Recognizing Properties of Periodic Graphs

Edith Cohen* and Nimrod Megiddo†

June 1990

Abstract

Abstract. A periodic (dynamic) graph is an infinite graph with a repetitive structure and a compact representation. A periodic graph is represented by a finite directed graph, called the dependence or the static graph, with d-dimensional integer vector weights associated with its edges. For every vertex in the dependence graph there corresponds a d-dimensional lattice in the periodic graph. For every edge (u, v) in the dependence graph with vector weight $\boldsymbol{a}$, there are infinitely many edges in the periodic graph, namely, from every point on the lattice corresponding to u to the point shifted by $\boldsymbol{a}$ on the lattice corresponding to v. Periodic graphs are used, for example, to model VLSI circuits and systems of uniform recurrence relations. In this paper we give algorithms to compute weakly connected components, to test bipartiteness, and to compute a minimum average cost spanning tree for d-dimensional periodic graphs.

1. Introduction

Periodic graphs are infinite graphs with a repetitive structure. They have a finite description (the "period") given by a directed graph with integer vector weights associated with the edges. A more formal definition follows.

Definition 1..1. Given $G = (V, E, \boldsymbol{f})$, where $\boldsymbol{f} : E \to Z^d$ is a weight function on the edges of G, the *periodic (dynamic)* graph $G^* = (V^*, E^*)$ defined by G has:

$$V^* = \{(\boldsymbol{z}, v) \mid \boldsymbol{z} \in Z^d, v \in V\}$$

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

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

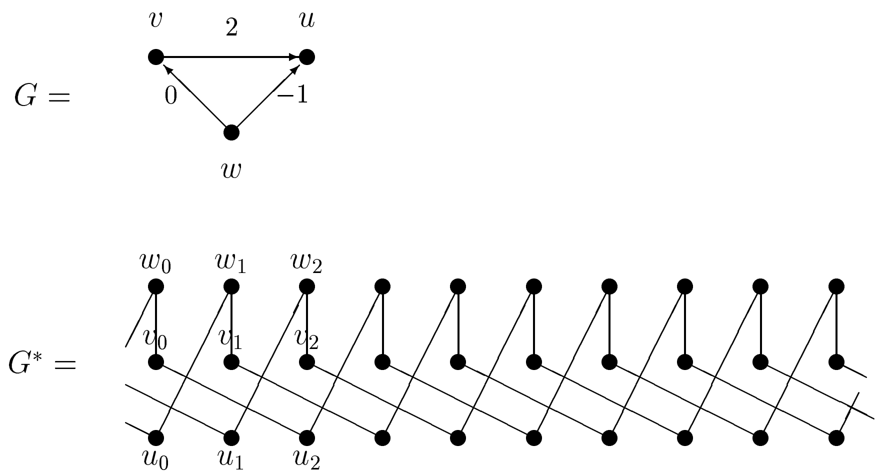

Figure 1: Example of a one-dimensional periodic graph.

$$E^* = \{((\boldsymbol{z}, u); (\boldsymbol{z} + \boldsymbol{f}(u, v), v)) \mid (u, v) \in E\} \ .$$

The *dimension* of G^* is d. The graph G is called the *dependence* or the *static* graph. For an edge $(u, v) \in E$, the edges *generated* by (u, v) are $((\boldsymbol{z}, u); (\boldsymbol{z} + \boldsymbol{f}(u, v), v))$ $(\boldsymbol{z} \in Z^d)$.

See Figure 1 and Figure 2 for examples of one- and two-dimensional periodic graphs, respectively. Note that for our purposes the periodic graph as defined above may be viewed as undirected. Therefore, a directed edge $(u, v) \in E$ in the dependence graph with $\boldsymbol{f}(u, v) = \boldsymbol{z}$ is equivalent to a directed edge (v, u) with weight $\boldsymbol{f}(v, u) = -\boldsymbol{z}$. The undirected infinite periodic graph defined by G does not change if (u, v) is replaced by (v, u). Periodic graphs can be used to model a system of uniform recurrence equations [12] and VLSI circuits [8, 9]. We are interested in testing properties of a periodic graph by working on the defining finite dependence graph. Previous work on periodic graphs was concerned with scheduling [2, 12, 15], planarity testing [7], cycle detection [1, 2, 9, 12, 13] and finding strongly connected components [2] of directed periodic graphs. Orlin [14] defined the concept of a one-dimensional dynamic/periodic graph and discussed the complexity of recognizing some properties of it. In particular, he considered finding the connected and strongly connected components, recognizing bipartiteness, and computing a minimum average cost spanning tree. Iwano [10] studied two-dimensional periodic graphs and gave algorithms that test bipartiteness, and compute connected components.

The current paper generalizes some of the results of Orlin and of Iwano to higher dimensions. We consider here the problems of recognizing connectivity, bipartiteness, and computing a minimum average cost spanning tree. The periodic graph of Figure 1 is bipartite and has three connected components. The periodic graph of Figure 2 is connected but is not bipartite.

