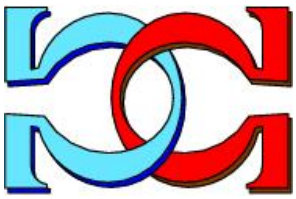
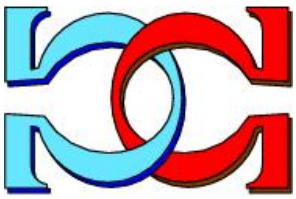
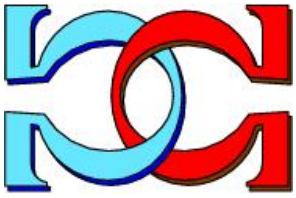


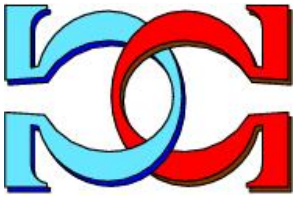
CDMTCS
Research
Report
Series



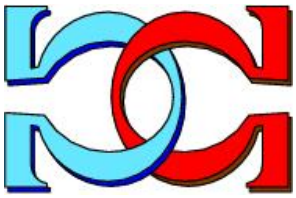
Arithmetic Progression Graphs



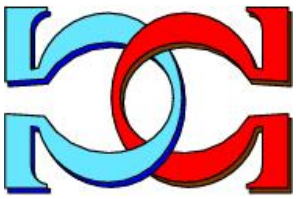
Michael J. Dinneen
Nan Rosemary Ke
Masoud Khosravani



Department of Computer Science,
University of Auckland,
Auckland, New Zealand



CDMTCS-356
March 2009



Centre for Discrete Mathematics and
Theoretical Computer Science

Arithmetic Progression Graphs

Michael J. Dinneen Nan Rosemary Ke
Masoud Khosravani

Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
{mjd,masoud}@cs.auckland.ac.nz
nke001@aucklanduni.ac.nz

Abstract

In this paper we study the problem of labeling the edges of a graph with positive integers such that the sequence of the sums of incident edges of each vertex makes a finite arithmetic progression. We give conditions for paths, cycles, and bipartite graphs to have such a labeling. We then address the opposite problem of finding an edge labeled graph for a given finite arithmetic progression. We use a constructive procedure to fully characterize those finite arithmetic progressions that have representations as edge labeled graphs. Then by presenting a pseudo polynomial-time algorithm, we address a more general problem of finding edge labels for a graph when the vertex labels are given. Finally, we count the connected graphs, up to eight vertices, that accept such a labeling by using a simple algorithm that detects a valid edge labeling.

Key words: graph labeling, arithmetic progression, graph algorithm

1 Introduction

A graph labeling is a mapping from graph elements (vertices, edges, faces, or a combination of them) to a nonempty set. Sometimes it is more convenient to consider the range as a set with a structure, like a ring, a field, or a subset of integers. Also, by restricting the type of mappings to injective, surjective, or bijective maps, and also the range of a mapping, one can define many types of labeling. Gallian's updated survey [Ga09] is a comprehensive reference on the subject, see also [Wa01, Su05]. Among the most studied methods of graph labeling we mention magic labeling [KR70], edge magic labeling [LST92], graceful labeling [Ro66, BS76], (k, d)-arithmetic labeling [AH90], and super edge-antimagic labeling [BM08]. In these types of labeling the mapping and its domain are very restricted and few graphs accept those kinds of labelings. The problem of deciding whether every tree has a graceful labeling is an open problem.

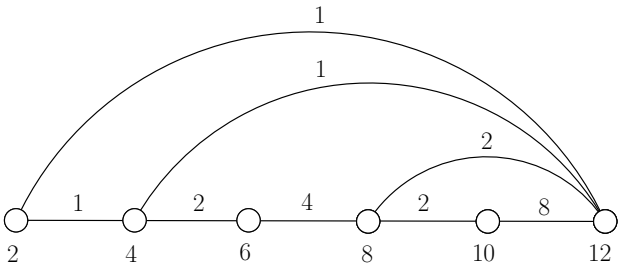


Figure 1: An AP-graph representing the sequence $(2, 4, 6, 8, 10, 12)$.

One problem that arises naturally is how some of the restrictions can be relaxed to allow for more labeled graphs. To achieve this objective, we will consider those edge labelings of a graph that produce arithmetic progressions on the vertices. To be more specific, we study these two problems: (1) Given a graph G, is it possible to assign positive integers to its edges such that the sequence of the sums of neighboring (incident) edges of each vertex makes a finite arithmetic progression? and (2) Given a finite arithmetic progression of positive integers with n terms, is it possible to find an edge labeled graph with n vertices such that the sequence of the sums of incident edges of its vertices produces the same arithmetic progression? As we see later, there are many graphs that are yes instances for the first question, and we refer to them as **Arithmetic Progression Graphs**, or **AP-graphs**. We also refer to the type of edge labeling that produces an AP-graph, as an **AP-labeling**. In Figure 1 a graph with its AP-labeling is depicted. We will also address the second question and give a full characterization of those finite arithmetic progressions that have edge labeled graphs, with respect to our definition.

The paper is organized as follows. In the next section, after listing some basic results, we present those conditions that are necessary and sufficient to test whether a graph is AP-labelable. In Section 3 we present necessary and sufficient conditions for paths and cycles that have AP-labelings; then, a necessary condition is given for bipartite graphs. In Section 4 we study those finite arithmetic progressions that have AP-graphs. To this end, constructive procedures are given that find the appropriate graphs for valid arithmetic progressions. A pseudo-polynomial time algorithm is given in Section 5 that determines whether there is an edge labeling for a graph with a given vertex labeling. We finish the paper by presenting some computational results, as summarized in Table 1, that indicate that most graphs are AP-graphs. The paper ends with a conclusion and some remarks on open problems.

