large family of convex (resp., concave) functions.

More specifically: suppose $\mathcal{Q} \subset R^d$ is a convex set given as an intersection of k halfspaces, and let $g : \mathcal{Q} \to R$ be a concave function that is computable by a piecewise affine algorithm (i.e., roughly, an algorithm that performs only multiplications by scalars, additions, and comparisons of intermediate values which depend on the input). Assume that such an algorithm $\mathcal{A}$ is given and the maximal number of operations required by $\mathcal{A}$ on any input (i.e., point in $\mathcal{Q}$) is T. Under these assumptions, for any fixed d, the function g can be maximized in a number of operations polynomial in k and T. (see [2, 4] for a description of the method without details specific to the parametric minimum cycle problem.)

Hence, the method can be applied to obtain strongly polynomial algorithms for parametric extensions of other problems .

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 describe a strongly polynomial algorithm for detecting zero-cycles. This algorithm is stated in terms of solving instances of a parametric version of the minimum cycle problem. Section 5 contains some necessary geometric lemmas. Section 6 discusses two other problems on periodic graphs for which the cycle detection algorithm is applicable. One problem is computing the strongly connected components, the other is scheduling the computation of systems of uniform recurrences which are modeled by periodic graphs. In Section 7 we give a strongly polynomial and logspace reduction of general linear programming problem to the cycle detection problem when we allow the dimension of the weights to be part of the input.

Sections 8–10 introduce the parametric minimum cycle and the parametric minimum cycle-mean problems and present strongly polynomial algorithms when the number of parameters is fixed. In Section 8 we give some definitions and describe the general setup. In Section 9 we present a simplified algorithm for the parametric minimum cycle-mean problem. The goal of this presentation is to give the reader a sense of how the strongly polynomial time bounds are achieved. Details which are not essential for the qualitative result of strongly polynomial time bounds are avoided. The reader may skip Section 9, since the succeeding sections are independent. In Section 10 we give an algorithm for the problem of parametric minimum cycle. This section introduces additional ideas whose purpose is to improve the sequential and parallel time bounds. In Section 11 we present a scheme for obtaining strongly polynomial algorithm for parametric extensions of other problems.

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:

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

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

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 $(u_0, u_1, \ldots, u_q = u_0)$ $((u_{i-1}, u_i) \in E)$, 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 $\boldsymbol{c}_{ij} = (c^1_{ij}, \ldots, c^d_{ij})^T$ $((i, j) \in E)$ (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 $\sum_{i,j} \boldsymbol{c}_{ij} x_{ij} = \mathbf{0}$. An integral connected nontrivial zero-circulation is called a *zero-cycle*.

3. An overview

We first present an informal overview of the basic ideas involved in the zero-cycle detection algorithm.

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 $\boldsymbol{\lambda} \in R^d$, the sum of the scalar weights $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$ around C is zero. It follows that for every $\boldsymbol{\lambda} \in R^d$, the weight of the minimum cycle relative to the scalars $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$ is nonpositive. In other words, if there exists a $\boldsymbol{\lambda} \in R^d$

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 $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$. In other words, we wish to maximize over $\boldsymbol{\lambda}$ the weight of the minimum cycle relative to the scalar weights $\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}$.

This task can be viewed as a parameterized extension of the well-known problem of detecting the existence of a negative cycle or finding a minimum weight cycle in a graph with scalar weights. The latter scalar weights problem can be viewed as asking for an evaluation of a function at a given $\boldsymbol{\lambda}$, which 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 function of 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 $\mathcal{NP}$-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 the quantity defined in (ii)) distinguishes the following three cases: I. there exists a negative cycle, II. there exists a zero cycle but no negative cycles, and III. all the cycles are positive.

If an algorithm for either (i) or (ii) of the preceding paragraph is given, which uses only additions, 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 [20, 21, 22]) is to run the given algorithm simultaneously on a continuum of values of $\boldsymbol{\lambda}$, 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 the lifted algorithm 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 vector $\boldsymbol{\lambda}$ is found such that all the cycles are positive then we are done. Otherwise, the lifted algorithm concludes that $\boldsymbol{\lambda} = \mathbf{0}$ is an optimal solution, i.e., for every $\boldsymbol{\lambda}$ there exists a nonpositive cycle, so the choice of $\boldsymbol{\lambda} = \mathbf{0}$ maximizes the weight of a minimum cycle. However, the zero vector itself does not convey enough information. Nonetheless, the algorithm actually computes a vector $\bar{\boldsymbol{\lambda}} \neq \mathbf{0}$ (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 $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r$. These values span in nonnegative linear combinations a suitable linear space, proving that there is no direction to move so

[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.

that the minimum cycle becomes positive. This "certificate" is used to actually find a zero-cycle when the algorithm decides that one exists. The scalar weights $\bar{\boldsymbol{\lambda}}^T \boldsymbol{c}_{ij}$ 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 [23], the oracle can be implemented by recursive calls to the same algorithm in a lower dimension. This will be explained in detail later.

We have outlined the general framework for establishing the qualitative result of strongly polynomial time bounds for any fixed dimension. However, to get more efficient algorithms and to establish membership in $\mathcal{NC}$, we perform multi-dimensional searches as in [1, 8, 23]. By doing so we reduce the number of calls to an "oracle" algorithm which actually need to be performed, to a polylog in the number of decisions. The design can be viewed as an integration of the techniques of [21] and [23] (and the further improvements of [1, 8]).

4. Detecting zero-cycles

In this section we develop an algorithm which decides the existence of a zero-cycle in the vector-weighted graph $G = (V, E, \boldsymbol{f})$, $\boldsymbol{f} : E \to Z^d$. If a zero-cycle exists in G, we find an explicit one. The algorithm introduced in this section uses as a subroutine the parametric minimum cycle algorithm of Section 10.

Proposition 4.1 *A graph $G = (V, E, \boldsymbol{f})$ with vector weights (see Problem 1.1) has a zero-cycle (see Definition 2.3) if and only if it has a connected zero-circulation.*

Proof: Note that if there exists a connected zero-circulation then there exists a *rational* connected one. Hence, there exists an integral connected zero-circulation which is equivalent to a zero-cycle (see Remark 2.2). ∎

Definition 4.2 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 $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_d)^T$ for which the scalar-weighted graph $(V, E, \boldsymbol{f}^T\boldsymbol{\lambda})$ has no negative cycles.

ii. A nonzero vector $\boldsymbol{\lambda} \in \operatorname{rel\,int} \mathcal{K}$ (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 $(V, E, \boldsymbol{f}^T\boldsymbol{\lambda})$ has only positive cycles is called a *witness* for G.

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

Remark 4.3 The cone $\mathcal{K}$ can be described as the projection on the $\boldsymbol{\lambda}$-space (R^d) of a cone in R^{n+d} (the space of $(\pi_1, \ldots, \pi_n, \boldsymbol{\lambda})$) which is characterized by the inequalities:

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

Note that the system of inequalities above is the linear programming dual of the zero-circulation problem.

Definition 4.4

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 $\boldsymbol{\lambda} \neq \mathbf{0}$ (i.e., $\boldsymbol{\lambda} \in \mathrm{rel\,int}\,\mathcal{K}$), 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 4.5 *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

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

By Remark 4.3, 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 $(V, E, \boldsymbol{f}^T \boldsymbol{\lambda})$, so $\boldsymbol{\lambda} \in \mathcal{K}$. ▮

Theorem 4.6

i. $\mathrm{ORTH}(G, \boldsymbol{\lambda})$ *is independent of* $\boldsymbol{\lambda}$*, and hence will be denoted by* $\mathrm{ORTH}(G)$*. In fact,* $\mathrm{ORTH}(G)$ *is the lineality space of* $\mathrm{CIRC}(G)$*. (In case* $\mathcal{K} = \{\mathbf{0}\}$*, define* $\mathrm{ORTH}(G)$ *to be the entire* R^d*.)*

ii. $\mathrm{ORTH}(G) = (\mathrm{lin}\,\mathcal{K})^{\perp}$*, that is, 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 Section 5. ∎

The zero-cycle detection algorithm partitions the graph recursively into node disjoint subgraphs. The tree structure defined by this partitioning process, with subgraphs as nodes, is referred to as the *decomposition tree* of the graph G. In this partition, the subgraphs are the connected components of a "maximal" (in the sense of the number of active edges) zero-circulation. This definition implies that a zero-cycle exists in G if and only if a zero-cycle exists at least in one of the subgraphs which G is partitioned into. If a subgraph is not partitioned any further, it is a "leaf" of the decomposition tree, and for this subgraph the algorithm determines the existence of a zero-cycle directly. In [17] and [18] this partition is computed by solving a set of linear programming problems in order to decide for each edge whether or not it is active in any zero-circulation in G. The subgraphs are the connected components induced by the active edges. In this paper, the computation of the partition is done differently by an algorithm that gives strongly polynomial time bounds.

For a given graph $G = (V, E, \boldsymbol{f})$ the algorithm first tries to find a witness (if a witness is found a zero-cycle does not exist and we stop). In case a witness does not exist, a separating vector is computed. The computation of a witness or a separating vector is done by using the parametric minimum cycle algorithm developed in Section 10. The algorithm then proceeds to compute a partition of G using the separating vector found in the previous step. If the partition has only one subgraph, it is shown that a zero-cycle exists in G; otherwise, the algorithm proceeds recursively on the subgraphs. Note that a witness or a separating vector can be computed by solving linear programming problems. The difficulty is to find a strongly polynomial time solution.

In the rest of this section we first discuss the two subroutines used by the algorithm and then proceed to the algorithm itself (Subsection 4.3). The first subroutine is the computation of a witness or a separating vector (Subsection 4.1). The second (Subsection 4.2) is the partitioning of the graph when a separating vector is given.

4.1. Computing a witness or a separating vector

Problem 4.7 Given is a graph $G = (V, E, \boldsymbol{f})$. Find a witness for G (see Definition 4.2) if one exists; otherwise, find a separating vector $\boldsymbol{\lambda}$ or conclude that no such vector exists,[3] and provide a collection $\mathcal{C}$ of circulations with vector-values $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^r$ along with a set of positive numbers $\alpha_1, \ldots, \alpha_r$ such that $r = O(d)$, $\text{cone}\{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^r\} \supseteq \text{ORTH}(G)$, and $\sum_{i=1}^{r} \alpha_i \boldsymbol{c}^i = \mathbf{0}$.

Remark 4.8 The collection $\mathcal{C}$ is used to compute an explicit zero-cycle if one exists. It enables us to construct a circulation of any given value $\boldsymbol{c}' \in \text{ORTH}(G)$. The decision problem (existence of a zero-cycle) can be solved even if $\mathcal{C}$ is not given.

Proposition 4.9 *Problem 4.7 can be solved using three applications of the parametric minimum cycle algorithm on G with $d-1$ parameters.*

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

Proof: Deferred to Section 8. ∎

The following proposition is used for the proof of Proposition 4.9.

Proposition 4.10 *Given vectors $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^r \subset R^d$, and a subspace $S \subset R^d$, the following two conditions are equivalent:*

i. For every $\boldsymbol{\lambda} \notin S$,

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

ii. $\text{cone}\{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^r\} \supseteq S^\perp$.

Proof: The equivalence follows from Farkas' Lemma (see Proposition 4.12). First we assume (i) and show that (ii) is implied. Consider $\boldsymbol{z} \in S^\perp$. If a vector $\boldsymbol{y} \in R^d$ is such that $\boldsymbol{y}^T \boldsymbol{z} < 0$, then obviously $\boldsymbol{y} \notin S$. The latter, combined with (i) gives the left hand side condition on Farkas' Lemma. Therefore, from the right hand side we have $\boldsymbol{z} \in \text{cone}\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r\}$.

We show that (ii) implies (i). Assume that $\boldsymbol{z} \in S^\perp \Rightarrow \boldsymbol{z} \in \text{cone}\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r\}$. It follows from Farkas' Lemma that for all $\boldsymbol{z} \in S^\perp$, we have $(\forall \boldsymbol{y} \in R^d)\boldsymbol{y}^T \boldsymbol{z} < 0 \Rightarrow \min\{\boldsymbol{y}^T \boldsymbol{c}_i\} < 0$. Consider a vector $\boldsymbol{\lambda} \notin S$. There must exist $\boldsymbol{z} \in S^\perp$ such that $\boldsymbol{z}^T \boldsymbol{\lambda} < 0$ (otherwise, $\forall \boldsymbol{z} \in S^\perp, \boldsymbol{z}^T \boldsymbol{y} = 0$ in contradiction to $\boldsymbol{\lambda} \notin S$). We have $\boldsymbol{z}^T \boldsymbol{\lambda} < 0$. Therefore it follows from the left hand side of Farkas' Lemma that $\min\{\boldsymbol{\lambda}^T \boldsymbol{c}_i\} < 0$. ∎

Corollary 4.11 *Let the vectors $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^r$ be circulation values. If for every vector $\boldsymbol{\lambda} \notin \text{lin}\,\mathcal{K}$, $\min\{\boldsymbol{\lambda}^T \boldsymbol{c}^1, \ldots, \boldsymbol{\lambda}^T \boldsymbol{c}^r\} < 0$, then $\text{cone}\{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^r\} \supseteq \text{ORTH}(G)$.*

Proof: Take $S = \text{lin}\,\mathcal{K}$, and recall from Theorem 4.6 part (ii) that $\text{ORTH}(G) = (\text{lin}\,\mathcal{K})^\perp$. ∎

Proposition 4.12 *[Farkas' Lemma [9]] For any vectors $\boldsymbol{z}, \boldsymbol{c}_i \in R^d$, $i = 1, \ldots, r$,*

$$(\forall \boldsymbol{y} \in R^d)(\boldsymbol{y}^T \boldsymbol{z} < 0 \Rightarrow \min_i\{\boldsymbol{y}^T \boldsymbol{c}_i\} < 0) \Leftrightarrow \boldsymbol{z} \in \text{cone}(\boldsymbol{c}_i) \ .$$

4.2. Computing the partition

After computing a separating vector, the zero-cycle detection algorithm proceeds to compute a partition of the graph. In this subsection we define this partition, and discuss some of its properties. We also present the algorithm that computes the partition when the separating vector is given.

The essence of the following proposition is mentioned in [18].

Proposition 4.13 *Let $G = (V, E, w)$ be a scalar-weighted graph with no negative cycles. Using one application of an all-pairs shortest path algorithm we can find vertex disjoint subgraphs $G_1, \ldots, G_q$ of G with the following properties. Edges or vertices that are not active in any zero-cycle of G are not contained in any of the G_i's. Two vertices u and v are in the same G_i if and only if there exists a (scalar) zero-cycle of G in which both u and v are active.*

Proof: Apply an all-pairs shortest path algorithm to compute the distance d_{uv} between all pairs of vertices $u, v \in V$. Two vertices u, v are in the same subgraph G_i if and only if $d_{uv} + d_{vu} = 0$. If $d_{vv} > 0$, then v is not a part of a zero-cycle and does not belong to any G_i.

In order to identify all the edges that participate in some zero-cycle, do the following. Select arbitrarily some vertex w and use a single-source shortest path algorithm to compute the distances π_v $(v \in V)$ from the vertex w to all other vertices. For every edge (u, v) define,

$$\delta_{uv} \equiv \pi_u - \pi_v + d_{uv} \geq 0 \ .$$

Determine that (u, v) is an active edge if and only if $\delta_{uv} = 0$. ∎

Remark 4.14 Each component of the partition of Proposition 4.13 contains a zero-cycle where all the vertices of the component are active. This zero-cycle can be constructed easily from the shortest paths.