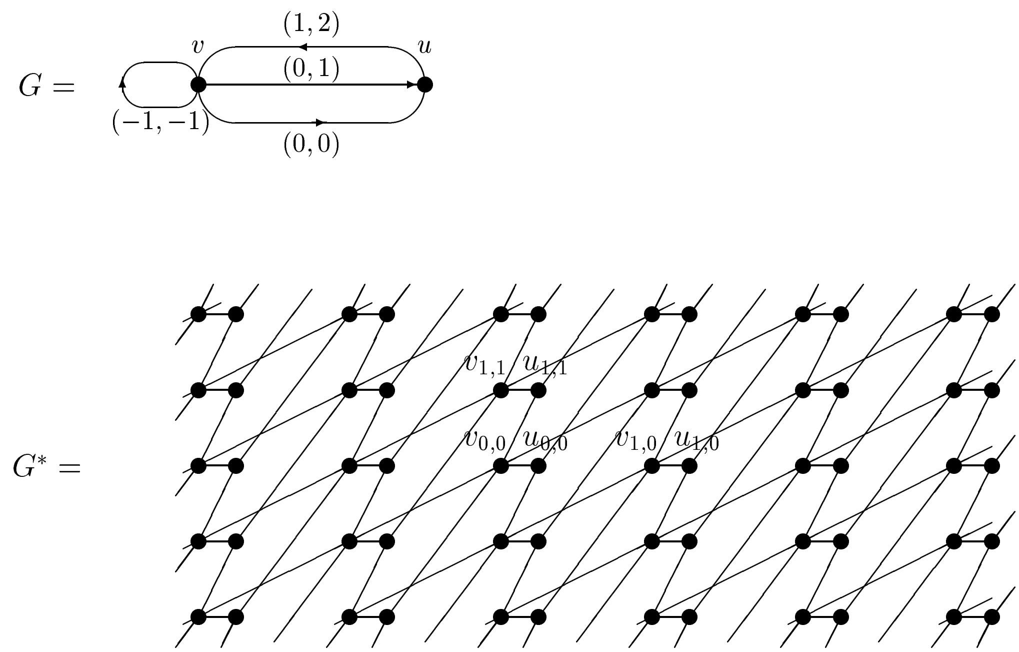

Figure 2: Example of a two-dimensional periodic graph.

Section 2 gives an algorithm for the connectivity problem. This algorithm is used in the succeeding sections. It is a generalization of the algorithms of [14, 7]. The connectivity algorithm computes dependence graphs whose corresponding periodic graphs are isomorphic to the connected components of the original periodic graph. Section 3 gives an algorithm that checks whether the periodic graph is bipartite. Section 4 gives an algorithm for finding a tree that spans a given periodic graph and has an asymptotically minimal average cost. The problems of recognizing bipartiteness and of computing a minimum average cost spanning tree in one-dimensional periodic graphs were studied by Orlin [14]. The problems of finding connected components and testing bipartiteness in two-dimensional periodic graphs were studied by Iwano [7]. In this paper we generalize these results to higher dimensions.

2. Connectivity

Problem 2..1. Given $G = (V, E, \boldsymbol{f})$, recognize whether or not the periodic graph G^* is connected. If G^* is not connected, find $G_i = (V_i, E_i, \boldsymbol{f}_i)$, $i = 1, \ldots, r$, such that the corresponding periodic graphs G_i^* are isomorphic to the connected components of G^*.

An algorithm that solves Problem 2..1 for one-dimensional periodic graphs was given by Orlin [14]. An algorithm for two-dimensional periodic graph was given by Iwano [7]. Orlin also proved that if the dependence graph G is connected, then all the connected components of G^* are isomorphic, and therefore only one dependence graph suffices for describing the connected components of G^*. We show that the same is true in higher dimensions.

The following proposition is a restatement of Lemma 4 of [14]. The proof given there works for any dimension.

Definition 2..2. Let $\boldsymbol{f} : E \to R^d$ be a weight function on the edges of a graph. A function $\boldsymbol{d} : V \to R^d$ is called a *potential.* Denote by $\boldsymbol{f}^{(d)} : E \to R^d$ the function $\boldsymbol{f}^{(d)}(i,j) = \boldsymbol{f}(i,j) + \boldsymbol{d}(i) - \boldsymbol{d}(j)$.

Proposition 2..3. *For any potential function $\boldsymbol{d} : V \to Z^d$, if $G = (V, E, \boldsymbol{f})$ and $G' = (V, E, \boldsymbol{f}^{(d)})$, then G^* is isomorphic to $(G')^*$ and the isomorphism $\mathcal{H} : V^* \to (V')^*$ is given by $\mathcal{H}(\boldsymbol{z}, i) = (\boldsymbol{z} + \boldsymbol{d}(i), i)$.*

Definition 2..4. If $G = (V, E, \boldsymbol{f})$ contains a spanning tree $T \subset E$ such that for all the edges $e \in T$, $\boldsymbol{f}(e) = \mathbf{0}$, then G is in a *basic form.*

The following corollary of Lemma 4 of [14] generalizes to vector weights:

Corollary 2..5. *Given $G = (V, E, \boldsymbol{f})$ and a spanning tree $T \subset E$ of G, a potential function $\boldsymbol{d}$ can be computed such that for the dependence graph with new weights $G' = (V, E, \boldsymbol{f}^{(d)})$, the periodic graph $(G')^*$ is isomorphic to G^* and $\boldsymbol{f}^{(d)}(e) = \mathbf{0}$ for all $e \in T$.*

Proof: This is done by computing the distances on the tree T from one vertex to all the other vertices, and choosing $\boldsymbol{d} : V \to Z^d$ to be the distance function. ∎

We refer to the above procedure as a transformation of G into a basic form.