2 Definitions and Basic Results

In this paper we assume that all graphs are simple connected graphs without any loops and multi-edges. We denote the set of the (exclusive) neighbors of a vertex v by $N(v)$. Let $G = (V, E)$ be a graph with n vertices, we say G has an **AP-labeling** (X, a, d), where $X : E \to Z^+$ is a total function, a is the initial value, and d is the constant difference of

the arithmetic progression $a, a+d, ..., a+(n-1)d$ such that for each vertex $v_i \in V(G)$, $0 \leq i \leq n-1$, we have

$$\sum_{u \in N(v_i)} X(uv_i) = a + \sigma(i)d,$$

where σ is a permutation on $\{0, 1, \ldots, n-1\}$. If a graph has such a labeling we say it is an **AP-graph**.

Sometimes we just refer to an edge labeling X on a graph $G(V, E)$ by the list of its values on the set of edges as $X = (x_1, \ldots, x_m)$, where m is the number of edges, and $X(e_i) = x_i$, $e_i \in E$. The following results are trivial consequences of the definition.

Proposition 1 *Let G be a graph of order n and size m. If there is an AP-labeling (X, a, d), such that $X = (x_1, \ldots, x_m)$, then*

$$K = \sum_{i=1}^{m} x_i = \frac{na}{2} + \frac{n(n-1)}{4}d.$$

Proposition 2 *Let G be a graph that has an AP-labeling (X, a, d), then $\delta(G) \leq a$, where $\delta(G)$ is the minimum degree of G.*

Proposition 3 *Let $P = v_1, e_1, v_2, \ldots, e_{n-1}, v_n$ be a path, that has an AP-labeling (X, a, d). Also suppose the labeling is such that $X(e_i) = x_i$, $1 \leq i \leq n-1$. Then d divides $|x_i - x_j|$ if both i and j are even or odd.*

Proposition 4 *Let A be the incidence matrix of a graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Also let (a, d) be a pair of initial value and constant difference in an arithmetic progression. Then G has an AP-labeling (X, a, d) if and only if there is a permutation σ of $\{0, \ldots, n-1\}$ and a linear equation $\mathbf{Ax} = \mathbf{b}$ that has solutions of positive integers, where*

$$\mathbf{b} = \begin{bmatrix} a + d\sigma(0) \\ \vdots \\ a + d\sigma(n-1) \end{bmatrix}.$$

Due to Proposition 1, each AP-labeling also corresponds to an integer partitioning of K (the sum of an edge labeling) into m positive parts, that can be used as a naive algorithm for finding an AP-labeling for a graph. Also Proposition 4 reduces the problem of finding an AP-labeling for a graph to solving a system of linear equations with positive integer values. In Section 6 we will describe our emperical results by applying these algorithms and enumerating all AP-graphs with up to eight vertices.

3 Paths, Cycles and Bipartite Graphs

Note that with respect to Proposition 4, to find an AP-labeling (X, a, d) for a graph we need to choose not only proper values for a and d, but also a proper permutation σ, that distributes terms of the corresponding arithmetic progression among the vertices of the graph. In this section we show that, when dealing with paths and cycles, finding an AP-labeling is equivalent to finding a proper permutation.

Theorem 5 *Let P be a path of length n, then P has an AP-labeling (X, a, d) if and only if there is a permutation σ that satisfies the following conditions.*

1. $\sum_{i=0}^{k-1}(-1)^i\sigma(i) > -\frac{a}{d}$ *if k is odd and $1 \leq k < n$, and*
2. $\sum_{i=0}^{k-1}(-1)^i\sigma(i) > 0$ *if k is even and $2 \leq k < n-1$, and*
3. $\sum_{i=0}^{n-1}(-1)^i\sigma(i) = 0$, *when n is even, or $\sum_{i=0}^{n-1}(-1)^{i+1}\sigma(i) = \frac{a}{d}$, when n is odd.*

Proof. Let $P = v_1, e_1, v_2, e_2, \ldots, v_{n-1}, e_{n-1}, v_n$. If P has an AP-labeling (X, a, d) such that $X = \{x_1, x_2, \ldots, x_{n-1}\}$, where $X(e_i) = x_i$, then we have

$$\begin{cases} x_1 & = & a + \sigma(0)d \\ x_1 + x_2 & = & a + \sigma(1)d \\ & \vdots & \\ x_{n-2} + x_{n-1} & = & a + \sigma(n-2)d \\ x_{n-1} & = & a + \sigma(n-1)d \end{cases}$$

for a permutation σ on $\{0, \ldots, n-1\}$. Now we multiply both sides of the ith equation with $(-1)^{i-1}$. By adding the first k terms, $1 \leq k < n$, for odd values of k we have $x_k = a + d\sum_{i=0}^{k-1}(-1)^i\sigma(i)$.

For even values of k, $2 \leq k < n$, we have $x_k = \sum_{i=0}^{k-1}(-1)^i\sigma(i)$ and since the values of labels are positive integers, we get the desired inequalities of Statements 1 and 2 in the theorem. The same argument applies for Statement 3, by adding all terms.

For proving the sufficient condition note that given a permutation σ, that satisfies the conditions, we can compute the values of the edge labels by

$$x_k = \begin{cases} a + d\sum_{i=0}^{k}(-1)^{i+1}\sigma(i) & k \text{ is even,} \\ d\sum_{i=0}^{k}(-1)^{i+1}\sigma(i) & k \text{ is odd,} \end{cases}$$

and Statements 1–3 ensure us that the values of the labels are positive integers. □

Theorem 6 *Let C be a cycle of length n, then C has an AP-labeling (X, a, d) if and only if there is a permutation σ that satisfies the following conditions.*

1. *If n is odd, $\sum_{i=0}^{k-1}(-1)^{i+k+1}\sigma(i) + \sum_{i=k}^{n-1}(-1)^{i+k}\sigma(i) > \frac{a}{d}$, for $0 \leq k < n$*
2. *If n is even, $\sum_{i=0}^{n-1}(-1)^i\sigma(i) = 0$, and also there is a positive integer c such that $a + d\sum_{i=0}^{k-1}(-1)^i\sigma(i) > c$, for k even, and $c > d\sum_{i=0}^{k-1}(-1)^i\sigma(i)$, for k odd.*