Proposition 4.15 *Suppose $\boldsymbol{\lambda}$ is a separating vector of $G = (V, E, \boldsymbol{f})$. Consider the scalar weights $w = \boldsymbol{f}^T\boldsymbol{\lambda}$ on the edges of G. Observe that by the definition of a separating vector, there are no negative cycles in the scalar weighted (V, E, w). Let $G_1, \ldots, G_q$ be the partition of G into subgraphs as defined in Proposition 4.13, relative to the scalar weights w. 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 $(V, E, \boldsymbol{f}^T\boldsymbol{\lambda})$. By the definition in Proposition 4.13, all the vertices active in $\boldsymbol{x}$ are in the same component G_i and hence $\boldsymbol{x}$ is a vector zero-cycle of G_i. ∎

Proposition 4.16 *If* CIRC(G) *contains a nontrivial linear subspace then a nontrivial (vector) zero-circulation exists in G.*

Proof: The proof is immediate. ∎

Proposition 4.17 *If $\mathcal{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r\} \subset R^\ell$ and $\alpha_i > 0$ $(i = 1, \ldots, r)$ are such that $\sum_{i=1}^r \alpha_i \boldsymbol{c}_i = \boldsymbol{0}$, then for any $\boldsymbol{v} \in R^\ell$, it takes $O(\ell^2 r)$ time either to find nonnegative rational constants $\beta_1, \ldots, \beta_r$ such that $\boldsymbol{v} = \sum \beta_i \boldsymbol{c}_i$, or to recognize that no such constants exist.*

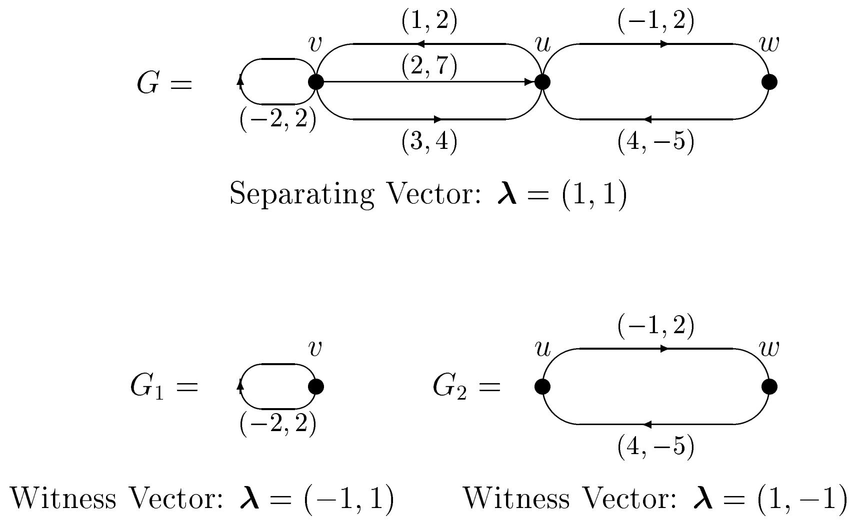


Figure 2: Example of the decomposition of a graph G

Proof: Express $\boldsymbol{v}$ as a linear combination of the vectors in $\mathcal{C}$ by solving the linear system of equations $\boldsymbol{v} = \sum \gamma_i \boldsymbol{c}_i$. This system has ℓ equations and r variables, and thus can be solved by Gaussian eliminations using $O(\ell^2 r)$ operations. If γ_i are nonnegative take $\beta_i = \gamma_i$; otherwise, denote $\alpha = \min_{1 \le i \le r} \alpha_i$, $\gamma = \min_{1 \le i \le r} \gamma_i$ and let $\beta_i = \gamma_i - (\gamma/\alpha)\alpha_i$. It is easy to verify that β_i $(1 \le i \le r)$ are nonnegative and $\sum_{i=1}^{r} \beta_i \boldsymbol{c}_i = \mathbf{0}$. ∎

Proposition 4.18 *Let $\boldsymbol{\lambda}$ be a separating vector of $G = (V, E, \boldsymbol{f})$. Let $G_1, \ldots, G_q$ be the partition of G into subgraphs (as defined in Proposition 4.13), relative to the scalar weights $\boldsymbol{f}^T\boldsymbol{\lambda}$. If the partition constitutes a single subgraph (i.e, $q = 1$), then G has a (vector) zero-cycle.*

Proof: If G has a single component relative to $\boldsymbol{f}^T\boldsymbol{\lambda}$, then all active vertices and edges are contained in G_1. Observe that all cycles with vector value in $\mathrm{ORTH}(G)$ are scalar zero-cycles relative to $\boldsymbol{f}^T\boldsymbol{\lambda}$. There exists a scalar zero-cycle in $(V, E, \boldsymbol{f}^T\boldsymbol{\lambda})$ in which all the vertices of G_1 are active. Thus, there exists a value $\boldsymbol{c} \in \mathrm{ORTH}(G)$ which is attained at a circulation where all the vertices in G_1 are active, so this circulation is connected. By Theorem 4.6 and Proposition 4.16, there exists a circulation, not necessarily connected, whose value is $-\boldsymbol{c}$. The active vertices in this circulation must be contained in G_1. By combining the connected circulation supporting $\boldsymbol{c}$ with the one supporting $-\boldsymbol{c}$, we obtain a connected (nontrivial) zero-circulation, that is, a zero-cycle of G. ∎

Remark 4.19 Suppose we have a set $\mathcal{C}$ of (vector) cycle values such that $\mathrm{cone}\,\mathcal{C} \supseteq \mathrm{ORTH}(G)$. The vector zero-cycle of Proposition 4.18 can be explicitly constructed as follows. We compute a connected zero-circulation relative to the scalar weights $\boldsymbol{f}^T\boldsymbol{\lambda}$, in which all the vertices are active. The vector value of this circulation is $\boldsymbol{c} \in \mathrm{ORTH}(G)$ (see Remark 4.14). It follows from Proposition 4.17 that we can construct a circulation with value $-\boldsymbol{c}$. The combination of the two circulations is a connected zero-circulation.

Remark 4.20 Assume the graph G does not have a separating vector (that is, $\mathrm{ORTH}(G) = R^d$). If we are given a set $\mathcal{C}$ of cycle values whose conic hull equals R^d, then a zero-circulation can be constructed as follows. Find a cycle in which all the vertices are active (G is strongly connected). Denote the value of this cycle by $\boldsymbol{c}$. It follows from Remark 4.14 that we can find a circulation with value $-\boldsymbol{c}$. The combination of the two circulations is a (nontrivial) connected zero-circulation.

Remark 4.21 Remarks 4.19 and 4.20 discuss the construction of an explicit zero-cycle. Observe that if $\mathcal{C}$ is of size $O(d)$, then the time complexity of constructing a zero-cycle is $O(d^3)$ (see Proposition 4.17).

Proposition 4.22 *A witness for G exists if and only if G does not have a non-trivial zero-circulation.*

Proof: The proof is immediate. ∎

4.3. The algorithm

Algorithm 4.23 [zero-cycle detection]

i. Run an algorithm for Problem 4.7 on G (see Proposition 4.9). If a witness for G is found then stop. Otherwise, find a collection $\mathcal{C}$ of circulation values such that $\operatorname{cone}\mathcal{C} \supseteq \mathrm{ORTH}(G)$, 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 4.20 for an explicit construction of the zero-cycle). Otherwise,

ii. Construct the partition of G which is defined in Propositions 4.13 and 4.15. If the partition is empty then G does not have a zero-cycle. Otherwise,

iii. If there is only one component (i.e., $q = 1$), then by Proposition 4.18, $G = (V, E, \boldsymbol{f})$ has a zero-cycle (see Remark 4.19 for how to find the zero-cycle explicitly).

iv. Run the zero-cycle detection algorithm on $G_1, \ldots, G_q$ (recursively). By Proposition 4.15, G has a zero-cycle if and only if at least one of $G_1, \ldots, G_q$ has one.

In the rest of the present section we prove the correctness and analyze the complexity of Algorithm 4.23.

Proposition 4.24 *If G is partitioned into $G_1, \ldots, G_q$ (see Proposition 4.15) and for some G_i, $\dim(\mathrm{ORTH}(G_i)) = \dim(\mathrm{ORTH}(G))$, then G_i will not be partitioned any further by the algorithm.*

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

Corollary 4.25 *Algorithm 4.23 terminates after at most $d-1$ phases of partitioning.*

Proposition 4.26 *The time complexity of the zero-cycle detection algorithm for a graph $G = (V, E, \boldsymbol{f})$ (where $\boldsymbol{f}$ is d-dimensional) is dominated by the complexity of $3d$ applications of solving Problem 4.7 on G.*

Proof: First, observe that the complexity of explicitly constructing a zero-cycle (see Remark 4.21) is dominated by the complexity of the rest of the algorithm. Consider the recursion tree of Algorithm 4.23. The recursion tree corresponds to the decomposition tree of the graph G. By Corollary 4.25 this tree has d levels. Each level is a phase of partitioning a collection of subgraphs $G_1, \ldots, G_q$, with total number of $n = |V|$ vertices. The total computation done at such a phase is solving Problem 4.7 for each subgraph G_i, and then, if needed, partitioning it as described in Proposition 4.13. Observe that the time and processor complexities of solving Problem 4.7 and partitioning all the subgraphs at a certain phase, are dominated by the complexities of the same computation done on the graph G. Recall (see Proposition 4.13) that a partitioning operation amounts to an all-pairs shortest path computation. Therefore, the complexity of computing the partition is dominated by the complexity of solving Problem 4.7. It follows that at each level of the tree, the total complexity of the computation is dominated by the complexity of solving Problem 4.7 on G. ∎

Theorem 4.27 *The complexity of the zero-cycle detection algorithm for a graph $G = (V, E, \boldsymbol{f})$ (where $\boldsymbol{f}$ is d-dimensional) is essentially dominated by $3d$ applications of the parametric minimum cycle algorithm presented in Section 10, when applied to instances with $d-1$ parameters which involve the graph G.*

Proof: The proof follows from Propositions 4.9 and 4.26. ∎

5. Geometric lemmas

In this section we give the necessary lemmas which establish the proof of Theorem 4.6. The reader is referred to [11] for background.

For any subset C of R^d, denote

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

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

The following proposition states well known facts about cones [10].

Proposition 5.1

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

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

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

Proposition 5.2

i. *If $L \subset R^d$ is a linear subspace, then $L^+ = L^\perp$.*

ii. *For every cone C we have $C^+ = C_p^+ \cap L^\perp$, where $C = L \oplus C_p$ as above.*

Proof: The proof of part i follows from the fact that if L is a linear subspace and $\boldsymbol{y} \in L^+$, then $\boldsymbol{y}^T \boldsymbol{d} = 0$ for all $\boldsymbol{d} \in L$. Part ii follows from the equality $C^+ = C_p^+ \cap L^+$ and from part i. ∎

Proposition 5.3 *If C is a pointed cone, C^+ is of full dimension.*

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

Proposition 5.4

$$\dim(C^+) = \dim(L^\perp)\ .$$

Proof: It follows from Proposition 5.3 that C_p^+ is of full dimension in the space $\operatorname{lin} C_p$. Recall that $\operatorname{lin} C_p \subset L^+$. The proof follows from Proposition 5.2 part ii. ∎

Proposition 5.5 *If $\boldsymbol{\lambda} \in \operatorname{rel\,int}(C^+)$, then for all $\boldsymbol{c} \in C_p$ ($\boldsymbol{c} \neq \boldsymbol{0}$), $\boldsymbol{\lambda}^T \boldsymbol{c} > 0$.*

Proof: From the proof of Proposition 5.3 and Proposition 5.4 it follows that there exists a vector $\boldsymbol{v}$ such that for every unit-vector $\boldsymbol{u} \in C_p$, $\boldsymbol{v}^T \boldsymbol{u} \geq 1$, and for every $\boldsymbol{w} \in L$, $\boldsymbol{v}^T \boldsymbol{w} = 0$. The set C^+ is full dimensional relative to $L^\perp$. Therefore, if $\boldsymbol{\lambda}^T \boldsymbol{c} = 0$ for some $\boldsymbol{c} \in C_p$ ($\boldsymbol{c} \neq \boldsymbol{0}$), then $\boldsymbol{\lambda} \notin \operatorname{rel\,int} C^+$. ∎

Let $C = \mathrm{CIRC}(G)$ (see Definition 4.4). Let L and C_p be as in Proposition 5.1. Let $\mathcal{K}$ be as in Definition 4.2.

Proposition 5.6

$$\mathcal{K} = C^+$$

Proof: Immediate from Proposition 4.5. ∎

Proposition 5.7 *For every* $\boldsymbol{\lambda} \in \operatorname{rel\,int}(\mathcal{K})$*, the set* $\mathrm{ORTH}(G, \boldsymbol{\lambda})$ *is equal to the linear subspace* L*.*

Proof: It follows from Propositions 5.5 and 5.6, that if $\boldsymbol{\lambda} \in \operatorname{rel\,int} \mathcal{K}$ and $\boldsymbol{c} \in C$ are orthogonal, then $\boldsymbol{c} \in L$. On the other hand, since $\mathcal{K} \subseteq L^{\perp}$ (see Proposition 5.2), if $\boldsymbol{c} \in L$ and $\boldsymbol{\lambda} \in \mathcal{K}$, then $\boldsymbol{\lambda}^T \boldsymbol{c} = 0$. ∎

6. Applications of the zero-cycle detection algorithm

We first introduce some notation for the discussion of periodic graphs (see Section 1). For a given $G = (V, E, \boldsymbol{f})$ where $\boldsymbol{f} : E \to Z^d$ and $V = \{1, \ldots, n\}$, denote by $G^* = (V^*, E^*)$ the infinite periodic graph that is defined by G as explained in Section 1. We refer to G^* as a d-dimensional periodic graph. Formally,

$$V^* = Z^d \times V = \{(\boldsymbol{z}, i) \; : \; \boldsymbol{z} \in Z^d, i \in V\} ,$$

$$E^* = Z^d \times E = \{(\boldsymbol{z}, e) \; : \; \boldsymbol{z} \in Z^d, e \in E\} .$$

If $e = (i, j)$ we also identify the edge $(\boldsymbol{z}, e)$ with the pair $((\boldsymbol{z}, i), (\boldsymbol{z} + \boldsymbol{f}(e), j))$.

The zero-cycle detection algorithm of Section 4 computes the decomposition tree of an input graph $G = (V, E, \boldsymbol{f})$ and the separating vectors for all the subgraphs sitting at the nodes of this tree. Recall that this computation can be performed by solving polynomially many LP programs. In Section 4 we presented strongly polynomial time solution when the dimension d is fixed. We discuss two problems which can be solved easily when the decomposition tree and the separating vectors are given.

6.1. Scheduling

The first application is the problem of scheduling a system of uniform recurrence equations. The problem was raised by Karp, Miller and Winograd [17] and algorithms that solve it were given in [17, 27, 26]. These algorithms are stated in terms of solving systems of linear inequalities and therefore do not establish strong polynomiality. We show that the knowledge of the decomposition tree and the separating vectors enables us to produce an immediate solution. Hence, we obtain strongly polynomial complexity bounds.

A system of uniform recurrence equations is a finite set of relations among functions $F_i : Z^d \to R$ $(i = 1, \ldots, n)$,

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

(The definition can be easily extended to accommodate the case where the value of some F_i is related directly to more than one value of some F_j.) Such a system can be modeled by a finite graph $G = (V, E, \boldsymbol{f})$, where the functions F_i correspond to the vertices. It is called the "dependence graph" in [17]. This graph defines a periodic graph G^* whose vertices correspond to the function values $F_i(\boldsymbol{z})$, $(i = 1, \ldots, n,\ \boldsymbol{z} \in Z^d,\ \boldsymbol{z} \geq \mathbf{0})$. The direct dependencies among function values are modeled by the edges of G^* as follows. If $e = (i, j) \in E$ and $\boldsymbol{f}(e) = \boldsymbol{a}$, then the evaluation of $F_i(\boldsymbol{z})$ requires the knowledge of $F_j(\boldsymbol{z} - \boldsymbol{a})$ (and having all the required knowledge is sufficient). For simplicity, suppose it takes one time unit to evaluate the ψ_i's, that is, given all the required knowledge, it takes one time unit to calculate the function value.

For "efficient" parallel evaluation of the function values, one would like to find large sets of "independent" values, that is, sets of values that can be computed simultaneously. Here, two values are independent if there is no directed path in G^* between their corresponding vertices. A set of values is called independent if every two members of the set are independent. The problem of finding a maximal independent set of values is not easy, since the problem of deciding whether there exists a directed path in G^*, from $(\boldsymbol{z}^1, i^1)$ to $(\boldsymbol{z}^2, i^2)$ is $\mathcal{NP}$-Complete, even for one-dimensional periodic graphs (see [25]).

A subspace $S_i \subset V_i$ is said to be *independent* if the values $F_i(\boldsymbol{z})$ $(\boldsymbol{z} \in S_i)$ are independent. Interestingly, one can compute in polynomial time maximal independent subspaces [17, 27]. Let $V_i = Z^d \times \{i\}$, $i = 1, \ldots, n$. A maximal independent subspace S_i gives a partition of V_i into independent "isomorphic" flats S_v $(\boldsymbol{v} \in S_i^{\perp})$, where $S_v = (\boldsymbol{v}, 0) + S_i = \{(\boldsymbol{v} + \boldsymbol{z}, i) \mid \boldsymbol{z} \in S_i\}$.

The algorithm for maximal independent subspaces finds for each i, $i = 1, \ldots, n$, a matrix M_i of dimensions $(d - \dim(S_i)) \times d$, whose rows are linearly independent, and whose null space is S_i. Following [27], the matrix M_i is called the *scheduling matrix* of i.

An algorithm that computes the scheduling matrix for the special case where the decomposition tree of G is of depth one was given in [17]. In this special case, assuming that G is strongly connected, the scheduling matrix would be the same for all i. In fact, $M = M_1 = \cdots = M_n$ consists of a single vector $\boldsymbol{v} \in R^d$, which is computed by solving a set of linear programming problems. Obviously, the null space of $\boldsymbol{v}$ is of dimension $d - 1$. Any solution of the set of linear programs used in [17] is in the interior of the set $\{\boldsymbol{v} \mid (\forall \boldsymbol{c} \in \mathrm{CIRC})(\boldsymbol{v}^T \boldsymbol{c} \geq \mathbf{0})\}$. Observe that every such $\boldsymbol{v}$ is a separating vector (see Definition 4.2) for G. Moreover, it is a witness since the decomposition tree has depth one. A more formal statement follows.