Definition 2..6. Given an integer matrix $\boldsymbol{A} \in Z^{d \times m}$, we define an equivalence relation $\stackrel{A}{\sim}$ on Z^d as follows. Two vectors $\boldsymbol{a}, \boldsymbol{b} \in Z^d$ are in the same equivalence class ($\boldsymbol{a} \stackrel{A}{\sim} \boldsymbol{b}$) if and only if there exists an integer vector $\boldsymbol{z} \in Z^m$ such that $\boldsymbol{a} - \boldsymbol{b} = \boldsymbol{A}\boldsymbol{z}$.

Remark 2..7. Observe that if $d = m = 1$, then the relation $a \stackrel{A}{\sim} b$ simply means $a \equiv b \pmod{A_{11}}$. In general, an equivalence class of the relation $\stackrel{A}{\sim}$ is simply a translate (by a vector in Z^d) of the lattice spanned by the columns of $\boldsymbol{A}$. When $m < d$, there are infinitely many equivalence classes. When $m = d$ and $\boldsymbol{A}$ is nonsingular, there are finitely many equivalence classes. When $m > d$, there exists a $d \times d$ matrix $\boldsymbol{B}$ that defines the same equivalence relation as $\boldsymbol{A}$ (see Problem 2..8).

Problem 2..8. Let

$$\mathcal{L} = \left\{ \sum_{j=1}^{m} z_j \boldsymbol{a}^j \mid z_j \in Z \right\} \subset Z^d .$$

be the lattice spanned by m vectors $\boldsymbol{a}^1, \ldots, \boldsymbol{a}^m \in Z^d$. Find vectors $\boldsymbol{b}^1, \ldots, \boldsymbol{b}^\ell \in Z^d$, a permutation $(i_1, \ldots, i_d)$ of $\{1, \ldots, d\}$, and vectors $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m \in Z^\ell$ with the following properties:

1. $\mathcal{L} = \{\sum_{j=1}^{\ell} z_j \boldsymbol{b}^j | z_j \in Z\}$.
2. The vectors $\boldsymbol{b}^j$ are linearly independent.
3. The permutation $(i_1, \ldots, i_d)$ is such that for $j = 1 \ldots \ell$, $\boldsymbol{b}^j_{i_k} = 0$ for $k < j$.
4. For all $j \in \{1, \ldots m\}$ $\boldsymbol{a}^j = \boldsymbol{B}\boldsymbol{z}^j$, where $\boldsymbol{B}$ is the matrix whose columns are the $\boldsymbol{b}^j$'s.

Proposition 2..9. *Problem 2..8 can be solved in $O(d^2 m B(d \log(d\|\boldsymbol{A}\|)))$ time, where $\|\boldsymbol{A}\| = \max_{ij} |\boldsymbol{a}^j_i|$ and $B(t)$ bounds the time required to perform the extended Euclidean algorithm on two t bit integers.*

Proof: Denote by $\boldsymbol{A} \in Z^{d \times m}$ the matrix whose j'th column is $\boldsymbol{a}^j$ $(j = 1, \ldots, m)$. Note that the rows of $\boldsymbol{A}$ are not necessarily independent. We begin by identifying a maximal set of linearly independent rows of $\boldsymbol{A}$. This can be done in polynomial time by "careful" Gaussian elimination [4] . Without loss of generality, suppose that the first ℓ rows comprise such a set (otherwise, the rows of $\boldsymbol{A}$ can be permuted accordingly), and denote the submatrix by $\boldsymbol{A}' \in Z^{\ell \times m}$. Let $\boldsymbol{A}'' \in Z^{(d-\ell) \times m}$ be the submatrix consisting of the last $d - \ell$ rows of $\boldsymbol{A}$. Moreover, during the process of identifying the submatrices, we also find a $\boldsymbol{Y} \in Q^{(d-\ell) \times \ell}$ such that $\boldsymbol{A}'' = \boldsymbol{Y}\boldsymbol{A}'$, and the size of each entry of $\boldsymbol{Y}$ is bounded by the determinants of the square submatrices of $\boldsymbol{A}$ (that is, $\|\boldsymbol{Y}\| = O(\|\boldsymbol{A}\|^\ell)$). This computations can be done in $O(\ell d m)$ arithmetic operations on numbers of size $O(\|\boldsymbol{A}\|^\ell)$. Let $\mathcal{L}' \subset Z^\ell$ denote the lattice spanned by the columns of $\boldsymbol{A}'$. It is easy to verify that $(\boldsymbol{b}', \boldsymbol{b}'')$ $(\boldsymbol{b}' \in R^\ell$, $\boldsymbol{b}'' \in R^{d-\ell})$ is in $\mathcal{L}$ if and only if $\boldsymbol{b}' \in \mathcal{L}'$ and $\boldsymbol{b}'' = \boldsymbol{Y}\boldsymbol{b}'$. Moreover, if $\boldsymbol{u}^1, \ldots, \boldsymbol{u}^\ell \in \mathcal{L}'$ span the lattice $\mathcal{L}'$, then the vectors $(\boldsymbol{u}^1, \boldsymbol{Y}\boldsymbol{u}^1), \ldots, (\boldsymbol{u}^\ell, \boldsymbol{Y}\boldsymbol{u}^\ell)$ span the lattice $\mathcal{L}$.