Proof. Let $C = v_1, e_1, v_2, e_2, \ldots, v_{n-1}, e_{n-1}, v_n, e_n, v_1$ be a cycle. If we write the appropriate system of linear equations, we have:

$$\begin{cases} x_1 + x_n & = & a + \sigma(0)d \\ x_1 + x_2 & = & a + \sigma(1)d \\ & \vdots & \\ x_{n-2} + x_{n-1} & = & a + \sigma(n-2)d \\ x_{n-1} + x_n & = & a + \sigma(n-1)d \end{cases}$$

Now, as in the case of paths, if we multiply each ith equation with $(-1)^{i-1}$ and add all the resulting terms, for the case when n is odd we have $x_n = \frac{a}{2} + \frac{d}{2}\sum_{i=0}^{n-1}(-1)^i\sigma(i)$, then by solving the other equations with respect to the value of x_n, for $1 \leq k < n$, we have

$$x_k = \frac{a}{2} + \frac{d}{2}\left(\sum_{i=0}^{k-1}(-1)^{i+k+1}\sigma(i) + \sum_{i=k}^{n-1}(-1)^{i+k}\sigma(i)\right).$$

When n is even, all variable are cancelled and we have $\sum_{i=0}^{n-1}(-1)^i\sigma(i) = 0$. Note that in this case by adding the first k terms when k is odd, $1 \leq k < n$, we have

$$x_n + x_k = a + d\sum_{i=0}^{k-1}(-1)^i\sigma(i),$$

and when k is even, $2 \leq k < n-1$, we have

$$x_n - x_k = d\sum_{i=0}^{k-1}(-1)^i\sigma(i),$$

and since the values of edge labels are positive integers we have when k is odd,

$$x_n < a + d\sum_{i=0}^{k-1}(-1)^i\sigma(i)$$

for $1 \leq k < n$, and also $d\sum_{i=0}^{k-1}(-1)^i\sigma(i) < x_n$ when k is even and $2 \leq k < n-1$. □

Corollary 7 *There is no AP-labeling for paths and cycles of length n, when $n \equiv 2 \pmod 4$.*

As we saw earlier, there are some paths and cycles that do not accept any AP-labeling. By the next theorem, we show it is not restricted to just those type of graphs. There are other bipartite graphs that have the same property. In Section 6 we use the following theorem and its corollary to prove that many of those graphs that are illustrated in Figures 3 and 4 are not AP-graphs.

Theorem 8 *Let G be a bipartite graph with p and q vertices in each part such that $p \geq q$. If G has an AP-labeling (X, a, d), then*

$$\frac{(p-q)(p+q-1)}{pq} < 2.$$

Proof. By constructing the appropriate system of linear equations and multiplying those terms that represent sums of vertices in the part with q vertices by -1, and adding the equations, we have

$$(p-q)a + \left(\sum_{i=0}^{p-1}\sigma(i) - \sum_{j=p}^{p+q-1}\sigma(j)\right)d = 0$$

that results

$$(p-q)\frac{a}{d} = \sum_{i=p+1}^{p+q} \sigma(i) - \sum_{j=1}^{p} \sigma(j),$$

where σ is a permutation on $\{0, 1, \ldots, p+q-1\}$.

If $p = q$ then we have $\sum_{i=p+1}^{p+q} \sigma(i) = \sum_{j=1}^{p} \sigma(j)$, also if $(p-q)\frac{a}{d} \geq 0$, then $\sum_{i=p}^{p+q-1} \sigma(i) \geq \sum_{j=1}^{p-1} \sigma(j)$, both as necessary conditions.

The left hand side of the last inequality takes its largest value when $\sigma_i \in \{0, \ldots, p-1\}$, $0 \leq i \leq p-1$, and $\sigma_j \in \{p, \ldots, p+q-1\}$, $p \leq j \leq p+q-1$. So we have:

$$pq + \frac{q(q-1)}{2} > \frac{p(p-1)}{2}.$$

That yields

$$\frac{(p-q)(p+q-1)}{pq} < 2.$$

□

The following corollary is a result of the proof of the previous theorem.

Corollary 9 *A bipartite graph with n vertices and equal parts has no AP-labeling, when* $n \equiv 2 \,(mod\, 4)$.

As easily seen by Theorem 8 one class of graphs that accepts no AP-labeling is the set of stars $\{K_{1,p} \mid p \geq 3\}$. The next theorem shows that if a graph is close to a star with respect to a few number of edge contractions, then it also has no AP-labeling.

Theorem 10 *A graph G has no AP-labeling if there is a sequence* $e_1, \ldots, e_k$ *of edges in G, such that (i) by contracting them G reduces to a star, where the induced connected subgraph* $H = G[\{e_1, \ldots, e_k\}]$ *is contracted to the center of the star, and (ii)* $k < \frac{2n-1-\sqrt{n^2+(n-1)^2}}{2}$.

Proof. Let $V(G) = \{v_0, v_1, \ldots, v_{n-1}\}$. Also, suppose that there is an AP-labeling (X, a, d) for G that defines a vertex labeling $l_0 : G \to Z^+$ such that $l_0(v_i) = \sum_{u \in N(v_i)} X(v_i u) = a + \sigma(i)d$. We consider $G_0 = G$ and we denote by G_i the resulting graph of the contracting e_i in G_{i-1}, $i = 1, \ldots, k$. After contracting an edge $e_i = xy$ to a vertex v_{xy}, we relabel the edges of G_i by the following rule. If a vertex w was adjacent to both x and y, we replace any multiple edges by a new edge wv_{xy} and we assign the sum of the labels of wx and wy in G_{i-1} to it. The labels of other edges remain unchanged. After iteration i we have a new vertex labeling $l_i : G_i \to Z^+$.