Proposition 6.1 *Suppose $G = (V, E, \boldsymbol{f})$ is strongly connected and G^* has no cycles. If the decomposition tree of G is of depth one and $\boldsymbol{v}$ is a separating vector (and hence a witness) for G, then the null space of $\boldsymbol{v}$ is a maximal independent subspace.*

Proof: The null space of $\boldsymbol{v}$ has dimension $d - 1$. Therefore, if it is independent, it must be maximal. It remains to show that the null space of $\boldsymbol{v}$ is independent. First, we claim that for any subspace S, if $S \cap \mathrm{CIRC} = \{\mathbf{0}\}$, then for all i $(i = 1, \ldots, n)$ and for all $\boldsymbol{u} \in S^{\perp}$, the set $\{(\boldsymbol{u} + \boldsymbol{z}, i) \mid \boldsymbol{z} \in S\}$ is independent. To prove this claim, assume to the contrary that for some $\boldsymbol{b} \neq \mathbf{0}$ in S and some i, there exists a directed path in G^* from $(\boldsymbol{u}, i)$ to $(\boldsymbol{u} + \boldsymbol{b}, i)$. Thus, there is a cycle in G with vector weight $\boldsymbol{b}$, which implies $\boldsymbol{b} \in \mathrm{CIRC}$, and hence a contradiction. Second, we claim that the intersection of the null space of any witness $\boldsymbol{v}$ with CIRC is equal to $\{\mathbf{0}\}$. To prove the second claim, observe that if witness exists, then the cone CIRC of possible circulation values is pointed. Therefore, since ORTH(G) is the lineality space of CIRC(G) (see Theorem 4.6), we have $\dim(\text{ORTH}(G)) = 0$. Observe that ORTH(G) is the intersection of the null space of any separating vector with CIRC (see Definition 4.4). Assuming the second claim holds, the first claim implies that $S(\boldsymbol{v})$ is an independent subspace. This concludes the proof of the proposition. ∎

Roychowdhury and Kailath [27] generalized the result of [17] and gave an algorithm which computes the scheduling matrices for any dependence graph G, where the decomposition tree is not necessarily of depth one. In the general case, the scheduling matrices M_i ($i = 1, \ldots, n$) need not be all identical, or even of the same dimension. Their algorithm first computes the decomposition tree of G, along with the separating vectors of the subgraphs sitting at the nodes of the decomposition tree. Subsequently, the algorithm uses these separating vectors to construct the scheduling matrices. The latter construction is trivial (see Definition 6.2 and Proposition 6.3).

The algorithm of Roychowdhury and Kailath [27] (like the algorithm for the depth one case of [17]) is based on solving $O(m)$ sets of linear inequalities and therefore, does not establish strong polynomiality. Recall that the zero-cycle detection algorithm computes the decomposition tree of G along with a collection of separating vectors that correspond to the subgraphs of G sitting at the nodes of the decomposition tree. Hence, the results obtained here imply that the scheduling matrices can be computed within the time bounds of the zero-cycle detection algorithm, that is, in $\mathcal{NC}$ and strongly polynomial time.

When the decomposition tree of G and the separating vectors at its nodes are given, it is easy to compute the scheduling matrices [27]:

Definition 6.2 Let $G = (V, E, \boldsymbol{f})$ be a dependence graph, where $V = \{1, \ldots, n\}$. Consider the decomposition tree of G, and the separating vectors of the subgraphs sitting at the nodes of the tree. For each vertex $i \in V$, consider the set of subgraphs that are sitting in the decomposition tree and of which i is a member. This set of subgraphs corresponds to a path in the decomposition tree. Define the *path* of a vertex i to be the ordered set of subgraphs along this path.

Proposition 6.3 *For a given dependence graph $G = (V, E, \boldsymbol{f})$, the scheduling matrix M_i of a vertex i is the matrix whose rows are the separating vectors of the subgraphs along the path of i.*

6.2. Strong connectivity

Another application of the zero-cycle detection algorithm is the following. Given a dependence graph $G = (V, E, \boldsymbol{f})$, compute the strongly connected components of G^*, that is, find graphs $G_i = (V_i, E_i, \boldsymbol{f}_i)$ such that the graphs G_i^* are isomorphic to each of the strongly connected components of G^*. The problem of strong connectivity on periodic graphs was first raised by Orlin [25]. However, his paper is concerned only with one-dimensional periodic graphs (i.e., when $f : E \to Z$ is a scalar function). Orlin gave an algorithm for the one-dimensional case that does not seem to generalize to higher dimensions. This is not surprising since the strong connectivity problem is obviously at least as hard as zero-cycle detection in G: the periodic graph G^* does not have a cycle if and only if it has no nontrivial strongly connected components. We show that the zero-cycle detection algorithm can be used to compute the strongly connected components of G^*. More specifically, we prove that the strongly connected components are the connected components of the subgraphs sitting at the leaves of the decomposition tree.

The relation between the decomposition tree of the dependence graph G and the strongly connected components of G^* is given by the following proposition:

Proposition 6.4 *The strongly connected components of G^* are precisely the connected components of the graphs G_i^*, where G_i is any subgraph of G sitting at a leaf of the decomposition tree of G.*

Proof: The proof is immediate from the following two claims:
The first claim is that every strongly connected component of G^* must be contained in some G_i^*. This is obvious since all zero-cycles of G must be contained in one of the G_i's.
The second claim is that every connected component of a G_i^* is strongly connected. It suffices to show that every path $(e_1, e_2, \ldots, e_\ell)$ in G_i is part of a zero-cycle. Since every e_j $(j = 1, \ldots, \ell)$ participates in a zero-cycle C_i, the cycle $\bigcup_{j=1}^{\ell} C_i$ is a zero-cycle which contains the path. ∎

Remark 6.5 A strongly connected component $S \subset V^*$ of G^* is such that if $(\boldsymbol{a}, i), (\boldsymbol{b}, i) \in S$, then $(\boldsymbol{a} + \alpha(\boldsymbol{a} - \boldsymbol{b}), i) \in S$ for any integer α.

It follows from Proposition 6.4 that for a given dependence graph $G = (V, E, \boldsymbol{f})$, we can compute a collection of dependence graphs $\hat{G}_i$ $(i = 1, \ldots, r)$, such that the graphs $(\hat{G}_i)^*$ are isomorphic to the strongly connected components of G^*. This is done as follows.

Algorithm 6.6 [Strongly connected components of G^*]

i. Compute the decomposition tree of G. Denote by G_i $(i = 1, \ldots, r)$ the subgraphs sitting at the leaves of the decomposition tree.

ii. For each G_i, compute a dependence graph $\hat{G}_i$, such that $(\hat{G}_i)^*$ is isomorphic to each of the the connected components of G_i^*.

Step i of the algorithm involves the computation of the decomposition tree, that is, the zero-cycle detection algorithm. Step ii involves the computation of the connected components of the G_i^*'s. An algorithm for computing connected components of a periodic graph is given in [6].

7. Discussion

The obvious open question that arises is whether Problem 1.1, where the dimension d is part of the input, can be solved in strongly polynomial time, and whether it is in the class $\mathcal{NC}$. It is interesting to note the following:

Proposition 7.1 *The problem of detecting a zero cycle (Problem 1.1) is $\mathcal{P}$-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 an $\mathcal{NC}$ reduction) to the problem of solving the following system:

$$(S) \qquad \begin{array}{c} \boldsymbol{Ax} = \boldsymbol{0} \\ \boldsymbol{0} \neq \boldsymbol{x} \geq \boldsymbol{0} \ , \end{array}$$

where $\boldsymbol{A} \in R^{m \times n}$. Consider a network consisting of n parallel edges from vertex s to vertex t and one edge from t to s. (It is a trivial matter to avoid parallel edges if this is desired.) Associate with the i'th edge the weight-vector given by the i'th column of $\boldsymbol{A}$, and associate with the reverse edge the zero vector. 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. ∎

In view of Proposition 7.1, the questions stated in the beginning of this section are equivalent to two famous and difficult open questions.

Recall that we considered zero-cycle which were not necessarily simple. Unfortunately, if simplicity of the cycle is added to the requirements, the problem becomes $\mathcal{NP}$-complete. Moreover, even the problem of recognizing whether a graph with scalar weights has a simple cycle with a total weight of zero is $\mathcal{NP}$-complete. This follows from the fact that the knapsack problem can be reduced to detecting a simple zero-cycle in a graph whose edges form a ring, where two consecutive vertices are connected with two parallel edges.

8. Parametric minimum cycle: preliminaries

In this section we define the parametric minimum cycle and the parametric minimum cycle-mean problems as instances of a special class of convex optimization problems. In Subsection 8.1 we discuss the "hyperplane query" subproblem. Subsection 8.2 contains material relevant for solving the second part of Problem 4.7. We suggest to skip Subsection 8.2 in first reading. In Subsection 8.3 we give the proof of Proposition 4.9, and hence, complete the reduction of the zero-cycle detection problem to solving instances of Problems 8.5 and 8.7.

The parametric minimum cycle and the parametric minimum cycle-mean problems are a generalization of the corresponding nonparametric problems. In instances of these problems with $d-1$ parameters, the edge weights are not constants but are linear functions of the parameters. When values (a vector in R_1^d) are assigned to the parameters, we get an instance of the nonparametric problem. We associate with each parametric instance a function of the form $g : R^{d-1} \to R$ which is a mapping from assignments of values for the parameters to the solution of the corresponding nonparametric problem. We shall see that these functions are piecewise linear and concave. A solution of a parametric problem is, roughly, an assignment of values to the parameters that maximizes the solution of the resulting non-parametric problem. This is equivalent to the problem of maximizing the function associated with the parametric instance.

We give some definitions and notations.

Definition 8.1

i. For a finite set $\mathcal{C} \subset R^d$, denote by $L_{\mathcal{C}} : R^d \to R$ the lower envelope of the linear functions that correspond to the vectors in $\mathcal{C}$, $L_{\mathcal{C}}(\boldsymbol{\lambda}) = \min_{\boldsymbol{c} \in \mathcal{C}} \boldsymbol{c}^T \boldsymbol{\lambda}$. The vectors $\boldsymbol{c} \in \mathcal{C}$, and interchangeably the linear functions $\boldsymbol{c}^T \boldsymbol{\lambda}$ are referred to as *pieces* of $L_{\mathcal{C}}$. If for $\boldsymbol{\lambda}' \in R^d$ and $\boldsymbol{c} \in C$ we have $\boldsymbol{c}^T \boldsymbol{\lambda}' = L_{\mathcal{C}}(\boldsymbol{\lambda}')$ we say that $\boldsymbol{c}$ is *active* at $\boldsymbol{\lambda}'$.

ii. Suppose $H \subset R^d$ is a flat, $F \subset R^d$ is the subspace parallel to it, and $\mathcal{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r\} \subset R^d$ is a set of vectors. A *balancing combination* of $\mathcal{C}$ relative to H is a positive linear combination $\sum_{i=1}^r \alpha_i \boldsymbol{c}_i$ which is orthogonal to F. If $H = R^d$ we say that $\sum_{i=1}^r \alpha_i \boldsymbol{c}_i = \mathbf{0}$ is a balancing combination of $\mathcal{C}$ if $\alpha_1, \ldots, \alpha_r > 0$.

Let R^d_δ denote the set of vectors $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_d)^T \in R^d$ such that $\lambda_d = \delta$.

Definition 8.2 For $g : R^d \to R$ ($g : H \to R$ where $H \subset R^d$ is a hyperplane), we introduce the following definitions and notations:

i. Denote by $\Lambda_g \equiv \Lambda$ (possibly $\Lambda = \emptyset$) the set of vectors $\boldsymbol{\lambda} \in R^d$ ($\boldsymbol{\lambda} \in H$) where $g(\boldsymbol{\lambda})$ is maximized.

ii. Denote by $\mathcal{K}_g \equiv \mathcal{K}$ the set of $\boldsymbol{\lambda} \in R^d$ ($\boldsymbol{\lambda} \in H$) such that $g(\boldsymbol{\lambda}) \geq 0$. Also, denote by $\mathcal{K}_\delta$ the set of $\boldsymbol{\lambda} \in \mathcal{K} \cap R^d_\delta$.

iii. An algorithm that computes the function g is called *piecewise affine*, if the operations it performs on intermediate values that depend on the input vector are restricted to additions, multiplications by constants, comparisons, and making copies.

iv. When $g = L_{\mathcal{C}}$ ($\mathcal{C} \subset R^d$), we say that $g' = L_{\mathcal{C}'}$ is a *weak approximation* of g, if the pieces of g' comprise a subset of the pieces of g ($\mathcal{C}' \subset \mathcal{C}$) and $\operatorname{aff} \Lambda_g = \operatorname{aff} \Lambda_{g'}$. The function $g' = L_{\mathcal{C}'}$ is a *minimal* weak approximation of g, if there is no proper subset $\mathcal{C}''$ of $\mathcal{C}'$ such that $L_{\mathcal{C}''}$ is a weak approximation of g.

The functions we consider are concave, piecewise linear, and are of the form $g : R^{d-1} \to R$ (sometimes we denote the range by R^d_δ). We also assume that each such function is given by a piecewise affine algorithm (we use the notation $\mathcal{A}$) that evaluates it at a given point. The parametric minimum cycle and parametric minimum cycle-mean problems are special cases of the following problem:

Problem 8.3 If $g(\boldsymbol{\lambda}) > 0$ for some $\boldsymbol{\lambda}$, then output such $\boldsymbol{\lambda}$; otherwise, find $\boldsymbol{\lambda} \in \operatorname{rel\,int} \Lambda$. We sometimes add the following requirement. If $g \leq 0$, find a subset $\mathcal{C}$ of the pieces of g, such that $L_{\mathcal{C}}$ is a minimal weak approximation of g, and find a balancing combination of $\mathcal{C}$ relative to R_1^d. We refer to this last task as the "optional" part of the problem.

Intuitively, the set $\mathcal{C}$ certifies that the function g does not exceed its maximum value. The advantage of considering it is that while the number of pieces of g may be vary large, the size of a minimal weak approximation is at most $2d$.

The scheme we present here to solve Problem 8.3 for the parametric minimum cycle is quiet general and can be applied to any concave function g as above.

Let $G = (V, E, \boldsymbol{f})$ be as in Problem 1.1. The vector-weights $\boldsymbol{f}(e)$ $(e \in E)$ are interpreted as linear functions of $d-1$ variables (parameters):

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

When $\boldsymbol{\lambda}$ is assigned with specific numerical values, we denote by $\boldsymbol{f}^T\boldsymbol{\lambda}$ the resulting set of scalar weights.

Parametric minimum cycle-mean :.

Definition 8.4 Consider $G = (V, E, \boldsymbol{f})$, where for $(i,j) \in E$, $\boldsymbol{f}(i,j) = \boldsymbol{c}_{ij} \in R^d$,

i. For a subset of edges $E' \subset E$, denote by $\boldsymbol{f}(E')$ the $(d-1)$-variable linear function $(1/|E'|)\sum_{e \in E'} \boldsymbol{f}(e)$.

ii. Denote by $g(\boldsymbol{\lambda})$ the minimum cycle-mean[4] relative to the scalar-weights $\boldsymbol{\lambda}^T\boldsymbol{c}_{ij}$.

Problem 8.5 [Parametric Minimum Cycle-Mean]
For a given graph $G = (V, E, \boldsymbol{f})$, if $g(\boldsymbol{\lambda}) > 0$ for some $\boldsymbol{\lambda}$, then output one such $\boldsymbol{\lambda}$; otherwise, find $\boldsymbol{\lambda} \in \operatorname{rel\,int} \Lambda$.

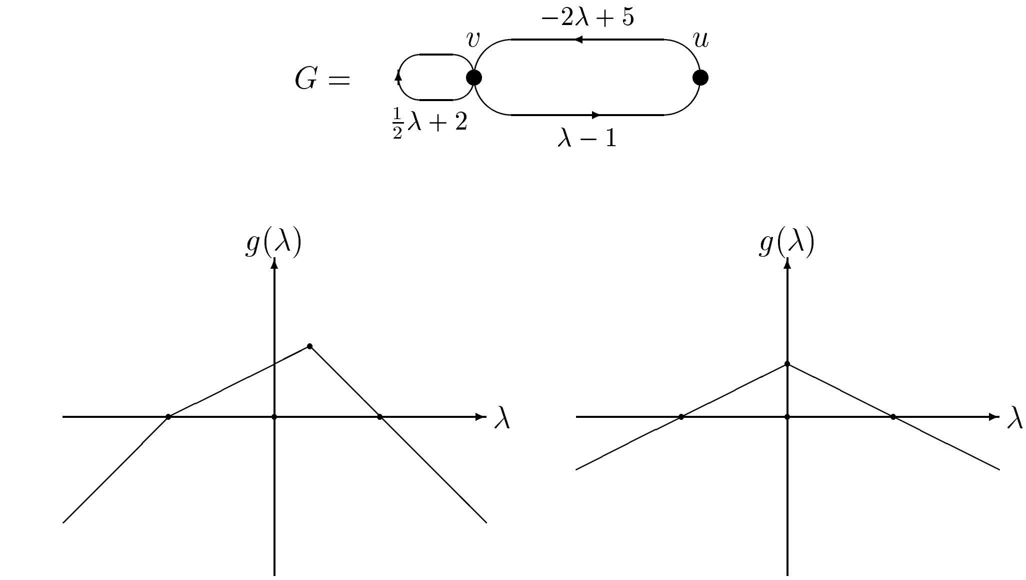
Parametric minimum cycle :.