Denote the columns of $\boldsymbol{A}'$ by $\boldsymbol{a}'^j$. Observe that the lattice $\mathcal{L}'$ spanned by the $\boldsymbol{a}'^j$'s does not change if any vector $\boldsymbol{a}'^j$ is replaced by $-\boldsymbol{a}'^j$, or by $\boldsymbol{a}'^j + \boldsymbol{a}'^k$ or $\boldsymbol{a}'^j - \boldsymbol{a}'^k$ for some $k \neq j$. By a classical theorem of Hermite [6] (note that $\boldsymbol{A}'$ has a full row rank), the matrix $\boldsymbol{A}'$ can be triangulated by elementary (unimodular) column operations, i.e., the only operations used in this transformation are additions of integral multiples of one column to another, and exchanges of columns. Neither of these operations alters the lattice spanned by the columns of $\boldsymbol{A}'$. The matrix $\boldsymbol{A}'$ is transformed so that the last $m - \ell$ ($\ell \leq d$) columns become zero, and the first ℓ columns comprise an integer lower triangular matrix with nonzero diagonal entries. If the diagonal entries are maximal in their rows, then the triangular matrix is unique and is called the Hermite normal form of $\boldsymbol{A}'$. It follows from Hermite's proof that this can be accomplished in at most ℓm applications of the extended Euclidean algorithm for the g.c.d. (of two numbers) plus $O(\ell^2 m)$ arithmetic operations on integers. Kannan and Bachem [11] proposed an improvement of this algorithm, where all the operations are done on integers of polynomial size. Hafner and McCurley [5] proposed an asymptotically faster algorithm where all integers encountered do not exceed $\ell^\ell \|\boldsymbol{A}'\|^{2\ell}$. Their algorithm runs in $O(\ell^2 m B(\ell \log(\ell \|\boldsymbol{A}\|)))$ bit operations. Moreover, for the simpler task of triangulating $\boldsymbol{A}'$, they used fast matrix multiplication to get even better time bounds. Using $O(\ell^{\theta-1} m \log(2m/\ell) B(\ell \log(\ell \|\boldsymbol{A}'\|)))$ bit operations (where $\theta < 2.376$, following Coppersmith and Winograd [3]), the algorithm computes a triangular matrix whose entries have absolute values bounded by $\ell^{\ell/2}\|\boldsymbol{A}'\|^\ell$.

Assuming $\boldsymbol{A}'$ has been transformed into triangular form, replace $\boldsymbol{A}''$ by $\boldsymbol{Y}\boldsymbol{A}'$. Note that the new $\boldsymbol{A}''$ has integer entries of absolute values $O(\ell^{\ell/2+1}\|\boldsymbol{A}\|^{2\ell})$. Thus, the new matrix $\boldsymbol{A}$ has for every k, $k = 1, \ldots, \ell$, $A_{kk} \neq 0$ and $A_{kj} = 0$ for $j > k$. Also, for $k = \ell + 1, \ldots, d$, $A_{kj} = 0$ for all $j > \ell$. Finally, let the vector $\boldsymbol{b}^j$ be equal to the j'th column of the matrix $\boldsymbol{A}$ ($j = 1, \ldots, \ell$). Note that properties (i) and (ii) of Problem 2..8 imply that $\ell \leq d$ and that the permutation $(i_1, \ldots, i_d)$ and the vectors $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m$ exist. Given the $\boldsymbol{b}^j$'s, the vectors $\boldsymbol{z}^j$ can be computed easily.

▮

Corollary 2..10.

1. *If $\boldsymbol{A}$ has full row rank, then Problem 2..8 can be solved in*

$$O(\ell^{\theta-1} m \log(2m/\ell) B(\ell \log(\ell \|\boldsymbol{A}'\|)))$$

bit operations (where $\theta < 2.376$).

2. *When d is fixed, Problem 2..8 can be solved in $O(m \log(\|\boldsymbol{A}\|))$ arithmetic operations.*

3. *If an integer multiple h of $\det \boldsymbol{B}$ is given, then it follows from Theorem 1 of [5] that Problem 2..8 can be solved in $O(d^2 m B(h))$ arithmetic operations.*

Algorithm 2..11 solves Problem 2..1. It first transforms the weights of G to a basic form. The algorithm continues and solves Problem 2..8 with respect to the transformed edge weights. The number of connected components of G^* is $\det \boldsymbol{B}$.