After contracting all edges we reach to a bipartite graph with two parts, part A that has vertices that do not belong to H, and part B that has only the center c of the star G_k. Since each edge has one side in each part we have

$$\sum_{v\in A} l_k(v) = l_k(c) \quad (1)$$

$$\sum_{v_i\in V(G)-V(H)} (a+\sigma(i)d) = \sum_{v_j\in V(H)} (a+\sigma(j)d) - 2\left(\sum_{uv\in E(H)} X(uv)\right) \quad (2)$$

Now note that Equation 2 is valid if $\sum_{v_j\in V(H)} (a+\sigma(j)d)$ is strictly greater than the left hand side so we must have

$$\sum_{v_i\in V(G)-V(H)} (a+\sigma(i)d) < \sum_{v_j\in V(H)} (a+\sigma(j)d)$$

and since, $k < \frac{2n-1-\sqrt{n^2+(n-1)^2}}{2} < \frac{n}{2}$, the number of nodes in the contracted part of graph is less that the number of nodes in the non-contracted part, so we have

$$\sum_{v_i\in V(G)-V(H)} \sigma(i) < \sum_{v_j\in V(H)} \sigma(j)$$

to make right hand side as large as possible we need to find a value $1 \le k < n$ such that we have:

$$\begin{aligned} 1+2+\ldots+(n-k-1) &< (n-k)+\ldots+n-1 \\ 2(n-k)(n-k-1) &< n(n-1) \\ 0 &< -2k^2+(4n+2)k-2(n^2+n). \end{aligned}$$

The statement of the theorem follows by solving the last inequality with respect to the variable k. □

4 Graphs of Finite Arithmetic Progression

Let the sequence $S = (a, a+d, \ldots, a+(n-1)d)$ be a finite arithmetic progression of positive integers with the initial value a and the constant difference d. In this section we try to find out whether there is an AP-graph G with n vertices, such that the set of the sums of neighboring edges of the vertices results in the same sequence as S. To this end, we first consider the case when $a = d$ and we introduce a constructive procedure to make a desired graph for a given sequence. Also we show that there are arithmetic progressions that have no representation as an AP-graph. Later on, we extend our results for the case when $a \neq d$.

Throughout the proofs of the following results we suppose that the vertices of a graph are indexed with respect to the increasing order of their labels. Also we consider the operation of **pairing and incrementing by c** on a sequence of even number of

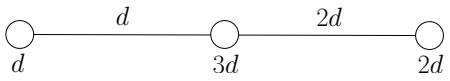


Figure 2: A family of AP-labelings for an AP-graph representing the sequences $\{(d, 2d, 3d) \mid d \geq 1\}$.

vertices $v_s, \ldots, v_{s+k}$, where k is odd. That is done by considering pairs of consecutive vertices (v_i, v_{i+1}), $i \in \{s, s+2, \ldots, s+k-1\}$, and if there is an edge between them, we increment its label by c. When there is no edge, we connect them by an edge and we assign c as its label. Note that by this operation the vertex label of each vertex in $v_s, \ldots, v_{s+k}$, is incremented by c.

Lemma 11 *Let $S = (d, 2d, \ldots, nd)$ be a finite arithmetic progression with $n \geq 3$ terms and with initial value and constant difference both equal to d. Then S has an AP-graph if and only if*

1. *the value of d is even, or*
2. *the value of d is odd, and $n \equiv 0 \pmod 4$ or $n \equiv 3 \pmod 4$.*

Proof. Let us first consider the case when d is even. If S has three terms, say $S = (d, 2d, 3d)$, the graph shown in Figure 2 is an appropriate AP-graph. Now we follow the argument inductively by considering two cases, when the number of terms is odd or even.

Suppose we have an AP-graph G for $S = (d, \ldots, (n-1)d)$. If n is even then we do the pairing and incrementing by d operation for the sequence of vertices $v_2, \ldots, v_{n-1}$ in G. Then we add a new vertex u and connect it to v_1 by an edge labeled by d. The resulting graph is an AP-graph for the sequence $(d, 2d, \ldots, nd)$.

For the case when n is odd, we do the pairing and incrementing operation by d for the sequence of vertices $v_3, \ldots, v_{n-1}$ in G. Then we add a new vertex u and we connect it to v_1 and v_2 by two edges, each has $\frac{d}{2}$ as its label. At last, if there is an edge between v_1 and v_2 we increment it by $\frac{d}{2}$, otherwise we connect them by a new edge that is labeled by $\frac{d}{2}$.

Now we prove that for odd value of d and $n \equiv 3 \pmod 4$, if there is an AP-graph G for S with $n-4$ terms, then there is an AP-graph for S with n terms. The base case when $n = 3$ follows easily by considering the graph illustrated in Figure 2, so we suppose that $n \geq 7$. We first employ the pairing and incrementing by $4d$ on the sequence of vertices $v_2, \ldots, v_{n-4}$ in G, such that the label of each vertex in this sequence increases by $4d$. Then we add four new vertices u_1, u_2, u_3, and u_4. Finally, we add an edge from u_1 to u_4 labeled by d, an edge from u_2 to u_3 labeled by $2d$, an edge from u_3 to v_1 labeled by $2d$, and an edge from u_4 to v_1 labeled by $2d$.

Now let consider the case when d is odd and $n \equiv 0 \pmod 4$. Since $n-1 \equiv 3 \pmod 4$, there is an AP-graph G for the sequence $(d, 2d, \ldots, (n-1)d)$. We apply the operation of pairing and incrementing by d for the sequence of vertices $v_2, \ldots, v_{n-1}$ in G. By adding a new vertex u and connecting it to v_1 with an edge labeled by d, we get the desired graph.