Definition 8.6 Consider $G = (V, E, \boldsymbol{f})$, where for $e \in E$, $\boldsymbol{f}(e) = \boldsymbol{c}_e \in R^d$,

i. For $E' \subset E$, denote by $\boldsymbol{f}(E')$ the $(d-1)$-variable linear function $\sum_{e \in E'} \boldsymbol{f}(e)$.

ii. Let $C = C(\boldsymbol{\lambda})$ denote a cycle of at most n edges which minimizes the total scalar weight $\boldsymbol{\lambda}^T\boldsymbol{c}_e$. Denote $g(\boldsymbol{\lambda}) = \boldsymbol{f}(C)^T\boldsymbol{\lambda}$.

Problem 8.7 [Parametric Minimum Cycle]
For a given graph $G = (V, E, \boldsymbol{f})$, if $g(\boldsymbol{\lambda}) > 0$ for some $\boldsymbol{\lambda}$, then output any such $\boldsymbol{\lambda}$; otherwise, find $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} \Lambda$ and a collection $\mathcal{C} = \{C_1, \cdots, C_r\}$ of cycles , each of at most n edges, such that $L_{\{\boldsymbol{f}(C_i) \mid i \in \{1,\ldots,r\}\}}$ is a minimal weak approximation of g (see Definition 8.2), along with a balancing combination of the cycle values $\boldsymbol{f}(\mathcal{C})$ relative to R_1^d.



Minimum cycle of at most 2 edges in G
$g(\lambda) = \min\{\frac{1}{2}\lambda + 2, \lambda + 4, -\lambda + 4\}$

Minimum cycle-mean in G
$g(\lambda) = \min\{\frac{1}{2}\lambda + 2, -\frac{1}{2}\lambda + 2\}$

Figure 3: Example of g for a graph G with 2-dimensional weights

Note that the function g defined for both problems above is of the form $g = L_{\mathcal{C}}$, where $\mathcal{C}$ is the collection of all possible vector values of cycle-means (Problem 8.5) or cycles of at most n edges (Problem 8.7). Also note that g is concave and computable by a piecewise affine algorithm (see Figure 3 for an example of such functions). The parallel of the optional part in Problem 8.3 is omitted in the statement of Problem 8.5. In Problem 8.7, however, the optional part corresponds to the collection of cycles. We explain the purpose of considering the optional part. Recall that in Section 4 the problem of detecting zero-cycles was reduced to solving Problem 4.7. Solving the latter problem amounts to (i) computing a separating vector, which is needed to solve the decision problem, and (ii) finding a collection of cycles, which is used for computing an explicit zero-cycle when one does exist. Proposition 4.9 tied the solution of Problem 4.7 to solving instances of the parametric minimum cycle and the parametric minimum cycle-mean problems. The proof of Proposition 4.9 was deferred to this section. We will see that in order to compute a separating vector (and hence decide existence of a zero-cycle) it suffices to consider Problems 8.5 and 8.7 where the optional part is omitted.

[4]The *minimum cycle-mean* is the minimum, over all simple cycles, of the total weight of a cycle divided by the number of its edges.

8.1. The oracle problem

We find the following subproblem useful for the solution of Problem 8.3. The goal in Problem 8.3 is to maximize a function over some domain. Intuitively, the following is a useful tool: 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 a procedure that solves this problem as an *oracle*. Clearly, such an "oracle" would enable us to perform binary searches over the $\boldsymbol{\lambda}$ space. In order to achieve strongly polynomial bounds, however, we use a more sophisticated approach where the number of hyperplane queries needed is of the order of the number of comparisons done by $\mathcal{A}$. Each hyperplane query can be resolved by one oracle call. By using the multi-dimensional search techniques and exploiting the parallelism of $\mathcal{A}$ we can do better: We present a solution where the number of oracle calls performed is a polylog in the number of comparisons (hyperplane queries needed).

The function g is a concave piecewise linear mapping from R_1^d 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 [23] and is developed below for the special structure of our problem.

Problem 8.8 Given is a concave function $g : R_1^d \to R$ 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, output any such $\boldsymbol{\lambda}$; otherwise,

ii. find $\boldsymbol{\lambda} \in H \cap \operatorname{rel\,int}(\Lambda)$, if such a $\boldsymbol{\lambda}$ exists, and generate a solution of Problem 8.3 for g; otherwise, if $H \cap \operatorname{rel\,int} \Lambda = \emptyset$,

iii. recognize which of the two halfspaces determined by H either intersects $\operatorname{rel\,int} \Lambda$, or has g unbounded on it.

We refer to a procedure that solves Problem 8.8 as an *oracle* and to the hyperplane H as the *query hyperplane*.

The method presented here solves instances of Problem 8.3 by running a "simulation" of the algorithm $\mathcal{A}$, where (i) additions and multiplication are replaced by vector operations, and (ii) comparisons are replaced by hyperplane queries. Problem 8.8 is solved by three recursive calls to instances of Problem 8.3 on functions of the form $g' : R_1^{d-1} \to R$. For a point $\boldsymbol{\lambda} \in R_1^d$, define $g_{\boldsymbol{\lambda}} = L_{\mathcal{C}'}$ as the function whose pieces are all the pieces of $g = L_{\mathcal{C}}$ which are active at $\boldsymbol{\lambda} \in R_1^d$ ($\mathcal{C}' = \{\boldsymbol{c} \in \mathcal{C} \mid \boldsymbol{c}^T\boldsymbol{\lambda} = g(\boldsymbol{\lambda})\}$). The functions to which the recursive calls are made are restrictions of functions of the form $g_{\boldsymbol{\lambda}}$ to hyperplanes. We give a method to convert a piecewise affine algorithm $\mathcal{A}$ that computes g to a piecewise affine algorithm $\mathcal{A}'$ that computes g' and performs the same number of operations as $\mathcal{A}$. Algorithms that solve Problem 8.8 for the parametric minimum cycle and the parametric minimum cycle-mean functions are given in following sections.

8.2. Geometric lemmas

We prove some properties of weak approximations and balancing combinations. These properties are used later for solving the "optional" part of Problem 8.3, that is, to solve the second part of Problem 4.7.

For a set $F \subset R^n$ we use the following notation: $\operatorname{aff} F$ is the affine hull of F, that is, the smallest flat which contains F, $\operatorname{lin} F$ is the subspace spanned by F, and $\operatorname{cone} F$ is the convex cone spanned by F (conic hull of F).

Proposition 8.9 *Suppose that $g = L_{\mathcal{C}}$, where $\mathcal{C}$ is a finite set of vectors, $\boldsymbol{\lambda}^* \in \Lambda_g$, and $\mathcal{C}' = \{\boldsymbol{c} \in \mathcal{C} \mid \boldsymbol{c}^T\boldsymbol{\lambda}^* = g(\boldsymbol{\lambda}^*)\}$. The function $L_{\mathcal{C}'}$ is a weak approximation of g.*

Proof: Consider g in a neighborhood of $\boldsymbol{\lambda}^*$. Since C is finite, there exists an open neighborhood N of $\boldsymbol{\lambda}^*$ such that $L_{\mathcal{C}'}(\boldsymbol{\lambda}) = g(\boldsymbol{\lambda})$ for every $\boldsymbol{\lambda} \in N$. Consider a vector $\boldsymbol{\lambda}'$ and denote by L the open line segment determined by $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}^*$. Consider a point $\boldsymbol{\lambda}'' \in L \cap N$, and let $\boldsymbol{c} \in C'$ be a piece of g active at $\boldsymbol{\lambda}''$. Since $g(\boldsymbol{\lambda}'') \le g(\boldsymbol{\lambda}^*)$ we have $g(\boldsymbol{\lambda}') \le g(\boldsymbol{\lambda}^*)$. Suppose that $\boldsymbol{\lambda}' \notin \operatorname{aff}\Lambda$. We need to show that $L_{\mathcal{C}'}(\boldsymbol{\lambda}') < g(\boldsymbol{\lambda}^*)$. For every $\boldsymbol{\lambda} \in L$, $g(\boldsymbol{\lambda}) < g(\boldsymbol{\lambda}^*)$, since otherwise we have $\boldsymbol{\lambda}' \in \operatorname{aff}\Lambda$. In particular the latter holds for $\boldsymbol{\lambda}''$. It follows that $L_{\mathcal{C}'}(\boldsymbol{\lambda}') \le \boldsymbol{c}^T\boldsymbol{\lambda}' < g(\boldsymbol{\lambda}^*)$. ∎

Corollary 8.10 *If $L_{\mathcal{C}'}$ is a minimal weak approximation of g, we have $\boldsymbol{c}^T\boldsymbol{\lambda} = g(\Lambda)$ for all $\boldsymbol{c} \in \mathcal{C}'$ and $\boldsymbol{\lambda} \in \operatorname{aff}\Lambda$. It follows that $\Lambda_{L_{\mathcal{C}'}} = \operatorname{aff}\Lambda$.*

Proposition 8.11 *If $g = L_{\mathcal{C}}$ and for some $\boldsymbol{\lambda} \in \operatorname{rel\,int}\Lambda$ and $\boldsymbol{c} \in \mathcal{C}$ we have $\boldsymbol{c}^T\boldsymbol{\lambda} = g(\Lambda)$, then $\boldsymbol{c}^T\boldsymbol{\lambda} = g(\boldsymbol{\lambda})$ for all $\boldsymbol{\lambda} \in \operatorname{aff}\Lambda$.*

The proof is immediate.

Proposition 8.12 *Suppose $g = L_{\tilde{\mathcal{C}}}$, where $\tilde{\mathcal{C}} \subset R^d$, is bounded from above. Let $\mathcal{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r\} \subset \tilde{\mathcal{C}}$ be a set of pieces of g. Under these conditions,*

i. The function $L_{\mathcal{C}}$ is a weak approximation of g if and only if $\operatorname{cone}\mathcal{C} \supseteq (\operatorname{aff}\Lambda_g)^{\perp}$. If $L_{\mathcal{C}}$ is a minimal weak approximation of g, $\operatorname{cone}\mathcal{C} = (\operatorname{aff}\Lambda_g)^{\perp}$.

ii. If $L_{\mathcal{C}}$ is a minimal weak approximation of g, then there exist positive numbers $\alpha_1, \ldots, \alpha_r$ such that $\sum_{i=1}^r \alpha_i\boldsymbol{c}_i = \boldsymbol{0}$ and hence $\operatorname{lin}\mathcal{C} = (\operatorname{aff}\Lambda_g)^{\perp}$.

iii. If $\alpha_1, \ldots, \alpha_r$ as in (ii) are given explicitly, then every $\boldsymbol{c} \in (\operatorname{aff}\Lambda)^{\perp}$ can be expressed as a nonnegative linear combination of vectors in $\mathcal{C}$ using $O(d^2r)$ operations.

Proof: The function $L_{\hat{\mathcal{C}}}$ is bounded from above on R^d, and hence is nonpositive. Therefore, $\boldsymbol{0} \in \Lambda, L_{\hat{\mathcal{C}}}(\Lambda) = 0$, and $\operatorname{aff}\Lambda = \operatorname{lin}\Lambda$ is a subspace. Part (i) follows from Proposition 4.10. The equality when $L_{\mathcal{C}}$ is a minimal weak approximation follows from Corollary 8.10. Parts (ii)-(iii) are direct consequences of Part (i) and Proposition 4.17. ∎

Remark 8.13 Suppose $H = \{\boldsymbol{\lambda} \in R^d \mid \boldsymbol{a}^T\boldsymbol{\lambda} = \beta\}$ ($\beta \geq 0$) is a hyperplane, $F = \{\boldsymbol{\lambda} \in R^d \mid \boldsymbol{a}^T\boldsymbol{\lambda} = \boldsymbol{0}\}$ is the subspace parallel to H, and $\boldsymbol{\lambda}^* \in H$ is a point on the hyperplane. By applying standard methods, using $O(d^3)$ operations we can find an affine mapping M from H onto R^{d-1} that maps $\boldsymbol{\lambda}^*$ to $\boldsymbol{0}$. All such mappings are of the form $M : H \to R^{d-1}$, $M(\boldsymbol{\lambda}) = \boldsymbol{L}(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*)$ where the matrix $\boldsymbol{L} \in R^{(d-1)\times d}$ is such that $\boldsymbol{L}(F) = \{\boldsymbol{L}\boldsymbol{\lambda} \mid \boldsymbol{\lambda} \in F\} = R^d$. Within the same time bound we can compute the matrices $\boldsymbol{L}^{-1} \in R^{d\times(d-1)}$ which map R^{d-1} onto the subspace F, and $\boldsymbol{F} \in R^{d\times d}$ such that for $\boldsymbol{y} \in R^d$, $\boldsymbol{F}\boldsymbol{y}$ is the projection of $\boldsymbol{y}$ into the subspace F.

Proposition 8.14 *Let $\mathcal{C} \subset R^d$ be a finite set of vectors, and suppose $L_\mathcal{C} : R^d \to R$ is bounded. Suppose we are given a hyperplane $H \subset R^d$, along with a point $\boldsymbol{\lambda}^* \in \Lambda_{L_\mathcal{C}|_H}$*[5]*. Under these conditions, in $O(d^3)$ operations we can compute an affine mapping $M : H \to R^{d-1}$ ($M(\boldsymbol{\lambda}^*) = \boldsymbol{0}$), and a linear mapping of the vectors in $\mathcal{C}' = \{\boldsymbol{c} \in \mathcal{C} \mid \boldsymbol{c}^T\boldsymbol{\lambda}^* = L_C(\Lambda)\}$ (the pieces of $L_\mathcal{C}$ which are "active" at $\boldsymbol{\lambda}^*$) into vectors in $\tilde{\mathcal{C}} \subset R^{d-1}$. These mappings convert the problem of computing a minimal weak approximation of $L_\mathcal{C}$ restricted to H into an equivalent problem of computing a minimal weak approximation of $L_{\tilde{\mathcal{C}}} : R^{d-1} \to R$.*

Proof: We use the notation of Remark 8.13. It follows from Proposition 8.9 that $L_{\mathcal{C}'}$ is a weak approximation of $L_\mathcal{C}|_H$, so we can consider only the vectors in the set $\mathcal{C}'$. Let $\boldsymbol{K} = \boldsymbol{F}\boldsymbol{L}^{-1}$, and $\tilde{\mathcal{C}} = \{\tilde{\boldsymbol{c}} \mid \boldsymbol{c} \in \mathcal{C}'\}$ where $\tilde{\boldsymbol{c}} = \boldsymbol{c}^T\boldsymbol{K}$. We will show that (i) the set $\tilde{\mathcal{C}}$ is such that for all $\boldsymbol{\lambda} \in H, \boldsymbol{c} \in \mathcal{C}'$, $\boldsymbol{c}^T\boldsymbol{\lambda} = \tilde{\boldsymbol{c}}^T M(\boldsymbol{\lambda}) + L_\mathcal{C}(\boldsymbol{\lambda}^*)$, and (ii) for $Y \subset \mathcal{C}$, the set $\tilde{Y} \subset \tilde{\mathcal{C}}$ is such that $L_{\tilde{Y}}$ is a minimal weak approximation of $L_{\tilde{\mathcal{C}}}$ if and only if L_Y is a minimal weak approximation of $L_\mathcal{C}|_H$.

For a vector $\boldsymbol{y} \in R^d$ denote by $\boldsymbol{y}' = \boldsymbol{y}^T\boldsymbol{a}/\|\boldsymbol{a}\|$ the projection of $\boldsymbol{y}$ into the subspace spanned by $\boldsymbol{a}$, and denote by $\hat{\boldsymbol{y}} = \boldsymbol{F}\boldsymbol{y} = \boldsymbol{y} - \boldsymbol{y}'$ the projection of $\boldsymbol{y}$ on F. Observe that $F = \{\beta\boldsymbol{a} \mid \beta \in R\}^\perp \subset R^d$, and thus for all $\boldsymbol{x}, \boldsymbol{y} \in R^d$, $\boldsymbol{x}^T\boldsymbol{y} = \hat{\boldsymbol{x}}^T\hat{\boldsymbol{y}} + \boldsymbol{x}'\boldsymbol{y}'$, and for all $\boldsymbol{\lambda} \in H$ we have $\boldsymbol{\lambda}' = (\boldsymbol{\lambda}^*)'$.