Algorithm 2..11. Connectivity

1. Compute a spanning tree $T \subset E$ of G and the appropriate potential function $\boldsymbol{d}$, and transform $\boldsymbol{f}$ to a basic form, i.e., replace $\boldsymbol{f}$ by $\boldsymbol{f}^{(d)}$ (see Corollary 2..5).

2. Denote by $\boldsymbol{a}^1, \ldots, \boldsymbol{a}^m$ the vector weights in the range of the (transformed) $\boldsymbol{f}$. Solve Problem 2..8 with respect to the $\boldsymbol{a}^j$'s, i.e., compute the vectors $\boldsymbol{b}^j, \boldsymbol{z}^j$.

3. If $\ell = d$, then the number of connected components of G^* is $\prod_{i=1}^d B_{ii}$. Otherwise, there are infinitely many connected components, each of which is an ℓ-dimensional periodic graph. Consider the new edge weights $\tilde{\boldsymbol{f}} : E \to Z^\ell$ given by $\tilde{\boldsymbol{f}}(e) = \boldsymbol{z}^j$, where $\boldsymbol{f}(e) = \boldsymbol{a}^j$. Construct the dependence graph $\tilde{G} = (V, E, \tilde{\boldsymbol{f}})$. The periodic graph $(\tilde{G})^*$ is isomorphic to each of the connected components of G^*. The isomorphism $\mathcal{H} : \tilde{V}^* \to V^*$ is given by $\mathcal{H}(\boldsymbol{a}, u) = (\boldsymbol{B}\boldsymbol{a} + \boldsymbol{z}, u)$, so that different connected components have different values of $\boldsymbol{z} \in Z^d$ associated with them.

The correctness of the algorithm is proven in the following Proposition.

Proposition 2..12. *Suppose $G = (V, E, \boldsymbol{f})$ is connected and $\boldsymbol{f}$ is in a basic form. Denote by $\boldsymbol{B} \in Z^{d\times\ell}$ the matrix sought in Problem 2..8 with respect to the vectors $\boldsymbol{f}(e)$ $(e \in E)$. Under these conditions,*

1. *If $\ell = d$, then the number of connected components of G^* is $\prod_{i=1}^d B_{ii}$.*

2. *The connected components of G^* correspond to the equivalence classes of $\stackrel{B}{\sim}$, i.e., two vertices $(\boldsymbol{a}, u), (\boldsymbol{b}, v) \in V^*$ are in the same connected component if and only if $\boldsymbol{a} \stackrel{B}{\sim} \boldsymbol{b}$.*

3. *The graph $(\tilde{G})^*$ is connected and isomorphic to each of the connected components of G^*.*

Proof: For the proof of part 1, note that $\det \boldsymbol{B}$ is the volume of a "cell" of the lattice, which is the same as the number of distinct lattices that can be obtained by an integral translation of the given lattice. To prove part 2 observe that $(\boldsymbol{a}, u)$ and $(\boldsymbol{b}, v)$ are in the same connected component if and only if $\boldsymbol{a} - \boldsymbol{b}$ equals an integer combination of the vectors $\boldsymbol{f}(E)$. Part 3 follows from part 2. The isomorphism $\mathcal{H} : \tilde{V}^* \to V^*$ is given by $\mathcal{H}(\boldsymbol{a}, u) = (\boldsymbol{B}\boldsymbol{a} + \boldsymbol{z}, u)$. ∎

Remark 2..13. The connected components of G^* are ℓ-dimensional periodic graphs. Observe that when the weights $\boldsymbol{f}$ are in basic form, then ℓ is the dimension of the vector space spanned by $\boldsymbol{f}(E)$.

Denote $m = |E|$, and $\|\boldsymbol{f}\| = \max_{e\in E} \|\boldsymbol{f}(e)\|_\infty$. Denote by $\mathrm{conn}(m, \|\boldsymbol{f}\|, d)$ the the number of operations required by the connectivity algorithm.

Proposition 2..14. *Let $B(t)$ be as in Proposition 2..9. We have*

$$\text{conn}(m, \|\boldsymbol{f}\|, d) = O(d^2 m B(d \log(d\|\boldsymbol{f}\|))) \ .$$

Proof: The computation done by Algorithm 2..11 amounts to transforming G to a basic form, and to a solution of an instance of Problem 2..8. Recall that the transformation to a basic form involves a computation of a spanning tree, and computing single source shortest paths on the tree. Therefore, the transformation is done in $O(m)$ operations. The proof follows from Proposition 2..9. ∎

Corollary 2..15. *The following are consequences of Corollary 2..10:*

1. *If the connected components of G^* are ℓ-dimensional periodic graphs, then Algorithm 2..11 requires $O(\ell^{\theta-1} m \log(2m/\ell) B(\ell \log(\ell\|\boldsymbol{f}\|)))$ operations ($\theta < 2.376$).*
2. *If the dimension d is fixed, the algorithm requires $O(m \log\|\boldsymbol{f}\|)$ operations.*
3. *Consider the problem of recognizing whether the number of connected components of G^* equals k, where k is some fixed integer. It follows from Corollary 2..10 part 3 that this task can be done in $O(d^2 m)$ time.*

3. Bipartiteness

In this section we consider the following problem:

Problem 3..1. Given a dependence graph $G = (V, E, \boldsymbol{f})$, decide whether or not G^* is bipartite.