To prove that there is no AP-graph when d is odd and $n \equiv 1, 2 \pmod 4$. Let (X, d, d) be an AP-labeling of a graph G with the edge labels $X = (x_1, x_2, \ldots, x_m)$. Then due to Proposition 1 the value of

$$\sum_{i=1}^{m} x_i = \frac{dn}{2} + \frac{n(n-1)}{4}d,$$

must be an integer. But when one computes the right hand side of the equation for those values of d and n, the resulting values are not integers. □

Theorem 12 *Let $S = (a, a+d, \ldots, a+(n-1)d)$ be a finite arithmetic progression with the initial value a, the constant difference d, and $n \geq 4$ terms. Then S has an AP-graph if and only if*

1. *the value of d and a are both even, or*
2. *d is even, a is odd, and n is even, or*
3. *d is odd, a is even, and $n \equiv 0, 1 \pmod 4$, or*
4. *d is odd, a is odd, and $n \equiv 0, 3 \pmod 4$.*

Proof. To prove this theorem we first apply Lemma 11 to produce an AP-graph G with $n-1$ vertices for $S' = (d, 2d, \ldots, (n-1)d)$, for valid numbers of d and n. We use two procedures to construct an AP-graph from G for S based on the number of terms in S'. The first procedure is used when the number of terms in S' is odd. It is done by pairing and incrementing operation by a for the sequence of $v_2 \ldots, v_{n-1}$, and then adding a new vertex and joining it to v_1 with an edge labeled by a. The second procedure is used when the number of terms in S' and a are both even. In this procedure we apply the operation of pairing and incrementing by a on $v_3, \ldots, v_{n-1}$, then we add a new vertex and attach it to v_1 and v_2 by edges that are labeled by $\frac{a}{2}$. Finally, if there is an edge between v_1 and v_2 we increment it by $\frac{a}{2}$, otherwise we join them by a new edge labeled by $\frac{a}{2}$. These procedures, when applied properly, give the proofs for Statements 1–3 and the part of Statement 4 when $n \equiv 0 \pmod 4$.

Now let us consider the case where d and a are odd and $n \equiv 3 \pmod 4$. Since $n \equiv 3 \pmod 4$, we have $n - 3 \equiv 0 \pmod 4$. So by Lemma 11 there is an AP-graph G for the sequence $(d, 2d, \ldots, (n-3)d)$. We first apply a pairing and incrementing by $a + 2d$ for the sequence $v_3, \ldots, v_{n-3}$. Then we add three new vertices u_1, u_2, and u_3 to the resulting graph. To complete the construction, we add two edges labeled by $\frac{a+d}{2}$ from u_3 to both v_1 and v_2. Also we connect u_1 to u_2 by an edge labeled by a, and u_2 to u_3 by an edge labeled by d. At last if there is any edge between v_1 and v_2 we increment it by $\frac{a+3d}{2}$, otherwise we add them via a new edge labeled by $\frac{a+3d}{2}$.

With respect to Proposition 1, the same argument as presented for Lemma 11 also applies to the no instances in this theorem. □

To complete the characterization of those arithmetic progression sequences that are vertex labels for some AP-graph, first note that there are no sequences of length less than three. Also, for $n = 3$ there are only two graphs to consider. It is easy to see that the path (see Figure 2) must have $a = d$. For the complete graph K_3 we get only those sequences satisfying $a > d$ and $a + d$ even.

5 Decision algorithms

In this part we present two complexity results on edge labeling of a graph that is closely related to the problem of deciding if a graph has an AP-labeling. We first consider the problem of finding edges labels for a graph such that the sums of the edge labels incident to each vertex yield a specified target set of vertex labels. We show that the problem is NP-complete by reducing a restricted version of the partition problem into two equal-sized partitions, denoted here by EqualPartition, (see [SP12] in [GJ79]).

Problem 13. FreeIncidentEdgeLabelable
Input: Graph $G = (V, E)$ and a (multi-)set $A = \{a_1, \ldots, a_{|V|}\}$
Question: Does there exist a bijection $f : V \to A$ and an edge labeling $h : E \to Z^+$ such that for all $v \in V$, $f(v) = \sum_{u \in N(v)} h(uv)$

Theorem 14 *The problem* FreeIncidentEdgeLablable *is NP-complete.*

Proof. It is easy seen that the problem is NP. To show NP-hardness of the problem we reduce from EqualPartition. Let $\{s_1, s_2, \ldots, s_n\}$ be an instance of EqualPartition. If either n or $S' = \sum_{i=1}^{n} s_i$ is odd then we map to a known no instance of FreeIncidentEdgeLabelable, say $G = K_2$ and $A = \{1, 2\}$. Otherwise, we transform it to a graph G with $n + 2$ vertices created by adding an edge between the centers of two stars $K_{1,n/2}$ and set $A = \{s_1, \ldots, s_n, S', S'\}$. Clearly, if the graph G can be edge labeled by some h, yielding vertex labels $A = \{f(1), \ldots, f(n+2)\}$, then the two center vertices must end up with the two labels S'. The only way to do this is for the connecting edge to have label $S'/2$ and the edges incident to the pendant vertices having labels that partition $\{s_1, \ldots, s_n\}$ into two parts of size $n/2$, each part summing up to $S'/2$. □

Our next result shows that by assigning fixed values to vertices, finding a proper edge labeling that produces that vertex labeling is almost polynomial-time solvable.

Problem 15. FixedIncidentEdgeLabelable
Input: Connected undirected graph $G = (V, E)$ with a vertex labeling $f : V \to Z^+$.
Question: Does there exist an edge labeling $g : E \to Z^+$ such that for all $v \in V$, $f(v) = \sum_{u \in N(v)} g(uv)$?

Theorem 16 *There exists a pseudo polynomial-time algorithm to decide the problem* FixedIncidentEdgeLabelable.

Proof. We give a decision algorithm that runs in time that is bounded by a polynomial of $n = |V|$ and the sum $T = \sum_{v \in V} f(v)$.