For $\boldsymbol{c} \in \mathcal{C}'$, define $\tilde{c} = \hat{\boldsymbol{c}}^T\boldsymbol{L}^{-1} = \boldsymbol{c}\boldsymbol{K}$. Consider vectors $\boldsymbol{c} \in \mathcal{C}'$ and $\boldsymbol{\lambda} \in H$. Recall that $\boldsymbol{c}^T\boldsymbol{\lambda}^* = \boldsymbol{L}_\mathcal{C}(\boldsymbol{\lambda}^*)$, and therefore we need to show that $\boldsymbol{c}^T(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) = \tilde{\boldsymbol{c}}^T M(\boldsymbol{\lambda})$. Note that $\boldsymbol{c}^T(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) = \hat{\boldsymbol{c}}^T(\hat{\boldsymbol{\lambda}} - \hat{\boldsymbol{\lambda}^*}) + \boldsymbol{c}'(\boldsymbol{\lambda}' - (\boldsymbol{\lambda}^*)')$. Since $\boldsymbol{\lambda}' = (\boldsymbol{\lambda}^*)'$ for all $\boldsymbol{\lambda} \in H$, we have $\boldsymbol{c}^T(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) = \hat{\boldsymbol{c}}^T\boldsymbol{L}^{-1}\boldsymbol{L}(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) = \tilde{\boldsymbol{c}}^T M(\boldsymbol{\lambda})$. ∎

Proposition 8.15 *Under the conditions of Proposition 8.14 the following holds: A set of vectors $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r\} \subset \mathcal{C}'$, and a set of positive numbers $\alpha_1, \ldots, \alpha_r$, $\alpha_i > 0$ satisfy $\sum_{i=1}^r \alpha_i\tilde{\boldsymbol{c}}_i = \boldsymbol{0}$, if and only if $\sum_{i=1}^r \alpha_i\boldsymbol{c}_i = -\gamma\boldsymbol{a}$ and $\gamma \geq 0$. Moreover, $\gamma > 0$ if and only if $L_\mathcal{C}|_H < 0$. The scalars α_i are independent of the particular choice of $\boldsymbol{\lambda}^*$ and the linear mappings.*

Proof: We continue to use the notation of the proof of Proposition 8.14. Recall that $\tilde{\boldsymbol{c}}L = \hat{\boldsymbol{c}}$, and thus $\sum_{i=1}^r \alpha_i\hat{\boldsymbol{c}}_i = (\sum_{i=1}^r \alpha_i\tilde{\boldsymbol{c}}_i)L = \boldsymbol{0}$. Hence, $\sum_{i=1}^r \alpha_i\boldsymbol{c}_i = \sum_{i=1}^r \alpha_i\hat{\boldsymbol{c}}_i + \sum_{i=1}^r \alpha_i\boldsymbol{c}_i'$ $= \sum_{i=1}^r \alpha_i\boldsymbol{c}_i'$. Denote $\boldsymbol{c}_i' = c_i'\boldsymbol{a}$. We need to show that $\sum_{i=1}^r \alpha_i c_i' \leq 0$ and the inequality is strict if and only if $L_\mathcal{C}(\boldsymbol{\lambda}^*) < 0$. Recall that for all $\boldsymbol{c} \in \mathcal{C}'$, $\boldsymbol{c}^T\boldsymbol{\lambda}^* = L_\mathcal{C}(\boldsymbol{\lambda}^*)$. Thus, $(\sum_{i=1}^r \alpha_i\boldsymbol{c}_i')^T\boldsymbol{\lambda}^* = (\sum_{i=1}^r \alpha_i c_i')\boldsymbol{a}^T\boldsymbol{\lambda}^* \leq 0$, and equality holds if and only if $L_\mathcal{C}(\boldsymbol{\lambda}^*) = 0$. To conclude the proof, observe that $\boldsymbol{\lambda}^* \in H$, and therefore $\boldsymbol{a}^T\boldsymbol{\lambda}^* = \beta > 0$. ∎

[5] $g|H$ is the function g whose domain is restricted to H

Proposition 8.16 *Suppose we are given a set of vectors $\mathcal{C} = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_r\} \subset R^d$ such that $\mathcal{C}$ spans R^d and a balancing combination of $\mathcal{C}$ (see Definition 8.1 part ii). By using $O(rd^2)$ operations we can find a minimal subset $\mathcal{C}' \subset \mathcal{C}$ ($|\mathcal{C}'| \le 2d$) that spans R^d and a balancing combination for $\mathcal{C}'$.*

Proof: Assume that the vectors $\boldsymbol{c}_1, \dots, \boldsymbol{c}_d$ span R^d. In order to find a solution we use an iterative step with the following properties. If $|\mathcal{C}| > 2d$, we find a balancing combination for $\mathcal{C} \setminus \tilde{\mathcal{C}}$ where $\tilde{\mathcal{C}} \subset \{\boldsymbol{c}_{d+1}, \dots, \boldsymbol{c}_{2d+1}\}$ and $|\tilde{\mathcal{C}}| \ge 1$. It is easy to verify that after repeating this step at most $r - 2d$ times, we get a balancing combination for a subset $\mathcal{C}'$ of $\mathcal{C}$, which spans R^d and is of size at most $2d$.

The iterative step is as follows. Solve the linear system of equalities $\sum_{i=d+1}^{2d+1} \beta_i \boldsymbol{c}_i = \boldsymbol{0}$. The vectors $\boldsymbol{c}_{d+1}, \dots, \boldsymbol{c}_{2d+1}$ are linearly dependent, and hence this system is feasible. Assume without loss of generality that for at least one index i, $\beta_i > 0$. Let $I_1 = \{d + 1 \le i \le 2d \mid \beta_i > 0\}$, $I_2 = \{d + 1 \le i \le 2d \mid \beta_i < 0\}$, $\gamma = \min_{i \in I_1} \alpha_i/\beta_i$, and $\tilde{\mathcal{C}} = \{\boldsymbol{c}_i \mid i \in I_1, \alpha_i/\beta_i = \gamma\}$. We have $\sum_{i \in I_1 \cup I_2} \beta_i \boldsymbol{c}_i = \boldsymbol{0}$, and $\tilde{\mathcal{C}} \ne \emptyset$. Consider $\boldsymbol{\alpha}' = (\alpha'_1, \dots, \alpha'_r)$ where $\alpha'_i = \alpha_i$ ($i \notin I_1 \cup I_2$), $\alpha'_i = \alpha_i - \gamma\beta_i$ ($i \in I_1 \cup I_2$). It is easy to verify that $\alpha'_i \ge 0$ and $\alpha'_i = 0$ if and only if $\boldsymbol{c}_i \in \tilde{\mathcal{C}}$. It follows that $\sum_{\boldsymbol{c}_i \in \mathcal{C} - \tilde{\mathcal{C}}} \alpha'_i \boldsymbol{c}_i = \sum_{\boldsymbol{c}_i \in \mathcal{C}} \alpha_i \boldsymbol{c}_i - \gamma \sum_{i \in I_1 \cup I_2} \beta_i \boldsymbol{c}_i = \boldsymbol{0}$. Thus, the vectors in $\mathcal{C} \setminus \tilde{\mathcal{C}}$ span R^d, and $\sum_{\boldsymbol{c}_i \in \mathcal{C} - \tilde{\mathcal{C}}} \alpha'_i \boldsymbol{c}_i$ is a balancing combination.

The complexity is as follows. The first step amounts to computing d independent vectors $\{\boldsymbol{c}_1, \dots, \boldsymbol{c}_d\}$, what can be done in $O(rd^2)$ time. In each iterative step we solve a system of linear equalities. This can be done in $O(d^3)$ operations, using Gaussian eliminations. Notice, however, that in two consecutive iterations, the linear systems have $d - |\tilde{\mathcal{C}}|$ columns in common. In such a case, when the matrix of the first problem is given in upper triangular form, the second system can be solved in only $O(|\tilde{\mathcal{C}}|d^2)$ operations. Thus, the total number of operations is $O(rd^2)$. ∎

Corollary 8.17 *Suppose $g = L_Y$ where ($Y \subset R^d$) is bounded from above. Assume we are given a set of vectors $\mathcal{C} = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_r\}$ and a balancing combination of $\mathcal{C}$ such that (i) $L_{\mathcal{C}}$ is a weak approximation of g, and (ii) there exist $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} \Lambda$ such that for all $\boldsymbol{c} \in \mathcal{C}$ we have $\boldsymbol{c}^T\boldsymbol{\lambda}^* = g(\boldsymbol{\lambda}^*)$. Under these conditions, in $O(rd^2)$ operations we can compute $\mathcal{C}' \subset \mathcal{C}$ such that $L_{\mathcal{C}'}$ is a minimal weak approximation of g, and a balancing combination of $\mathcal{C}'$. The size of such $\mathcal{C}'$ is at most $2(d - \dim(\Lambda))$.*

Proof: It follows from Corollary 8.10 and Propositions 8.11 that for all $\boldsymbol{c} \in \mathcal{C}$, $\boldsymbol{c} \in (\operatorname{aff} \Lambda)^\perp$. It follows from Proposition 8.12 that the vectors $\mathcal{C}$ span the subspace $(\operatorname{aff} \Lambda)^\perp$. To conclude the proof, note that the subspace $(\operatorname{aff} \Lambda)^\perp$ is of dimension $d - \dim(\Lambda)$. ∎

Proposition 8.18 *Suppose $g = L_{\mathcal{C}}$ has the property that there exist a vector $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} \Lambda$ such that $\boldsymbol{c}^T\boldsymbol{\lambda}^* = g(\Lambda)$ for all $\boldsymbol{c} \in \mathcal{C}$. Suppose we are given:*

i. Hyperplanes H_δ ($\delta \in \{-1, 0, 1\}$), where $H_\delta = \{\boldsymbol{\lambda} \mid \boldsymbol{a}^T\boldsymbol{\lambda} = \alpha + \delta\}$ for some vector $\boldsymbol{a}$ and $\alpha \in R$, and H_0 contains $\boldsymbol{\lambda}^*$.

ii. *Sets of vectors* $\mathcal{C}_\delta = \{\boldsymbol{c}_1^\delta, \ldots, \boldsymbol{c}_{r_\delta}^\delta\} \subset \mathcal{C}$ *and* $\boldsymbol{\alpha}^\delta \in R_+^{r_\delta}$ $(\delta \in \{-1, 0, 1\})$ *such that* $L_{\mathcal{C}_\delta}$ *is a minimal weak approximation of* $g|H_\delta$, *and* $\sum_{i=1}^{r_\delta} \boldsymbol{\alpha}_i^\delta \boldsymbol{c}_i^\delta$ *is a balancing combination of* $\mathcal{C}_\delta$ *relative to* H_δ $(\delta \in \{-1, 0, 1\})$.

By using $O(d^3 \sum_{\delta \in \{-1,0,1\}} r_\delta)$ *operations we can compute a set of vectors* $\tilde{\mathcal{C}}$ *and a balancing combination of* $\tilde{\mathcal{C}}$ *relative to* R_1^d, *such that* $\tilde{\mathcal{C}} \subset \bigcup_{\delta \in \{-1,0,1\}} \mathcal{C}_\delta$ *and* $L_{\tilde{\mathcal{C}}}$ *is a minimal weak approximation of* g.

Proof: We give a description of an algorithm. In the first step the algorithm computes a set $\mathcal{C}' \subset \mathcal{C}$ of size at most $\sum_{\delta \in \{-1,0,1\}} r_\delta$ and a balancing combination of $\mathcal{C}'$, such that $L_{\mathcal{C}'}$ is a weak approximation of g. This is done as follows.

i. If $\Lambda \subset H_0$, then $\mathcal{C}' = \bigcup_{\delta \in \{0,1,-1\}} \mathcal{C}_\delta$ is such that $L_{\mathcal{C}'}$ is a weak approximation of g. To find a balancing combination, the algorithm computes $\beta_\delta > 0$ $(\delta \in \{-1, 0, 1\})$ such that $\sum_{\delta \in \{-1,0,1\}} \beta_\delta \sum_{i=1}^{r_\delta} \alpha_i^\delta \boldsymbol{c}_i^\delta = \mathbf{0}$. Otherwise,

ii. $H_\delta \cap \Lambda \neq \emptyset$ (for either $\delta = 1$ or $\delta = -1$). It turns out that the set $\mathcal{C}' = \mathcal{C}_\delta$ is such that $L_{\mathcal{C}'}$ is a weak approximation of g. Hence, the algorithm chooses $\boldsymbol{\alpha}^\delta$ as the coefficients of a balancing combination.

Assuming that $\mathcal{C}'$ is a weak approximation of g the algorithm proceeds as follows. In case ii, $\tilde{\mathcal{C}} = \mathcal{C}'$ is obviously a minimal weak approximation of g. In case i it follows from Corollary 8.17 that by using $O(d^3 \sum_{\delta \in \{-1,0,1\}} r_\delta)$ operations we can compute $\tilde{\mathcal{C}} \subset \mathcal{C}$ and a balancing combination for $\tilde{\mathcal{C}}$, such that $L_{\tilde{\mathcal{C}}}$ is a minimal weak approximation of g.

What remains is to prove that $L_{\mathcal{C}'}$ is indeed a weak approximation of g. Denote by $\Lambda_\delta \subset H_\delta$ the set of maximizers of g restricted to H_δ. By using an appropriate translation of the coordinates, we can transform the problem so that $\mathcal{C} \subset R^{d-1}$, $g : R^{d-1} \to R$, $H_\delta = \{\boldsymbol{\lambda} \in R^{d-1} \mid \boldsymbol{a}^T \boldsymbol{\lambda} = \delta\}$, $\boldsymbol{\lambda}^* = \mathbf{0}$, and $g(\Lambda) = 0$. This transformation preserves the property of a subset being a weak approximation or having a balancing combination relative to a given flat. A balancing combination in the transformed problem corresponds to a balancing combination relative to R_1^d in the original one. Note that in the transformed problem, $\mathbf{0} \in \Lambda$, so the flat $\operatorname{aff} \Lambda$ is a subspace.

Let $\boldsymbol{\lambda}_\delta^* \in \operatorname{rel\,int} \Lambda_\delta$. Observe that g is linear, that is, for all $\alpha > 0$, $g(\alpha \boldsymbol{\lambda}) = \alpha g(\boldsymbol{\lambda})$. Thus, in case i $\Lambda = \Lambda_0$. In case ii consider the intersection of Λ with the open halfspace $\{\boldsymbol{\lambda} \mid \delta(\boldsymbol{a}^T \boldsymbol{\lambda}) > 0\}$. The set Λ is convex. Thus, if is not empty, it must contain a relative interior point of Λ. The hyperplane H_δ is contained in this halfspace and hence this intersection is not empty, and $\dim \Lambda_\delta = \dim \Lambda - 1$. Note that $\mathbf{0} \notin \operatorname{aff} \Lambda_\delta$. Thus, the flat $\operatorname{aff} \Lambda_\delta \oplus \{\beta \boldsymbol{\lambda}_\delta^* \mid \beta \in R\}$ must be contained in $\operatorname{aff} \Lambda$. On the other hand this flat is of the same dimension as $\operatorname{aff} \Lambda$ and therefore equality holds. Consider any $\boldsymbol{\lambda}' \in R^{d-1}$ such that $\boldsymbol{\lambda}' \notin \operatorname{aff} \Lambda$. In order to prove that $L_{\mathcal{C}'}$ weakly approximates g we need to show that there exists a $\boldsymbol{c} \in \mathcal{C}'$ such that $\boldsymbol{c}^T \boldsymbol{\lambda}' < 0$.

First, we prove case i. If $\boldsymbol{a}^T \boldsymbol{\lambda}' = 0$ then we are done, since $\boldsymbol{\lambda} \in H_0$ and $\mathcal{C}_0 \subset \mathcal{C}'$. Otherwise, assume without loss of generality that $\boldsymbol{a}^T \boldsymbol{\lambda}' = \beta > 0$. Consider the vector $\tilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}/\beta$. The vector $\tilde{\boldsymbol\lambda}$ lies on the line determined by $\mathbf{0}$ and $\boldsymbol\lambda$, and is such that $\boldsymbol{a}^T\tilde{\boldsymbol\lambda} = 1$. Since $\operatorname{aff}\Lambda$ is a flat, if it contains two points on a line, it contains the whole line. Therefore, $\mathbf{0} \in \operatorname{aff}\Lambda$ and $\boldsymbol\lambda' \notin \operatorname{aff}\Lambda$ imply that $\tilde{\boldsymbol\lambda} \notin \operatorname{aff}\Lambda$. It follows that there exists a $\boldsymbol{c} \in \mathcal{C}_1$ such that $\tilde{\boldsymbol\lambda}^T\boldsymbol{c} < 0$. Finally, observe that $\boldsymbol\lambda'^T\boldsymbol{c} = (\beta\tilde{\boldsymbol\lambda})^T\boldsymbol{c} < 0$.