Definition 3..2. Given a dependence graph G, define $\tilde{G} = (V, \tilde{E}, \tilde{\boldsymbol{f}})$ as follows. Let $\tilde{E}$ be the set of pairs $(u, v) \in V \times V$ such that there exists $w \in V$ with $\{(u, w), (w, v)\} \subseteq E$. For such pairs let $\tilde{\boldsymbol{f}}(u, v) = \boldsymbol{f}(u, w) + \boldsymbol{f}(w, v)$.

Remark 3..3. Note that the graph $\tilde{G}$ is well defined even when G does not have edge weights. In this case the edges of $\tilde{G}$ correspond to paths of length 2 in G. If G is connected, then it is bipartite if and only if $\tilde{G}$ has two connected components. If G is a nonbipartite connected graph, then $\tilde{G}$ is connected.

We assume in this section that G^* is connected. If this is not true, then the dependence graphs that define the connected components of G^* should be considered independently.

Algorithm 3..4. Bipartiteness

1. Construct $\tilde{G}$ as described in Definition 3..2.
2. If $\tilde{G}$ is not connected, then G^* is bipartite. Stop.
3. Otherwise, the graph $\tilde{G}^*$ is bipartite if and only if $(\tilde{G})^*$ is not connected.

Correctness: Observe that $(\tilde{G})^* = \tilde{G^*}$, that is, (u, v) is an edge of $(\tilde{G})^*$ if and only if there is a path of length two between u and v in G^*. Therefore, it follows from Remark 3..3 that G^* is bipartite if and only if the graph $(\tilde{G})^*$ is not connected. This condition is checked in Steps 2 and 3 of the algorithm.

Complexity: The algorithm amounts to constructing the graph $\tilde{G}$, and then testing if the periodic graphs corresponding to $\tilde{G}$ (or to its connected components) have one or two connected components. The construction of $\tilde{G}$ requires $O(m^2)$ time. Observe that $|\tilde{E}| = O(|E|^2)$. Thus, the time complexity of testing if $\tilde{G}^*$ has one or two connected components is $O(d^2m^2)$ (see Corollary 2..15 part 3). It follows that the time complexity of the algorithm is $O(d^2m^2)$.

4. Minimum Average Cost Spanning Tree

Orlin [14] defined a minimum average cost spanning tree of a one-dimensional periodic graph and gave an algorithm that computes one. We extend his definition to higher dimensions as follows.

Definition 4..1. Let $G = (V, E, \boldsymbol{f}, c)$ be a dependence graph together with edge *costs* given by $c : E \to R^+$.

1. Extend c to the edges of G^* in the obvious way: $c((\boldsymbol{z}, u); (\boldsymbol{z} + \boldsymbol{f}(u, v), v)) = c(u, v)$.
2. For any positive integer n, denote by $G^n = (V^n, E^n)$ the finite subgraph of G^* induced by the set of vertices $(\boldsymbol{z}, v)$ $(v \in V)$ with $|z_i| \leq n$ $(i = 1, \ldots, d)$.
3. A *minimum average cost spanning tree* (MACST) $T^* \subset E^*$ of G^* is a spanning tree for which the limit
$$\lim_{n\to\infty} (2n)^{-d} \sum_{e \in E^n \cap T^*} c(e)$$
exists and is minimal among all trees for which such a limit exists.
4. A spanning tree T^* is said to be *optimal* if each finite subtree $T_1^* \subset T^*$ is a minimum cost spanning tree of the subgraph of G^* induced by the nodes of T_1^*.

Definition 4..2. Let $\boldsymbol{B} \in Z^{d\times d}$ and let $\overset{B}{\sim}$ be the equivalence relation on Z^d defined by $\boldsymbol{B}$ in Definition 2..6. Let $\mathcal{L}$ be the lattice spanned by the columns of $\boldsymbol{B}$. For any $\boldsymbol{a} \in Z^d$, denote by $\mathcal{L}^{(a)}$ the lattice spanned by the columns of $\boldsymbol{B}$ together with $\boldsymbol{a}$. Denote by $\boldsymbol{B}^{(a)} \in Z^{d\times d}$ a matrix whose columns span the lattice $\mathcal{L}^{(a)}$. Denote by $R(\boldsymbol{B}) \subset Z^d$ any set of representatives of the equivalence classes of the relation $\overset{B}{\sim}$.

Remark 4..3. If $\boldsymbol{B}$ is full rank and triangular, then a triangular matrix $\boldsymbol{B}^{(a)}$ can be computed within $O(d^2 B(\log \det \boldsymbol{B}))$ time, which amounts to $O(d^2 \log \det \boldsymbol{B})$ arithmetic operations (see [5]).