The idea for the algorithm is to start with an initial assignment of edge weights and gradually increase them until the incident edges sum up to the desired vertex labels. We say a vertex is **unsaturated** if the sum of its incident edges is strictly less than its label. We use an approach similar to the augmenting path techniques used in many maximal matching algorithms. If there is an odd length sequence of edges, starting and ending at a unsaturated vertices (which could be the same vertex), then we can improve the saturation level of the system by adding, in an alternating fashion, +1 and -1 to the edges in this walk sequence. Here only the saturation levels of the first and last vertices increase by one. To ensure that each edge label stays positive we only apply this augmentation when all of the even-indexed edges have value at least 2. We call such a path an **improving walk**. The details of the algorithm are sketched in the procedure below.

```
    Procedure Membership(Graph G, Vertex labels f[1..n])
    begin
        Edge labels g[1..m] = (1, 1, ..., 1)
        Saturation values s[v] = Σ_{uv∈E} g[uv], for all v ∈ V
        if ∃v, s[v] > f[v] return false
L1:     while ∃v, s[v] ≠ f[v] do
            for each v such that s[v] ≠ f[v] do
                if there exists an improving walk starting at v do
                    # Note special case of ending at v if s[v] + 1 = f[v]
                    Augment the edges of the improving walk by
                    updating the values of g[] and s[]
                    next L1
            return false
        return true
    end
```

The program will terminate with at most $T/2$ iterations of the while loop at line L1 since the sum of saturation levels always increases by two. To find an improving walk we need to do a graph traversal using breadth-first search. We need to consider whether each of the other vertices is reachable at only an odd or even distance (not all the distances) from the starting vertex. Thus the search tree size is bounded by $2n$, and this can be computed in $O(n^2) = O(m)$ steps.

For correctness of the algorithm, if we saturate all of the vertices then we clearly have found a desired edge labeling (when the while loop at L1 terminates).

There is a special case where there may be an improving walk (circuit) from v to itself but $s[v]+2 > f[v]$. Consider that we could have augmented this circuit (i.e. increase $s[v]$ by 2) then decrease another incident edge vw of v by 1 to get $s[v] = f[v]$. This would mean that another unsaturated vertex y is the start of an improving path that ends at w. Thus we only do this special case if we can find a replacement improving path from y to w.

Now suppose the procedure can not find an improving walk from an unsaturated vertex v. This means that we have created a reachability tree from v (of all saturated vertices except v) where each edge e at even level has $g[e] \geq 2$. Each leaf u of this tree must either have all its neighbors already in the tree, or u is an odd distance from v and the edge label is 1 to a vertex not in the tree. By contradiction, assume the graph has an edge labelable g' such that each vertex is saturated but the algorithm halts with unsaturated vertices, given by g. Consider the edges incident to an unsaturated vertex u_0. There must be an edge u_0u_1 such that $g[u_0u_1] < g'[u_0u_1]$. We will show that there exists an improvable walk in the graph (not necessarily starting at u_0). Increase $g[u_0u_1]$ by one. Vertex u_1 must have been saturated before the increase (otherwise we would have an improvable walk of length 1). Thus since u_1 is (now) oversaturated there must be an edge u_1u_2 such that $g'[u_1u_2] < g[u_1u_2]$. Decrease $g[u_1u_2]$ by one. Then the process of incrementing/decrementing values of $g[u_iu_{i+1}]$ continues. Since the relabeling of g is converging to g' we must eventually reach an unsaturated vertex u_{i+1}. If $i+1$ is odd then we found an improving walk. Otherwise, start a new walk at the unsaturated vertex u_{i+1} and continue the process of converting g to g'. Since we assumed the graph is edge labelable and $\sum_{e \in E} |g'[e] - g[e]|$ is bounded we eventually have to increase the value of an unsaturated vertex u_{2k+1}. This only happens if we have an odd length walk (or discover an improving walk $u_{2(k-j)}, \ldots, u_{2k}, u_{2k+1}$, $0 \leq j \leq k$). Thus, our assumption that the algorithm halts is wrong. Thus, the algorithm will find some labeling if one exists. □

Note the running-time of the algorithm given in the previous proof of Theorem 16 should be implementable in time $O(n^3T)$. Thus if the maximum vertex label is bounded by some constant k then we have an algorithm that is fixed-parameter tractable via an $O(kn^4)$ time algorithm. To solve our AP-labeling problem we have to apply this algorithm $n!$ times, once for each permutation of $\{a, a+d, a+2d, \ldots, a+(n-1)d\}$ assigned as vertex labels. This running time is theoretically better than both our brute-force integer partition algorithm (Proposition 1) and integer programming algorithm (Proposition 4).

6 Experimental Results

In this section, as a part of our study, we are concerned with the problem of classifying all small graphs (with order at most eight) as having an AP-labeling or not. Our computational results are summerized in Table 1. Drawings of the smaller set of no instances

Table 1: The number of AP-graphs for connected graphs up to eight vertices.

Order n	1	2	3	4	5	6	7	8
Yes	0	0	2	5	20	101	849	11087
No	1	1	0	1	1	11	4	30

are shown in Figures 3–5. To achieve this goal, we implemented (using in Sage [St+09] libraries) the two simple brute-force algorithms based on the results of Propositions 1 and 4. When the number of edges m was at most the number of vertices n we used the simple algorithm based on Proposition 1, that is, we generated all integer partitions of K into m parts. However, in most cases, the size m was larger than the order n and we reduced our problem to an equivalent integer programming problem and called the efficient Sage IP solver for n linear equations. In both cases, we considered a wide range of the values of a and d and tried all permutations of its edges or vertices, respectively.