To prove case ii, define $\boldsymbol\lambda''$ to be the vector $\boldsymbol\lambda'' = \boldsymbol\lambda' + \beta\boldsymbol\lambda^*_\delta$ where $\beta = (\delta - \boldsymbol{a}^T\boldsymbol\lambda')/(\boldsymbol{a}^T\boldsymbol\lambda^*_\delta)$. Note that $\boldsymbol{a}^T\boldsymbol\lambda'' = \delta$ and thus $\boldsymbol\lambda'' \in H_\delta$. Moreover, $\boldsymbol\lambda'' \notin \operatorname{aff}\Lambda_\delta$, since otherwise $\boldsymbol\lambda' \in \operatorname{aff}\Lambda$. Thus, there exists a $\boldsymbol{c} \in \mathcal{C}_\delta$ such that $\boldsymbol{c}^T\boldsymbol\lambda'' < 0$. We have $\boldsymbol\lambda^*_\delta \in \operatorname{aff}\Lambda$ and therefore $\boldsymbol\lambda''$ has the same projection on $(\operatorname{aff}\Lambda)^\perp$ as $\boldsymbol\lambda'$. It follows from Proposition 8.11 that for all $\boldsymbol{c}' \in \mathcal{C}_\delta$, $\boldsymbol{c}' \in (\operatorname{aff}\Lambda)^\perp$. Hence, $\boldsymbol{c}^T\boldsymbol\lambda' = \boldsymbol{c}^T\boldsymbol\lambda'' < 0$

To find a balancing combination, recall from Proposition 8.15 that there exist numbers $\gamma_1 > 0, \gamma_{-1} < 0$ such that $\sum_{i=1}^{r_\delta}\alpha_i^\delta\boldsymbol{c}_i^\delta = \gamma_\delta\boldsymbol{a}$. Hence, we choose $\beta_\delta = |\gamma_{1-\delta}|$, $\beta_0 = 1$. It is easy to verify that

$$\sum_{\delta\in\{-1,0,1\}}\beta_\delta\sum_{i=1}^{r_\delta}\alpha_i^\delta\boldsymbol{c}_i^\delta = \mathbf{0}\ .$$

∎

8.3. Back to zero-cycles

We are now ready to give the deferred proof from Section 4.

Proof of Proposition 4.9: Observe that the weight of a minimum weight cycle in a graph with the scalar weights $\alpha\boldsymbol\lambda^T\boldsymbol{f}$ $(\alpha > 0)$ is linear in α. If the graph G has a witness $\boldsymbol\lambda$ then either $\lambda_d = 0$ or the vector $(1/\lambda_d)\boldsymbol\lambda$ is a witness as well. Therefore, we can restrict our search on the $\boldsymbol\lambda$-space to vectors $\boldsymbol\lambda \in R^d$ where $\lambda_d \in \{-1, 0, 1\}$. Denote

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

Run the parametric minimum cycle (resp., parametric minimum cycle-mean) algorithm on G as defined in Problem 8.7 (resp., Problem 8.5) three times with the following $(d-1)$-variable linear functions. For $\delta = -1, 0, 1$, we associate with the edges the linear functions $\boldsymbol{f}^T\boldsymbol\lambda_\delta$ where $\boldsymbol\lambda_\delta = (\lambda_1, \ldots, \lambda_{d-1}, \delta)^T$. If a witness for G exists, it must be found during at least one of the three runs. Otherwise, the maximum of the function g in each of the three subproblems is nonpositive. Thus, for all three instances, the corresponding sets $\mathcal{K}_\delta$ $(\delta \in \{-1,0,1\})$ consist of vectors for which $g = \mathbf{0}$. The algorithm computes nonzero vectors $\boldsymbol\lambda^*_\delta \in \operatorname{rel\,int}\mathcal{K}_\delta$, $\delta \in \{-1,0,1\}$ (if $\mathcal{K}_\delta \neq \mathbf{0}$), along with collections $\mathcal{C}^{(\delta)}$ (of size $|\mathcal{C}^{(\delta)}| = O(d)$) of cycles such that $L_{\{\boldsymbol{f}(C)\ |\ C\in\mathcal{C}^{(\delta)}\}}$ is a minimal weak approximation of g restricted to the hyperplane $\lambda_d = \delta$, and $\boldsymbol\alpha^\delta$ are coefficients of balancing combinations of $\mathcal{C}^\delta$ relative to $\lambda_d = \delta$. To find the separating vector $\boldsymbol\lambda$ we proceed as follows. (i) If $\mathcal{K}_\delta \neq \emptyset$ for $\delta \in \{-1, 1\}$, we choose $\boldsymbol\lambda$ to be $\boldsymbol\lambda^*_\delta$, and then

$$\boldsymbol\lambda = \boldsymbol\lambda^*_\delta \in \operatorname{rel\,int}\mathcal{K}_\delta \subset \operatorname{rel\,int}\mathcal{K}\ .$$

(ii) If $\mathcal{K}_{-1} = \mathcal{K}_1 = \emptyset$ and $\mathcal{K}_0 \neq \{\mathbf{0}\}$, we choose $\boldsymbol\lambda$ to be a nonzero vector $\boldsymbol\lambda^*_0 \in \operatorname{rel\,int}\mathcal{K}_0$, and so

$$\boldsymbol\lambda = \boldsymbol\lambda^*_0 \in \operatorname{rel\,int}\mathcal{K}_0 = \operatorname{rel\,int}\mathcal{K}\ .$$

(iii) The remaining case is $\mathcal{K}_{-1} = \mathcal{K}_1 = \emptyset$ and $\mathcal{K}_0 = \{\mathbf{0}\}$. Here we conclude that $\mathcal{K} = \{\mathbf{0}\}$, so there is no separating vector.

As done in the proof of Proposition 8.18, using $O(d^3)$ operations, we can find a set $\mathcal{C}$ of cycles and a balancing combination for the cycle values $\{\boldsymbol{f}(C) \mid C \in \mathcal{C}\}$, such that the lower envelope of the set of cycle values is a minimal weak approximation of the function g. To conclude the proof observe that the size of a minimal weak approximation is at most $O(2d)$ (see Corollary 8.17), and that $\operatorname{cone}\mathcal{C} \supseteq \mathrm{ORTH}(G)$ (see Corollary 4.11). ∎

9. Algorithm for parametric minimum cycle-mean

In this subsection we sketch a strongly polynomial time algorithm for the parametric minimum cycle-mean problem. As mentioned before, this algorithm is simpler than the parametric minimum cycle algorithm. However, its time bounds are worse. The purpose of discussing it here is to give the reader some intuition and explain some of the main ideas behind the strongly polynomial bounds. Thus, only the ideas that are essential for strongly polynomial time bounds are introduced. Two simplifications are made. First, the multi-dimensional search technique is not discussed. It improves the time complexity, but is not essential for strongly polynomial bounds. The second simplification is that the algorithm only decides the existence of a zero-cycle. Therefore, the collection $\mathcal{C}$ (see Problem 4.7) need not be computed, and it suffices to be able to compute a witness or a separating vector.

Remark 9.1 The minimum cycle-mean relative to a set of scalar weights can be found in $O(|E|\cdot|V|)$ time by an algorithm due to Karp [16]. With n^3 parallel processors, the minimum cycle-mean can be computed in $O(\log^5 n)$ time (see [21]).

Proposition 9.2 *The value of $g(\boldsymbol{\lambda})$ can be described as*

$$g(\boldsymbol{\lambda}) = \max_{\pi_1,\ldots,\pi_n} \min_{i,j} \left(-\pi_i + \pi_j - \boldsymbol{\lambda}^T \boldsymbol{c}_{ij}\right) .$$

Proof: Given $\boldsymbol{\lambda} \in R_1^d$, consider the following linear programming problem:

$$(P) \qquad \begin{aligned} \text{Minimize} \quad & \sum_{i,j} (\boldsymbol{\lambda}^T \boldsymbol{c}_{ij}) x_{ij} \\ \text{subject to} \quad & \sum_j (x_{ij} - x_{ji}) = 0 \qquad (i = 1, \ldots, n) \\ & \sum_{i,j} x_{ij} = 1 \\ & \boldsymbol{x} \geq \mathbf{0} \ . \end{aligned}$$

Obviously, (P) has an optimal solution. Now, every feasible solution $\boldsymbol{x}$ of (P) is a convex combination of feasible solutions $\boldsymbol{x}'$, where $E(\boldsymbol{x}')$ is a simple cycle. Since $\sum x'_{ij} = 1$, the value of the objective function at each $\boldsymbol{x}'$ is precisely the mean weight of the cycle determined by $\boldsymbol{x}'$. Hence (P) has an optimal solution of the form $\boldsymbol{x}'$. In other words, the optimal value of (P) is equal to $g(\boldsymbol{\lambda})$. Consider the dual of (P):

$$(D) \qquad \begin{array}{rl} \displaystyle\operatorname*{Maximize}_{\pi,t} & t \\ \text{subject to} & \pi_i - \pi_j + t \le -\boldsymbol{\lambda}^T \boldsymbol{c}_{ij} \; . \end{array}$$

Our claim follows from the fact that the optimal value of (D) is equal to $g(\boldsymbol{\lambda})$. ∎

Corollary 9.3 *The problem of maximizing $g(\boldsymbol{\lambda})$ can be formulated as a linear programming problem:*

$$(\tilde{P}) \qquad \begin{array}{rl} \displaystyle\operatorname*{Maximize}_{\pi,t,\lambda} & t \\ \text{subject to} & \pi_i - \pi_j + \boldsymbol{\lambda}^T \boldsymbol{c}_{ij} + t \le 0 \; . \end{array}$$

The dual of $(\tilde{P})$ is the following zero-circulation problem:

$$(\tilde{D}) \qquad \begin{array}{rll} \text{Minimize} & \displaystyle\sum_{i,j} d_{ij} x_{ij} & \\ \text{subject to} & \displaystyle\sum_{j} (x_{ij} - x_{ji}) = 0 & (i = 1, \ldots, n) \\ & \displaystyle\sum_{i,j} x_{ij} = 1 & \\ & \displaystyle\sum_{i,j} x_{ij} \boldsymbol{c}'_{ij} = \boldsymbol{0} & \\ & \boldsymbol{x} \ge \boldsymbol{0} \; . & \end{array}$$

Where $\boldsymbol{c}_{ij} = (\boldsymbol{c}'_{ij}, d_{ij})$, $\boldsymbol{c}'_{ij} \in Q^{d-1}$.

Remark 9.4 In view of Corollary 9.3, the problem of maximizing $g(\boldsymbol{\lambda})$ can be solved by an extension of the algorithm for linear programming in fixed dimension [23] and its improvements [1, 8]. Note that only the dimension of the $\boldsymbol{\lambda}$-space is fixed, whereas the number of π_i's varies. However, we will work recursively on the space of $\boldsymbol{\lambda}$'s where at the base of the recursion $(d = 1)$ we have a problem of the form of (D), i.e., a problem with fixed scalar-weights which is equivalent to the usual minimum cycle-mean problem (with no parameters).

Before discussing the algorithm for Problem 8.5 we present the respective "oracle" algorithm.

Problem 9.5 Given $G = (V, E, \boldsymbol{f})$ and a hyperplane H in the $\boldsymbol{\lambda}$-space, solve Problem 8.8 for g relative to H.

Problem 9.5 (an oracle call) is solved by recursive calls to an algorithm for Problem 8.5 on input graphs with vector weights of a lower dimension as we argue below.

Theorem 9.6 *Problem 9.5 can be solved by three recursive calls to an algorithm for Problem 8.5 in dimension $d-1$. The complexity of the additional computation is dominated by the calls to Problem 8.5.*

Proof: Consider Problem 8.5 subject to $\boldsymbol{a}^T\boldsymbol{\lambda} = \alpha$ (and $\boldsymbol{\lambda} \in R_1^d$). This is in fact a $(d-1)$-dimensional version of the original problem restricted to $H = \{\boldsymbol{\lambda} \in R_1^d | \boldsymbol{a}^T\boldsymbol{\lambda} = \alpha\}$. 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 $\boldsymbol{\lambda} \in H$. Suppose we also have corresponding values $\pi_1^0, \ldots, \pi_n^0$ and $t^{(0)} = g(\boldsymbol{\lambda}^{(0)})$. 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^d). If not, we wish to decide whether for all $\boldsymbol{\lambda}^* \in R_1^d$ such that $g(\boldsymbol{\lambda}^*) \geq g(\boldsymbol{\lambda}^{(0)})$, 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. Let E^0 denote the set of edges (i,j) such that

$$-\pi_i^0 + \pi_j^0 - (\boldsymbol{\lambda}^0)^T\boldsymbol{c}_{ij} = t^0 \ .$$

Notice that the subgraph G^0 of G induced by E^0 contains all the cycles of G whose cycle-mean is minimum relatively to the weights $(\boldsymbol{\lambda}^0)^T\boldsymbol{c}_{ij}$, and only such cycles. In order to determine whether $\boldsymbol{\lambda}^0$ is in the relative interior of the global maxima (and if not determine which side of H contains it), we can consider the behavior of g on two hyperplanes close to H. These hyperplanes are parallel to H, one on each side. We consider the local maxima relative to these planes. In order to avoid the problem of how close to get, we can consider just the pieces of g which are active at $\boldsymbol{\lambda}^0$. This is equivalent to considering only cycles whose cycle-mean is minimum relative to $(\boldsymbol{\lambda}^0)^T\boldsymbol{c}_{ij}$, namely, the cycles of the subgraph G^0. We now solve two problems on G^0 of maximizing t subject to

$$\pi_i - \pi_j + \boldsymbol{\lambda}^T\boldsymbol{c}_{ij} + t \leq 0 \qquad ((i,j) \in E^0) \ ,$$

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 $(d-1)$-dimensional problems since one of the λ_i's can be eliminated. Denote the optimal values of these problems by $t^{(1)}$ and $t^{(-1)}$. 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 $\operatorname{rel\,int} \Lambda$ is determined. Otherwise, if either both are less than, or both are equal to $t^{(0)}$, then $t^{(0)}$ is the global optimal value. The base of the recursion is the 1-dimensional problem (with no parameters at all). ∎

A sketch of the algorithm for Problem 8.5 is given below.

Algorithm 9.7 Consider a minimum cycle-mean algorithm that uses only comparisons, additions, and divisions (multiplications) by scalars as primitive operations. Define the corresponding *lifted* algorithm to operate on edge weights that are linear functions, defined by $\boldsymbol{f}(e)$ $(e \in E)$. The lifted algorithm maintains a collection $\mathcal{H}$ of open halfspaces which is initialized as the empty set. Additions and divisions by scalars are naturally "lifted" to additions and scalar divisions of linear functions. Comparisons are more intricate. To "compare" two linear functions, we compute the hyperplane H such that the result of a comparison is uniform throughout each of the halfspaces defines by H. We then perform an oracle call on H. If the oracle call did not result in a global solution, a halfspace h is found (one of the sides of H) which contains the maxima of g. The result of the comparison is the relation between the linear functions that holds throughout h. The halfspace h is then added to $\mathcal{H}$ and the algorithm continues. If the algorithm terminates and no oracle call resulted in a solution, a separating vector (or a witness) is found by considering a point in the intersection of all the open halfspaces in $\mathcal{H}$.

10. Algorithm for parametric minimum cycle

This section introduces an algorithm for the parametric minimum cycle problem. This problem is essentially an instance of Problem 8.3 for a particular family of functions. The general lines of this algorithm can be used to solve Problem 8.3 for any concave function g given by a piecewise affine algorithm.

We suggest an oracle algorithm which relies on making recursive calls to instances of Problem 8.7, where there are fewer parameters, but the weights are more complex.

In order to facilitate the recursion, the problem is generalized to the *extended* parametric minimum cycle problem (EPMC). The algorithm presented here solves the EPMC problem. Let us start with a definition of this problem.

Extended 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 $e \in E$, $\boldsymbol{f}(e) \in R^d$ and $\boldsymbol{w}(e) \in R^\ell$. We identify $\boldsymbol{f}(e)$ with the $(d-1)$-dimensional linear function:

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

The *weight* of an edge e is the $(\ell+1)$-tuple $(w_1(e), \ldots, w_\ell(e), \boldsymbol{f}(e))$. The definitions given in 8.6 are extended as follows.

Definition 10.1 Consider a graph $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$, where for $e \in E$, $\boldsymbol{f}(e) = \boldsymbol{c}_e \in R^d$.

i. 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.

ii. Let $C = C(\boldsymbol{\lambda})$ denote a $\boldsymbol{w}$-minimal cycle of at most n edges which minimizes the weight $\sum_{e \in C} \boldsymbol{\lambda}^T \boldsymbol{c}_e$. Denote $g(\boldsymbol{\lambda}) = \boldsymbol{f}(C)^T \boldsymbol{\lambda}$. 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 $\boldsymbol{\lambda}$; $g(\boldsymbol{\lambda})$ gives only the $(\ell+1)$st coordinate.

Remark 10.2 For a graph with vector weights in R^ℓ, the minimum, relative to the lexicographic order, among cycles of length less than or equal to n can be computed in one application of an all-pairs shortest path algorithm. This takes $O(\ell |E| \cdot |V|)$ time sequentially, or $O(\log^2 n)$ time using ℓn^3 processors in parallel. Therefore, the function $g(\boldsymbol{\lambda})$ can be evaluated by a piecewise affine algorithm (see Definition 8.2). Note that $g(\boldsymbol{\lambda}) = L_{\tilde{\mathcal{C}}}$, where $\tilde{\mathcal{C}} = \{\boldsymbol{f}(C) | C \text{ is a } w\text{-minimal cycle}\}$.