Algorithm 4..4 below computes an optimal MACST of G^*. The algorithm maintains a set of isomorphic trees and a matrix $\boldsymbol{B}$, such that the equivalence classes of $\overset{B}{\sim}$ correspond to these trees. The infinite tree T^* computed by the algorithm is represented by a collection of pairs $(e, \boldsymbol{B})$, where $e \in E$ and $\boldsymbol{B} \in Z^{d \times i}$ is an integer matrix. Each such pair represents a set of edges of G^* :

1. When $i < d$, each such pair represents infinitely many edges: for $(u, v) \in E$, $((\boldsymbol{z}, \boldsymbol{u}); (\boldsymbol{z} + \boldsymbol{f}(u, v), v) \in T^*$ if and only if $\boldsymbol{z} \in R(\boldsymbol{B})$. The algorithm generates one such pair for each $i < d$.
2. When $i = d$, the matrix $\boldsymbol{B}$ is non-singular. The algorithm may produce as many as $m - d$ such pairs. Each pair represents finitely many edges: for $(u, v) \in E$, $((\boldsymbol{z}, u); (\boldsymbol{z} + \boldsymbol{f}(u, v), v)) \in T^*$ if and only if $\boldsymbol{z} \in R(\boldsymbol{B})$ and $\boldsymbol{z} \overset{B}{\not\sim} \boldsymbol{y}$ for all $\boldsymbol{y} \in R(\boldsymbol{B}^{(f(e))})$. Observe that the number of edges that correspond to such pairs is $|R(\boldsymbol{B})| - |R(\boldsymbol{B}^{(f(e))})|$.

Each edge $e \in E$ can occur in at most one pair $(e, \boldsymbol{B})$. Algorithm 4..4 below consists of two phases. During the first phase (steps 3, 4), pairs with $i < d$ are selected. During the second phase (steps 5, 6), pairs with $i = d$ are selected. Within each phase, the algorithm is greedy and selects the edges (with a certain desired property) according to increasing cost. Without loss of generality, assume G^* is connected.

Algorithm 4..4. Optimal Minimum Average Cost Spanning Tree

1. Find a minimum cost spanning tree T of G and transform G to the basic form corresponding to T (see Corollary 2..5).
2. Initialize $T^* \leftarrow \{(t, \{\mathbf{0}\}) | t \in T\}$.
3. Do step 4 for $i = 1, \ldots, d$, and then go to 5.
4. Find an edge $e_i \in E$ of minimum cost, such that $\boldsymbol{f}(e_i)$ is linearly independent of the weights of previously selected edges: $\boldsymbol{f}(e_j)$ $(j < i)$. Let $\boldsymbol{B} \in Z^{d \times i}$ be the matrix whose columns are the vectors $\boldsymbol{f}(e_1), \ldots, \boldsymbol{f}(e_i)$. Add the pair $(e_i, \boldsymbol{B})$ to the forest T^*.
5. At this point $\boldsymbol{B} \in Z^{d \times d}$ is a square matrix, and T^* consists of $\det \boldsymbol{B}$ isomorphic trees. Transform $\boldsymbol{B}$ into an triangular matrix that defines the same lattice (see Proposition 2..9). Repeat step 6 until $\det \boldsymbol{B} = 1$.

6. Select an edge e of minimal cost such that $\boldsymbol{f}(e) \neq \boldsymbol{B}\boldsymbol{z}$ for all $\boldsymbol{z} \in Z^d$. Add the pair $(e, \boldsymbol{B})$ to T^*. Update $\boldsymbol{B}$ to be $\boldsymbol{B}^{(f(e))}$ (see Definition 4..2 and Remark 4..3).

Correctness: Denote by $e_1, \ldots, e_s$ $(d \leq s \leq m)$ the edges selected by the algorithm. The edges $e_1, \ldots, e_d$ are selected during step 4. First we prove that T^* is indeed a tree.

Proposition 4..5. *Throughout the execution of the algorithm, the connected components of T^* constitute a collection of isomorphic trees corresponding to the equivalence classes of the relation $\overset{B}{\sim}$.*

Proof: Consider any point during the execution of the algorithm and suppose $e_1, \ldots, e_k$ are the edges that have been selected so far. Recall that two vectors are in the same equivalence class of $\overset{B}{\sim}$ if and only if they differ by an integer linear combination of the vectors $\boldsymbol{f}(e_1), \ldots, \boldsymbol{f}(e_k)$. First, observe that after step (ii) of the algorithm, T^* consists of a collection of finite trees. The vertex sets of these trees have the form $\{(i, v) | v \in V\}$. The proposition follows by induction on k. ∎

The graph G^* is connected. Therefore, if all edges are selected, we must have $\det \boldsymbol{B} = 1$. Thus, the algorithm is guaranteed to terminate with $\det \boldsymbol{B} = 1$. It follows from Proposition 4..5 that when the algorithm terminates the forest T^* has one connected component.