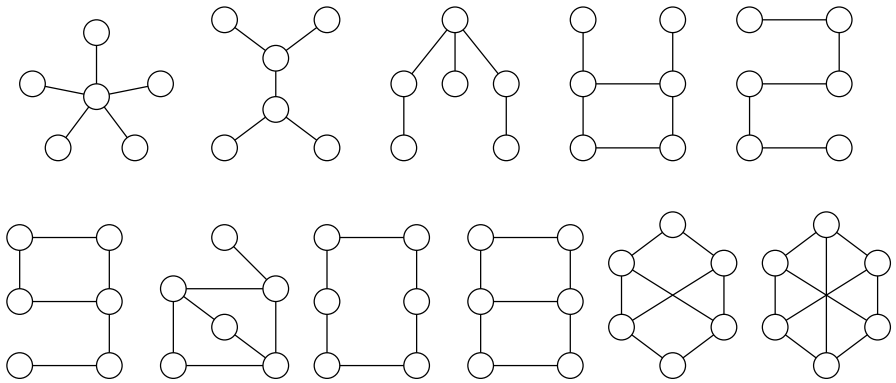


Figure 3: Connected graphs with six vertices that have no AP-labelings.

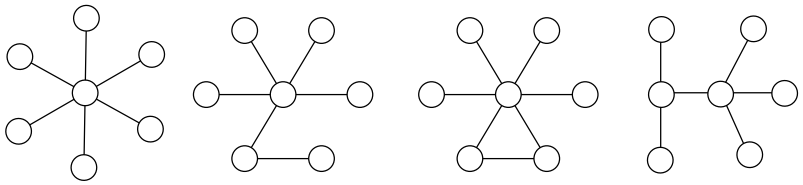


Figure 4: Connected graphs with seven vertices that have no AP-labelings.

We used geng program [Mc08] to produce all simple connected graphs up to eight vertices. For each graph, if the invoked algorithm finds an AP-labeling, then it saves the result and terminates. For example, Figure 6 displays AP-labelings for each of the AP-graphs with four vertices. Both our Sage programs and the first found AP-labelings for the yes instances of order 5 are available in the appendices. In addition, the yes instances of the AP-graphs with orders 6–8 are available from the authors, on request.

Note that even if the algorithm fails in finding any solution for a given graph, it does not guarantee the nonexistence of an AP-labeling (since we did not try all combinations of a and d). So we need a formal proof for the no instances. Fortunately, all of these remaining cases are easily confirmed as not having an AP-labeling from our theoretical results of Section 3, as discussed below.

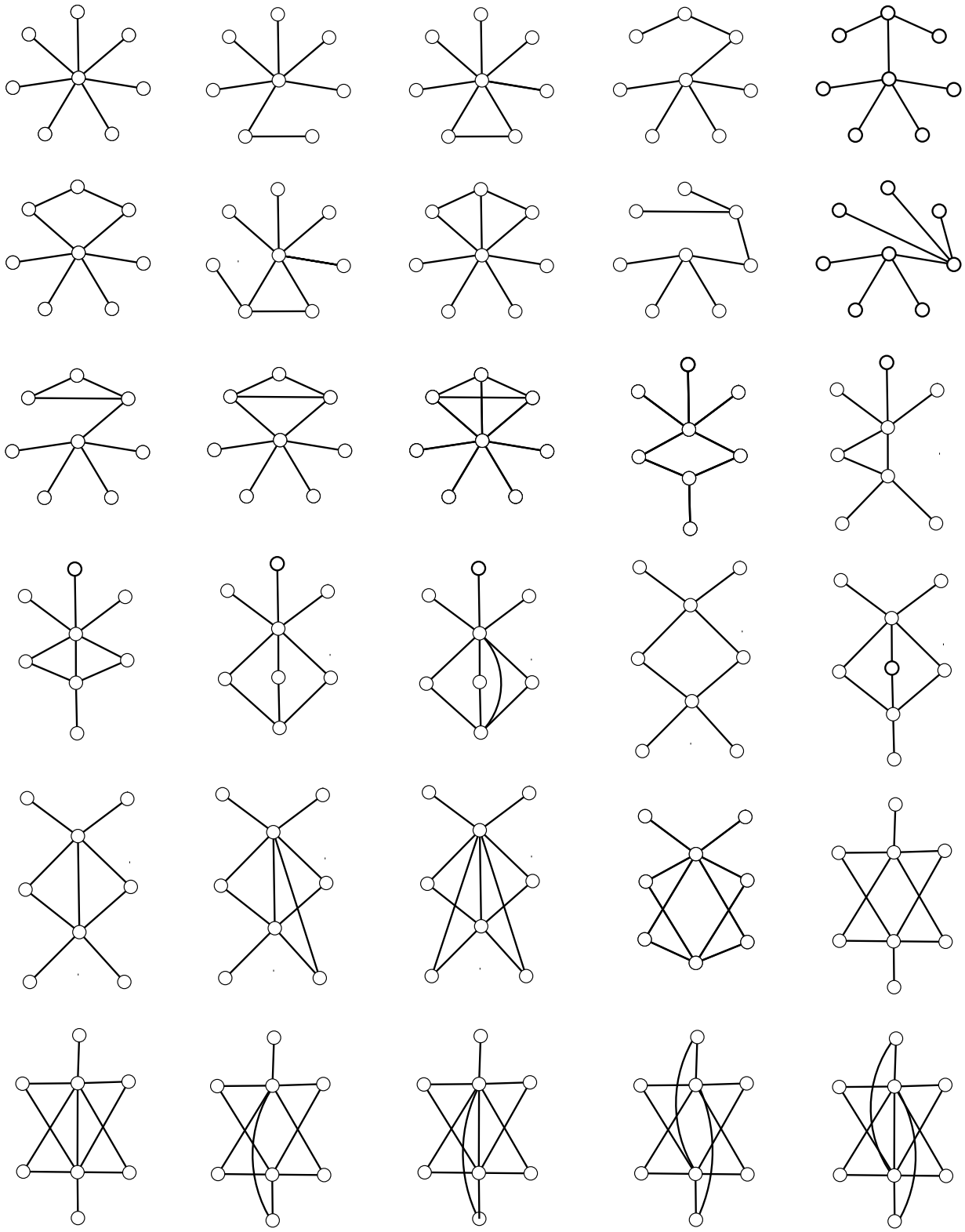


Figure 5: Connected graphs with eight vertices that have no AP-labeling.