Problem 10.3 [Extended Parametric Minimum Cycle]
Given is $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$ as above. If $g(\boldsymbol{\lambda}) > 0$ for some $\boldsymbol{\lambda}$, then output any such $\boldsymbol{\lambda}$; otherwise, find $\boldsymbol{\lambda} \in \operatorname{rel\,int} \Lambda$ and a collection $\mathcal{C} = \{C_1, \ldots, C_r\}$ $(r \le 2d)$ of $\boldsymbol{w}$-minimal cycles, where each consists of at most n edges, such that the lower envelope $L_{\{\boldsymbol{f}(C)|C\in\mathcal{C}\}}$ of their cycle values is a minimal weak approximation of g, and find a balancing combination of the cycle values $\boldsymbol{f}(C_1), \ldots, \boldsymbol{f}(C_r)$ relative to R^d_1.

The EPMC algorithm presented below performs "oracle" calls. The oracle algorithm recursively solves instances of the EPMC problem on G with sets of vector weights $\boldsymbol{w}' : E \to R^{\ell+1}$ and $\boldsymbol{f}' : E \to R^{d-1}$. The dimension of the $\boldsymbol{f}$ weights, which corresponds to the number of parameters, decreases in the recursive calls. In order to solve an instance of the parametric minimum cycle problem (Problem 8.7), we start with an instance of EPMC where $\boldsymbol{f}$ gives the vector edge weights and $\boldsymbol{w}$ is a set of null vectors. At the base of the recursion the weights $\boldsymbol{f}$ are null vectors, and the problem is reduced to the non-parametric problem of computing minimum cycle relative to the lexicographic order on d-tuples.

In Subsection 10.3 we propose Algorithm 10.14 for Problem 10.3. The algorithm executes calls to the oracle problem (Problem 8.8) relative to g. An algorithm for the oracle problem is given in Subsection 10.1. A call to the oracle is a costly operation. Therefore, one wishes to solve many hyperplane queries with a small number of oracle calls. In Subsection 10.2 we discuss the multi-dimensional search technique (introduced in [23]). By applying it, we are able to reduce the number of oracle calls performed to a polylog in the number of hyperplane queries.

10.1. Hyperplane queries

For a given a hyperplane H of R^d_1, we wish to solve Problem 8.8 for g relative to H. If $H \cap \operatorname{rel\,int} \Lambda \neq \emptyset$, we solve Problem 10.3, that is, we find $\boldsymbol{\lambda} \in \operatorname{rel\,int} \Lambda$, the collection $\mathcal{C}$, and a balancing combination of $\mathcal{C}$.

Problem 10.4 Given is $G = (V, E, \boldsymbol{w}, \boldsymbol{f})$ and a hyperplane $H = \{\boldsymbol{\lambda} \in R^d_1 \mid \boldsymbol{a\lambda} = \alpha\}$ in the $\boldsymbol{\lambda}$-space. Solve Problem 8.8 for g relative to H.

Theorem 10.5 *Problem 10.4 can be solved by an algorithm which performs three calls to instances of Problem 10.3 where $\boldsymbol{f}$ is $(d-1)$-dimensional. The time complexity of the additional computation is dominated by these calls: It can be done sequentially in $C|E| + D$ time for some constants $C = O(d)$ and $D = O(d^3)$. In parallel, it can be done in constant $B = O(d^3)$ time using $O(m + n^3)$ processors.*

Proof: Consider Problem 10.3 subject to $\boldsymbol{a}^T\boldsymbol{\lambda} = \alpha$ and $\boldsymbol{\lambda} \in R^d_1$. This is in fact a $(d-1)$-dimensional version of the original problem restricted to the hyperplane $H = \{\boldsymbol{\lambda} \in R^d_1 \mid \boldsymbol{a}^T\boldsymbol{\lambda} = \alpha\}$. 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 $\mathcal{C}^{(0)}$ and a balancing combination of $\mathcal{C}^{(0)}$ relative to H. 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^d). If not, then we wish to decide whether for all $\boldsymbol{\lambda}^* \in R_1^d$ such that $g(\boldsymbol{\lambda}^*) \geq g(\boldsymbol{\lambda}^{(0)})$, 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 $\boldsymbol{w}' = (\boldsymbol{w}, \boldsymbol{f}^T\boldsymbol{\lambda}^{(0)})$. Note that all the minimum cycles of G' correspond to minimum cycles with value $t^{(0)}$ at $\boldsymbol{\lambda}$ of G. We solve Problem 10.3 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 $(d-1)$-dimensional problems since one of the λ_i's can be eliminated. Denote the optimal values of these problems by $t^{(\delta)}$, the corresponding collections of cycles by $\mathcal{C}^\delta$, and let $\boldsymbol{\alpha}^\delta$ be coefficients of a balancing combination of $\mathcal{C}^\delta$ ($\delta \in \{-1, 1\}$). 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 $\operatorname{rel\,int} \Lambda$ is determined. Otherwise, if either both are less than, or both are equal to $t^{(0)}$, then $t^{(0)}$ is the global optimal value. In the latter case $\boldsymbol{\lambda}^{(0)} \in \operatorname{rel\,int} \Lambda$. It follows from Proposition 8.9 that the pieces of g which are active in a minimal weak approximation have the value $t^{(0)}$ at $\boldsymbol{\lambda}^{(0)}$. Thus, a minimal weak approximation of the function g' (see Definition 10.1) which corresponds to G' is a minimal weak approximation of the "original" g which corresponds to the graph G. The conditions of Proposition 8.18 and Corollary 8.17 hold for the function g'. Hence, in $O(d^3)$ operations we can construct a minimal weak approximation of g, and find a corresponding balancing combination. The base of the recursion is the one-dimensional problem (with no parameters at all) where C consists of a single cycle with minimal value, and the balancing combination is the value of this cycle. ∎

10.2. Employing multi-dimensional search

The multi-dimensional search problem was defined and used in [23] for solving linear programming problems in fixed dimension.

Problem 10.6 [multi-dimensional search]
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. For m given 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 [23]:

Theorem 10.7 *The solution of Problem 10.6 relative to some $m/2$ of the given hyperplanes can be found by using $\gamma \equiv \gamma(d)$ oracle calls. The additional computation can be done sequentially in $O(\gamma(d)m)$ time, and on m parallel processors in $O(\gamma(d)\log m)$ time. From [1, 8] $\gamma(d) = O((5/9)^d 3^{d^2})$.*

Corollary 10.8 *Problem 10.6 can be solved by using $O(\gamma(d)\log m)$ oracle calls and $O(\gamma(d)m)$ additional time (see [23]).*

Remark 10.9 The procedure described in [23] can be parallelized so that the additional computation (besides the $O(\gamma(d)\log m)$ oracle calls) is done by m parallel processors in $O(\gamma(d)\log^2 m)$ time.

Definition 10.10 We define a partial order on $R^d \setminus \{\mathbf{0}\}$ as follows. For any pair of distinct vectors $\boldsymbol{a}_1, \boldsymbol{a}_2 \in R^d$, denote

$$H = H(\boldsymbol{a}_1, \boldsymbol{a}_2) = \{\boldsymbol{\lambda} \in R^d_1 : \boldsymbol{a}_1^T\boldsymbol{\lambda} = \boldsymbol{a}_2^T\boldsymbol{\lambda}\} \ .$$

If g is unbounded on $H(\boldsymbol{a}_1, \boldsymbol{a}_2)$ or if $H(\boldsymbol{a}_1, \boldsymbol{a}_2) \cap \operatorname{rel\,int}\Lambda \neq \emptyset$, then we write $\boldsymbol{a}_1 <>_\Lambda \boldsymbol{a}_2$. Otherwise, g can be unbounded on at most one of the open subspaces determined by H, and also $\operatorname{rel\,int}\Lambda$ can intersect at most one of these open halfspaces. We denote $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$ (respectively, $\boldsymbol{a}_1 >_\Lambda \boldsymbol{a}_2$) if there exists a $\boldsymbol{\lambda} \in \operatorname{rel\,int}\Lambda$ such that $\boldsymbol{a}_1^T\boldsymbol{\lambda} < \boldsymbol{a}_2^T\boldsymbol{\lambda}$ (respectively, $\boldsymbol{a}_1^T\boldsymbol{\lambda} > \boldsymbol{a}_2^T\boldsymbol{\lambda}$), in which case the same holds for all these $\boldsymbol{\lambda}$'s, or if g is unbounded on the halfspace determined by the inequality $\boldsymbol{a}_1^T\boldsymbol{\lambda} < \boldsymbol{a}_2^T\boldsymbol{\lambda}$ (respectively, $\boldsymbol{a}_1^T\boldsymbol{\lambda} < \boldsymbol{a}_2^T\boldsymbol{\lambda}$). We also use the notation $<_P$ for a similar partial order relative to any set P.

Problem 10.11 Given are finite sets $A_1, \ldots, A_r$ of nonzero vectors, where $A_i = \{\boldsymbol{a}^i_1, \ldots, \boldsymbol{a}^i_{s_i}\}$ ($\boldsymbol{a}^i_j \in R^d$) and $s = \sum s_i$. 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, we need to find either one of the following:

i. A collection of closed halfspaces whose intersection P contains $\operatorname{rel\,int}\Lambda$, and indices $1 \le m_i \le s_i$ ($i = 1, \ldots, r$) such that for every $1 \le i \le r$ and every $1 \le j \le s_i$, $j \neq m_i$, we have $\boldsymbol{a}^i_{m_i} <_\Lambda \boldsymbol{a}^i_j$ and $\boldsymbol{a}^i_{m_i} <_P \boldsymbol{a}^i_j$.

ii. A hyperplane H such that either g is unbounded on H or $H \cap \operatorname{rel\,int}\Lambda \neq \emptyset$.

Proposition 10.12 *Problem 10.11 can be solved using $O(\gamma(d-1)\log s)$ oracle calls plus either*

i. $O(\gamma(d-1)\log^2 s)$ parallel time on $O(s)$ processors, or

ii. $O(\gamma(d-1)s\log s)$ sequential time.

(Where $\gamma(d-1)$ is as in Theorem 10.7.)

Proof: The underlying algorithm is an extension of the multi-dimensional search procedure for Problem 10.6 mentioned in Theorem 10.7. Here the set X is either $\operatorname{rel\,int}\Lambda$ 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 $\operatorname{rel\,int}\Lambda$; the case where X is contained in one of the open halfspaces corresponds to the case where either $\operatorname{rel\,int}\Lambda$ is contained in the halfspace, or g is unbounded on the halfspace (but bounded on the hyperplane). Also, note that the flat R_1^d on which the multi-dimensional search is done, is of dimension $d-1$.

We first explain how to recognize for a given pair of distinct vectors $\boldsymbol{a}_i, \boldsymbol{a}_j$ whether $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$, $\boldsymbol{a}_2 <_\Lambda \boldsymbol{a}_1$ or $\boldsymbol{a}_1 <>_\Lambda \boldsymbol{a}_2$. Consider the hyperplane $H = H(\boldsymbol{a}_i, \boldsymbol{a}_j) \subset R_1^d$ (see Definition 10.10). Suppose H is the query hyperplane presented to an oracle which recognizes the location of the set $\operatorname{rel\,int}\Lambda$ relative to H (in the sense of Problem 10.4). In particular, if $H \cap \operatorname{rel\,int}\Lambda \neq \emptyset$, then the oracle discovers this fact and returns a $\boldsymbol{\lambda} \in H \cap \operatorname{rel\,int}\Lambda$. Similarly, if g is unbounded on H, then the oracle reports this fact and provides a $\boldsymbol{\lambda}$ such that $g(\boldsymbol{\lambda}) > 0$. In the remaining cases the oracle reports either $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$ or $\boldsymbol{a}_2 <_\Lambda \boldsymbol{a}_1$.

The multi-dimensional search algorithm computes, adaptively, $O(\gamma(d-1)\log s)$ hyperplane queries. If any of these hyperplanes either intersects $\operatorname{rel\,int}\Lambda$ 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 $\operatorname{rel\,int}\Lambda$ is nonempty and contained in the interior of P. Moreover, vectors $\boldsymbol{a}_{m_1}, \ldots, \boldsymbol{a}_{m_r}$ are found, such that for every i, $\boldsymbol{a}^i_{m_i} <_P \boldsymbol{a}^i_j$, for all $1 \le j \le s_i, j \neq m_i$.

We implement the algorithm as follows. View the dimension d as fixed. It was shown in [1] and [8] that by using 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 apply $O(\log s)$ phases. First, for each i $(1 \le i \le r)$ 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 10.10). In a constant number $\gamma(d-1)$ of oracle calls and $O(\log s)$ time we can locate $\operatorname{rel\,int}\Lambda$ 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 10.11). We now drop one vector from every pair for which the location relative to $\operatorname{rel\,int}\Lambda$ 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 $O(\gamma(d-1)\log s)$ time on $O(s)$ processors, and $O(\gamma(d-1))$ oracle calls. The sequential time bound is $O(\gamma(d-1)s\log s)$ plus $O(\gamma(d-1)\log s)$ oracle calls. In parallel, using $O(s)$ processors, the time is $O(\gamma(d-1)\log^2 s)$ plus $O(\gamma(d-1)\log s)$ oracle calls. ∎

10.3. Algorithm for extended parametric minimum cycle

The algorithm described below solves Problem 10.3. It finds a vector $\boldsymbol{\lambda} \in \operatorname{rel\,int}\Lambda$, 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 $L_{\{\boldsymbol{f}(C)|\ C\in\mathcal{C}\}}$ of the linear functions defined by $\boldsymbol{f}(\mathcal{C})$ is a minimal weak approximation of g. The number of cycles in $\mathcal{C}$ is at most $2d$.

Definition 10.13 Consider a scalar minimum cycle algorithm, where the only primitive operations on expressions that depend on the edge weights are additions, multiplications by scalars, and comparisons. We define the corresponding *lifted algorithm* for input graphs of the form $G = (V, E, \boldsymbol{f}, \boldsymbol{w})$. The weight of an edge $e \in E$ on these input graphs is an $(\ell+1)$-tuple $(w_1(e), \ldots, w_\ell(e), \boldsymbol{f}(e))$, where $\boldsymbol{f}(e) \in R^d$ is viewed as a $(d-1)$-dimensional linear function and $\boldsymbol{w}(e) \equiv (w_1(e), \ldots, w_\ell(e))$. The lifted algorithm is an extension of the scalar minimum cycle algorithm that operates on such $(\ell+1)$-tuples instead of scalars. The extension of the operations of addition and multiplication by a scalar is straightforward, namely, given tuples $(\boldsymbol{w}_1, \boldsymbol{f}_1)$, $(\boldsymbol{w}_2, \boldsymbol{f}_2)$ their sum is $(\boldsymbol{w}_1+\boldsymbol{w}_2, \boldsymbol{f}_1+\boldsymbol{f}_2)$, and the multiplication by a scalar α is $(\alpha\boldsymbol{w}_1, \alpha\boldsymbol{f}_1)$. 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 have the partial order $<_\Lambda$ (see Definition 10.10). 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 linear functions $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$. For this purpose, the lifted algorithm computes the hyperplane $H(\boldsymbol{f}_1, \boldsymbol{f}_2)$ and solves Problem 10.4 (hyperplane query) relative to H. A set of hyperplane queries is resolved by performing "oracle" calls (see Subsection 10.2). The lifted algorithm maintains a set $\mathcal{H}$ of closed halfspaces which initially is empty. The hyperplane query decides whether or not 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 an oracle call resulted in a solution to Problem 10.3. Otherwise, the resolved hyperplane query tells us which of the halfspaces defined by $H(\boldsymbol{f}_1, \boldsymbol{f}_2)$ (see Definition 10.10) contains the set $\operatorname{rel\,int}\Lambda$, so this hyperplane is added to $\mathcal{H}$.

Algorithm 10.14 [Extended parametric minimum cycle]

Step 1. Run the lifted minimum cycle algorithm, collecting into $\mathcal{H}$ all the halfspaces resulting from oracle calls where comparisons are resolved. Either some oracle call resulted in a global solution, or otherwise, the algorithm terminates normally. Denote by C_M the minimum cycle found.

Step 2. Denote by P the intersection of the halfspaces in $\mathcal{H}$.

i. Compute $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} P$. This amounts to a linear programming problemwith $d-1$ variables and $|\mathcal{H}|$ constraints, and hence it can be solved in $O(|\mathcal{H}|)$ sequential time [23]. Note that the size of $\mathcal{H}$ is bounded by the number of oracle calls.

ii. If the function $L_{\boldsymbol{f}(C_M)}$ is not constant on R^d_1, that is, not all the coefficients $f_1(C_M), f_2(C_M), \ldots, f_{d-1}(C_M)$ equal zero, then g is unbounded. Otherwise,

iii. consider $g(\boldsymbol{\lambda}^*) = f_d(C_M)$.

- If $g(\boldsymbol{\lambda}^*) > 0$, then output $\boldsymbol{\lambda}^*$ and stop. Otherwise,
- the function $L_{\boldsymbol{f}(C_M)}$ is a weak approximation of g, and $P = \Lambda$. Hence, $\boldsymbol{\lambda}^* \in \operatorname{rel\,int}\Lambda$. Output $\boldsymbol{\lambda}^*$ and $\mathcal{C} = \{C_M\}$.