Proposition 4..6. *The tree T^* computed by the algorithm has a minimum average cost.*

Proof: Observe that the set $\bar{T}$, of edges of G^* corresponding to the set $T \cup \{e_1\}$, is contained in T^* and does not contain a cycle. Thus, $\bar{T}$ is a forest of G^*. We will show that it can be converted into a tree by adding an asymptotically small number of edges from E^*, so that the limit of the average cost is the same as in $\bar{T}$.

Consider the intersection of $\bar{T}$ with G^n. Imagine running any greedy algorithm for a minimum spanning tree of G^n. We show that such an algorithm selects all the edges of $\bar{T} \cap G^n$. Obviously, the edges generated by T do not introduce any cycles, and they are of minimum cost. All the other edges of G, whose weights equal $\mathbf{0}$, generate edges of G^* that introduce cycles. However, the algorithm selects the next minimum cost edge $e_1 \in E$ such that $\boldsymbol{f}(e_1) \neq \mathbf{0}$, and adds to T^* all the edges generated by e_1. Thus, no cycle is introduced.

By now, we have a forest of isomorphic trees of size $\Omega(n/\|\boldsymbol{f}\|)$ that cover G^n. The asymptotic average cost of the forest is

$$\lim_{n \to \infty} \frac{(2n)^d c(T \cup \{e_1\})}{(2n)^d} = c(T \cup \{e_1\}) \ .$$

This gives a lower bound on the asymptotic average cost of the spanning tree of G^*. Observe that it is an upper bound as well, since the number of additional edges needed to complete this forest to a tree is $O((2n)^{d-1}\|\boldsymbol{f}\|)$. It easy to see that, regardless of the choice of these additional edges, the asymptotically minimum average cost of the tree is the same as that of the initial forest, i.e., $c(T \cup \{e_1\})$. ∎

Proposition 4..7. *The tree T^* is optimal (see Definition 4..1 Part 4).*

Proof: To prove optimality, consider some subtree T_1^* of T^*. Denote by H the subgraph of G^* induced by the nodes of T_1^*. We need to show that T_1^* is a minimum cost spanning tree of H, i.e., the edges of T are the same as the ones picked by some greedy algorithm. Algorithm 4..4 is greedy within each of its two phases. Edges are selected according to increasing cost. When an edge e is selected, the algorithm adds to T^* a maximal subset of the edges generated by e, so that T^* remains acyclic. We need to show that during the first phase of the algorithm, no two trees that are connected by an edge can be connected by an edge of a lower cost. Consider the state before choosing a new edge in the first phase. Observe that all vertices of a connected component always lie on a "flat" which is a translate of the subspace spanned by the columns of $\boldsymbol{B}$ (see Proposition 4..5). The weight of any edge selected by the algorithm during step 4 is linearly independent of the columns $\boldsymbol{B}$. Thus, the new edges must lie between two such "flats". All edges of lower cost are linearly dependent on the columns of $\boldsymbol{B}$. Therefore, they lie within such flats. It follows that the new edges added to T^* are minimum cost edges, linking connected components of the current T^*. Hence, they occur in some minimum cost spanning tree. ∎

The algorithm amounts essentially to computing a minimum cost spanning tree in G, and then computing the matrix $\boldsymbol{B}$, and updating it at most m times (see Remark 4..3). Thus, the complexity is $O(n \log n + md^2 B(d \log(d\|\boldsymbol{f}\|)))$.

References

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

[2] Cohen, E., and Megiddo, N. Strongly polynomial time and NC algorithms for detecting cycles in periodic graphs. Tech. Rep. RJ 7587 (70764), IBM Almaden Research Center, San Jose, CA 95120-6099, July 1990.

[3] Coppersmith, D., and Winograd, S. Matrix multiplication via arithmetic progressions. In *Proc. 19th Annual ACM Symposium on Theory of Computing* (1987), ACM, pp. 1–6.

[4] Edmonds, J. Systems of distinct representatives and linear algebra. *J. Research of the Nat. Bur. Standards 71B* (1967), 241–245.

[5] Hafner, J. L., and McCurley, K. S. Asymptotically fast triangularization of matrices over rings. Tech. Rep. RJ 6921, IBM Almaden Research Center, San Jose, CA 95120-6099, 1989.

[6] Hermite, C. Sur l'introduction des variables continues dans la théorie des nombres. *J. Reine Angewandte Mathematik 41* (1851), 431–451.

[7] Iwano, K. Some problems on doubly periodic infinite graphs. Tech. Rep. CS-TR-078-87, Princeton University, 1987.

[8] Iwano, K., and Steiglitz, K. Optimization of one-bit full adders embedded in regular structures. *IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-34* (October 1986), 1289–1300.

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

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

[11] Kannan, R., and Bachem, A. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput. 8*, 4 (November 1979), 499–507.

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

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

[14] Orlin, J. B. Some problems in dynamic/periodic graphs. In *Progress in combinatorial optimization* (Orlando, Florida, 1984), W. R. Pulleyblank, Ed., Academic Press, pp. 273–293.

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