10.4. Correctness

If an oracle call results in a solution in Step 1 of Algorithm 10.14, 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 closed halfspaces is obtained. Recall that if an oracle call on a hyperplane H did not result in a solution then the returned halfspace h has the following properties: (i) if the function g is bounded then $\Lambda \subset h$ but $\Lambda \not\subset H$, (ii) if the function g is unbounded, then it must be bounded on the hyperplane H, and unbounded on the halfspace h. Let P be the polyhedron $P = \bigcap_{h \in \mathcal{H}} h$. It follows that if g is bounded then $P \supset \Lambda$, and if g is unbounded then it must be bounded outside and on the boundary of P. Note that P must be full dimensional ($\dim P = d-1$), for if not then it must be contained in one of the query hyperplanes, which contradicts the previous statement.

Observe that for all pairs $\boldsymbol{a}_1, \boldsymbol{a}_2$ of vectors compared by the lifted minimum cycle algorithm, one of the following must hold: either $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$ and $\boldsymbol{a}_1 <_P \boldsymbol{a}_2$, or $\boldsymbol{a}_2 <_\Lambda \boldsymbol{a}_1$ and $\boldsymbol{a}_2 <_P \boldsymbol{a}_1$. The latter is obvious when we perform an oracle call for each hyperplane query, and it is easy to see that it still holds when we employ the multi-dimensional search technique (see Problem 10.11 and Proposition 10.12) and solve these hyperplane queries by a smaller number of oracle calls. Thus, the vector value $\boldsymbol{f}(C_M)$ of the minimum cycle found by the algorithm must be such that $\boldsymbol{f}(C_M) <_\Lambda \boldsymbol{f}(C)$ and hence $\boldsymbol{f}(C_M) <_P \boldsymbol{f}(C)$ for any $\boldsymbol{w}$-minimal cycle C. It follows that $g(\boldsymbol{\lambda}) = \boldsymbol{f}(C_M)^T\boldsymbol{\lambda}$ for all $\boldsymbol{\lambda} \in P$. Thus, g is unbounded if and only if $\boldsymbol{f}(C_M)$ is not a constant, and the correctness of step ii follows. To show the correctness of step iii assume that $\boldsymbol{f}(C_M)$ is a constant, and thus $g = f_d(\boldsymbol{\lambda})$ for all $\boldsymbol{\lambda} \in P$. Since $P \supset \Lambda$ we have $P = \Lambda$. It follows that $\lambda^* \in \operatorname{rel\,int} \Lambda$, $\operatorname{aff} \Lambda = R_1^d$, and $L_{\boldsymbol{f}(C_M)}$ is a minimal weak approximation of g.

10.5. Complexity

The complexity of the algorithm is related to the number of oracle calls. We would like to resolve many hyperplane queries by performing only a polylogarithmic number of oracle 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 10.12.

Theorem 10.15 *Algorithm 10.14 can be implemented with complexity as follows (where $m = |E|$ and $n = |V|$).*

i. $O(\log^{2d} n + \log^d m)$ *parallel time on* $O(n^3 + m)$ *processors.*

ii. $O(m(\log^{2d} n + \log^d m))$ *sequential time, when* $m = \Omega(n^3 \log n)$.

iii. $O(\log^{2d} n(n^3 + m))$ *sequential time, when* $m = O(n^3 \log n)$ *and* $m = \Omega(n^2)$.

iv. $O(n^3 \log^{2(d-2)} n + nm \log^{2(d-1)} n)$ *sequential time, when* $m = O(n^2)$.

Proof: The problem of all-pairs shortest path can be solved in $O(\log^2 n)$ time using n^3 processors by the Floyd-Warshall algorithm [7]. The algorithm for this problem runs in $O(\log n)$ phases. During 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 10.11, with $s = m$ for the first phase, and $s = n^3$ for the remaining $O(\log n)$ phases. The complexity is analyzed in Proposition 10.12.

Denote by T_d and R_d, respectively, the sequential time complexity and the parallel time complexity with $n^3 + m$ processors, of the d-dimensional problem. Recall from Theorem 10.5 and Remark 10.9 that the time complexity of one oracle call is $3T_{d-1} = O(T_{d-1})$ on a single processor and $3R_{d-1} = O(R_{d-1})$ on $O(n^3 + m)$ processors. When $d = 1$, an oracle call can be implemented simply by a scalar minimum cycle algorithm. We derive recursion formulas for R_i and T_i. The oracle calls are executed sequentially. First we derive an expression for the parallel complexity when $d = 1$. 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 $R_1 = O(\log^2 n + \log m)$. It follows that

$$R_d = O(\log^3 n + \log^2 m + (\log^2 n + \log m)R_{d-1}),$$

which proves i.

The sequential complexity for $d = 1$ is $T_1 = O(\min\{nm, m + n^3\})$. Parts ii-iv of the proposition follow from the recursion:

$$T_d = O(n^3 \log n + m + (\log^2 n + \log m)T_{d-1}).$$

The above analysis applies only to the complexity of Step 1 of the algorithm. In Step 2 we compute $\boldsymbol{\lambda} \in \operatorname{rel\,int} P$. This is done by solving a linear program with $O(\mathcal{H})$ equations and $O(d)$ variables. The size of $\mathcal{H}$ is at most the number of oracle calls performed, that is, $|\mathcal{H}| = O(\log^2 n + \log m)$. Therefore the complexity of Step 2 is dominated by that of Step 1. ∎

Remark 10.16 There are hidden constant factors in the complexities stated in Theorem 10.15, which depend on the dimension d. First, there is an extra factor of d on the serial time complexity and the number of parallel processors, since most "operations" are done on vectors in R^d and take d time units. Second, there is an $O(3^d\gamma(d-1))$ factor on the time complexities. The factor $\gamma(d-1) = O((5/9)^d 3^{(d-1)^2})$ is due to the multi-dimensional search (see Subsection 10.2), and the factor of $O(3^d)$ is due to the fact that an oracle call involves three calls to a problem of lower dimension. It follows that the constant factor in the serial time complexity is $O(d3^d\gamma(d-1))$. In the parallel case there is a factor of $O(3^d\gamma(d-1))$ for the time complexity, and a factor of $O(d)$ on the number of processors.

11. Parametric extensions of problems

The technique introduced here to obtain strongly polynomial algorithms for the parametric extensions of the minimum cycle and the minimum cycle-mean problems is a general tool. It is applicable to a variety of other problems, where we are given a strongly polynomial algorithm for a problem and want to obtain a strongly polynomial algorithm for a parametric extension of the problem (when the number of parameters is fixed). We state the conditions where this technique is applicable and present applications.

Definition 11.1 [Parametric extensions]

i. A *problem* $S : \mathcal{P} \to R$ is a mapping from a set $\mathcal{P}$ of instances into the set of real numbers. We say that $S(P)$ is the *solution* of the problem for the instance $P \in \mathcal{P}$. Suppose that every instance $P \in \mathcal{P}$ has a *size* $\|P\|$ associated with it. The size of an instance is not necessarily defined to be the number of bits in its representation. It may be any natural parameter (for example, the number of edges in a weighted graph).

ii. Let $\mathcal{A}$ be an algorithm that computes $S(P)$. Denote by $T_{\mathcal{A}}(P)$ the number of elementary operations the algorithm performs on the instance P. The algorithm $\mathcal{A}$ is *polynomial* if $T_{\mathcal{A}}(P) = O(p(\|P\|))$ for some polynomial $p(\bullet)$.

iii. A *d-parametric extension* $P^d = (\mathcal{M}, \mathcal{Q})$ of $\mathcal{P}$ is defined as follows, where $\mathcal{Q} \subset R^d$ is a polyhedron given as an intersection of k halfspaces, and $\mathcal{M} : \mathcal{Q} \to \mathcal{P}$ is a mapping from points $\boldsymbol{\lambda} \in \mathcal{Q}$ to instances of $\mathcal{P}$. The extension P^d corresponds to a subset of instances $\{\mathcal{M}(\boldsymbol{\lambda}) \mid \boldsymbol{\lambda} \in \mathcal{Q}\} \subset \mathcal{P}$. We refer to $\mathcal{M}(\boldsymbol{\lambda}) \in \mathcal{P}$ as the instance of $\mathcal{P}$ *induced* by $\boldsymbol{\lambda}$. For an extension P^d, we define $g : \mathcal{Q} \to R$ as a mapping from vectors $\boldsymbol{\lambda} \in \mathcal{Q}$ to the solution of the corresponding induced instance $g(\boldsymbol{\lambda}) = S(\mathcal{M}(\boldsymbol{\lambda}))$. A *solution of the parametric extension* P^d is defined as follows. Consider the maximum of $g(\boldsymbol{\lambda})$. If it is finite, a solution consists of the maximum and a vector $\boldsymbol{\lambda} \in R^d$ that belongs to the relative interior of the set of vectors which maximize S. Formally, if $\mathcal{Q}$ is empty or if $S(\mathcal{M}(\boldsymbol{\lambda}))$ is unbounded on $\mathcal{Q}$, these facts are recognized. Otherwise, a pair $(m, \boldsymbol{\lambda}^*) \in R \times R^d$, where $m = \max_{\lambda \in \mathcal{Q}} g(\boldsymbol{\lambda})$, and $\boldsymbol{\lambda}^* \in \operatorname{rel\,int}\{\boldsymbol{\lambda} \mid g(\boldsymbol{\lambda}) = m\}$ is computed. We denote $T = \max_{\lambda \in \mathcal{Q}} T_{\mathcal{A}}(\mathcal{M}(\boldsymbol{\lambda}))$.

Theorem 11.2 *Let $S : \mathcal{P} \to R$ be a problem in the sense of Definition 11.1. Let $\mathcal{A}$ be an algorithm that evaluates S, and let $P^d = (\mathcal{M}, \mathcal{Q})$ (where $|\mathcal{Q}| = k$) be a corresponding parametric extension. We assume that*

i. the function g is concave,

ii. the mapping $\mathcal{M}$ is computable by a piecewise affine algorithm $\mathcal{A}_{\mathcal{M}}$ (see Definition 8.2) in less than T operations, and

iii. the combined algorithm which computes an instance $\mathcal{A}_{\mathcal{M}}(\boldsymbol{\lambda}) \in \mathcal{P}$ and applies $\mathcal{A}$ to $\mathcal{A}_{\mathcal{M}}(\boldsymbol{\lambda})$, is piecewise affine.

Denote by C the maximum (over $\boldsymbol{\lambda} \in \mathcal{Q}$) number of comparisons performed by the combined algorithm. Suppose the comparisons can be divided into r sets of sizes $C_1, \ldots, C_r$ ($C = \sum_{i=1}^{r} C_i$) such that the algorithm runs in r phases, where C_i independent comparisons are performed in phase i.

Under these conditions, the d-parametric extension P^d can be solved using

$$\beta(d)kT(\sum_{i=1}^{r}\lceil\log C_i\rceil)^d \text{ operations, where } \beta(d) = 3^{O(d^2)} \ .$$

Remark 11.3 In the above formulation we defined a problem as a mapping into the set of real numbers $S : \mathcal{P} \to R$. The results generalize to cases where the range of S is R^ℓ for $\ell > 1$ and the notions of maximum and concavity are defined with respect to the lexicographic order as follows: We say that a function $g : \mathcal{Q} \subset R^d \to R^\ell$ is *concave* with respect to the lexicographic order $\leq_{\text{lex}}$ if for every $\alpha \in [0, 1]$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{Q}$,

$$\alpha g(\boldsymbol{x}) + (1-\alpha)g(\boldsymbol{y}) \leq_{\text{lex}} g(\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \ .$$

Applications where the range of S is R^2 were given in [5].

Below we present some applications of Theorem 11.2. Norton, Plotkin, and Tardos [24] applied a similar scheme and presented additional applications.

Adding variables to LP's with two variables per inequality . Linear programming problems with at most two variables in each constraint and in the objective function were shown to have a strongly polynomial time algorithm by Megiddo [22]. Lueker, Megiddo and Ramachandran [19] gave a polylogarithmic time parallel algorithm for the problem which uses a quasipolynomial number of processors. The best known time bounds for the problem are given in [2]. Cosares, using nested parametrization, extended Megiddo's strong polynomiality result to allow objective functions which have a fixed number of nonzero coefficients. This result can be further extended to include the following. For a fixed d, we consider linear programming problems as above, but we allow certain d additional variables to appear anywhere in the constraints and in the objective function without being "counted." This problem is a d-parameter extension of the two variables per constraint problem, where the "parameters" are the d additional variables. For each choice of values for the parameters we have a corresponding induced system with two variable per constraint. It is easy to verify that the conditions of Theorem 11.2 hold. Hence, this class of problems also has a strongly polynomial time algorithm, and a polylogarithmic time parallel algorithm which uses a quasipolynomial number of processors.

Parametric flow problems. Theorem 11.2 was applied in [5] to generate strongly polynomial algorithms for parametric flow problems with a fixed number of parameters and to some constrained flow problems with a fixed number of additional constraints. Complementing results showing the P-completeness of these problems when the number of parameters is not fixed, were also given.

References.

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

[2] E. Cohen. *Combinatorial Algorithms for Optimization Problems.* PhD thesis, Department of Computer Science, Stanford University, Stanford, Ca., 1991.

[3] E. Cohen and N. Megiddo. Strongly polynomial and NC algorithms for detecting cycles in dynamic graphs. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 523–534. ACM, 1989.

[4] E. Cohen and N. Megiddo. Maximizing concave functions in fixed dimension. Technical Report RJ 7656 (71103), IBM Almaden Research Center, San Jose, CA 95120-6099, August 1990.

[5] E. Cohen and N. Megiddo. Complexity analysis and algorithms for some flow problems. In *Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms*, pages 120–130. ACM-SIAM, 1991.

[6] E. Cohen and N. Megiddo. Recognizing properties of periodic graphs. In P. Gritzmann and Sturmfels B., editors, *Applied Geometry and Discrete Math. The "Victor Klee Festschrift"*, volume 4, pages 135–146. ACM AMS, 1991.

[7] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms.* McGraw-Hill Book Co., New York, 1990.

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

[9] J. Farkas. Theorie der einfachen ungleichungen. *J. Reine und Angewandte Math.*, 124:1–27, 1902.

[10] B. Grötschel, L. Lovasz, and Schrijver A. *Geometric algorithms and combinatorial optimization.* Springer-Verlag, New York, 1988.

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

[12] K. Iwano. Some problems on doubly periodic infinite graphs. Technical Report CS-TR-078-87, Princeton University, 1987.

[13] K. Iwano and K. Steiglitz. Testing for cycles in infinite graphs with periodic structure. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 46–53. ACM, 1987.

[14] K. Iwano and K. Steiglitz. Planarity testing of doubly connected periodic infinite graphs. *Networks*, 18:205–222, October 1988.

[15] K. Iwano and K. Steiglitz. A semiring on convex polygons and zero-sum cycle problems. *SIAM J. Comput.*, 19:883–901, 1990.

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

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

[18] K. S. Kosaraju and G. F. Sullivan. Detecting cycles in dynamic graphs in polynomial time. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 398–406. ACM, 1988.

[19] G. S. Lueker, N. Megiddo, and V. Ramachandran. Linear programming with two variables per inequality in poly log time. *SIAM J. Comput.*, 19(6):1000–1010, 1990.

[20] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. of Oper. Res.*, 4:414–424, 1979.

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

[22] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12:347–353, 1983.

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

[24] C. H. Norton, S. A. Plotkin, and É. Tardos. Using separation algorithms in fixed dimension. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 377–387. ACM-SIAM, 1990.

[25] J. B. Orlin. Some problems in dynamic/periodic graphs. In W. R. Pullyblank, editor, *Progress in combinatorial optimization*, pages 273–293, Orlando, Florida, 1984. Academic Press.

[26] V. P. Roychowdhury. *Derivation, Extensions, and Parallel Implementation of Regular Iterative Algorithms*. PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, Ca., 1989.

[27] V. P. Roychowdhury and T. Kailath. Study of parallelism in regular iterative algorithms. In *Proc. of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 367–376. ACM, 1990.

Contents.

1 Introduction **1**

2 Preliminaries **5**

3 An overview **5**

4 Detecting zero-cycles **7**

4.1 Computing a witness or a separating vector 9

4.2 Computing the partition 10

4.3 The algorithm 13

5 Geometric lemmas **14**

6 Applications of the zero-cycle detection algorithm **16**

6.1 Scheduling 16

6.2 Strong connectivity 18

7 Discussion **20**

8 Parametric minimum cycle: preliminaries **20**

8.1 The oracle problem 24

8.2 Geometric lemmas 25

8.3 Back to zero-cycles 29

9 Algorithm for parametric minimum cycle-mean **30**

10 Algorithm for parametric minimum cycle **33**

10.1 Hyperplane queries 34

10.2 Employing multi-dimensional search 35

10.3 Algorithm for extended parametric minimum cycle 37

10.4 Correctness 39

10.5 Complexity 39

11 Parametric extensions of problems **41**