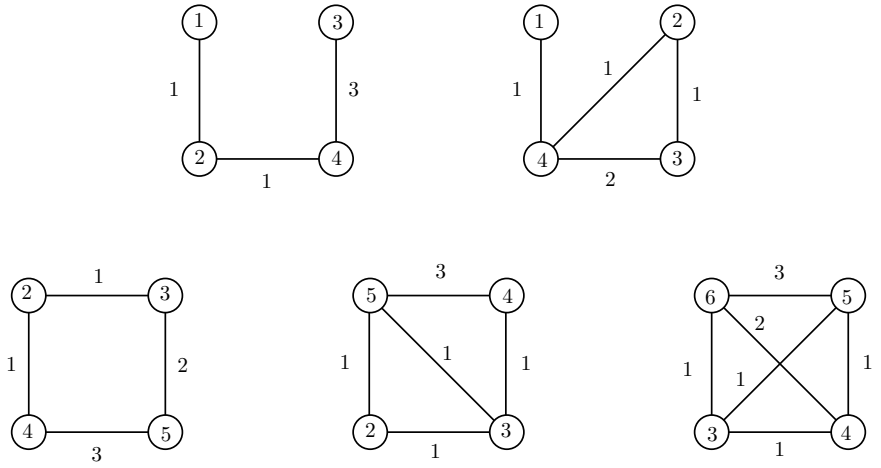


Figure 6: The AP-graphs with four vertices, with AP-labelings ($a = \delta(G), d = 1$).

Note that by Theorem 8 we know that any star with more than four vertices is not an AP-graph; this applies to the three stars in the figures. The other graphs in Figure 3 have six vertices and are bipartite and as they have parts with equal size, by Corollary 9 we know that they do not accept any AP-labelings. To prove that the remaining three graphs in Figure 4 are not AP-graphs we use Theorem 10. We see, in each case, that only one edge (i.e. $k = 1$) needs to be contracted to produce the star $K_{1,5}$ allowing us to use Theorem 10.

Likewise, the fact that none of the graphs in Figure 5 is an AP-graph follows from Theorem 8 and (using $k \leq 2$) Theorem 10.

7 Conclusions

In this paper we have introduced a new graph labeling problem to decide whether a graph's vertices can be labeled as an arithmetic progression (under constraints of being induced from edge labelings by positive integers). We have developed three different algorithms: a simple brute-force integer partition algorithm, a mapping to an instance of integer programming, and, one based augmenting improvement of vertex saturations. By experimental search, we have discovered that most connected graphs are AP-graphs. It is observed that most of the 'no'-cases are sparse graphs and star-like. In the other direction, we have characterized all arithmetic sequences that are representable by some AP-graphs.

There are several open questions related to our work in this new area. First, we would like to find a polynomial-time algorithm or (more likely) a proof that the AP-graph decision problem is NP-complete. For a given graph we know that we can scale an edge-labeling (X, a, d) to get another $(X', k \cdot a, k \cdot d)$ labeling by simply setting $X'(e) = k \cdot X(e), e \in E(G)$. However, we would like to know how many non-equivalent labelings a graph has, which would be a nice characteristic of the structure of a graph. In particular, what are the smallest a and d that need to be checked for a graph of order n and m.

Currently, we do not know of an upper bound for the smallest a and d and need to either get lucky with a guess or prove (by hand or using the limited structural constraints of Section 3) that no such arithmetic progression exists.

References

[AH90] B. D. Acharya and S. M. Hedge, Arithmetic graphs, J. Graph Thoery, 14, 175-299, 1990.

[BM08] M. Bača and M. Miller, Super Edge-Antimagic Graphs, BrownWalker Press, Boca Raton, 2008.

[BS76] G. S. Bloom and S. W. Golomb, Numbered complete graphs, unusual rulers, and assorted applications, Theory and applications of graphs (Proc. Internat. Conf., Western Mich. Univ., Kalamazoo, Mich., 1976), Lecture Notes in Math., Vol. 642, Springer, Berlin, 53–65, 1978.

[Ga09] J. A. Gallian, A dynamic survey of graph labeling. Electron. J. Combin. 16, 2009.

[GJ79] M. R. Garey and D. S. Johnson, Computers and intractability, A guide to the theory of NP-completeness, W. H. Freeman and Co., San Francisco, Calif., 1979.

[KR70] A. Kotzig and A. Rosa, Magic valuation of finite graphs Canad. Math. Bull. 13, 451-461, 1970.

[LST92] S.-M. Lee, E. Seah, and S.-K.Tan, On edge magic graphs, Congresus Num, 86, 179-191, 1992.

[Mc08] B. McKay, nauty software, version 2.4, 2008, `http://cs.anu.edu.au/~bdm/nauty`

[Ro66] A. Rosa, On certain valuations of the vertices of a graph, Theory of Graphs (Internat. Sympos., Rome, 1966), Gordon and Breach, New York, 349–355, 1967.

[St+09] W. A. Stein et al., *Sage Mathematics Software (Version 3.3)*, The Sage Development Team, 2009, `http://www.sagemath.org`.

[Su05] K. A. Sugeng, Magic and antimagic labeling of graphs. Phd. Thesis, University of Ballarat, Ballarat, Australia, 2005.

[Wa01] W. D. Wallis, Magic Graphs, Birkhäuser Boston, Boston, MA, 2001.

A Three Simple Programs that Check for AP-labelings

First we give common code (python/sage) to read in a graph (networkx format) and extract arithmetic progression parameters a and d from the command-line arguments is given.

```
import networkx as nx
from sage.all import *

a=1; d=1
if len(sys.argv)>1: a=int(sys.argv[1])  # command-line args
if len(sys.argv)>2: d=int(sys.argv[2])

def magic(L):  # check if the sorted L is an arith. prog.
  for i in range(len(L)-1):
    if L[i+1]-L[i]!=d: return 0
  return 1

g=nx.read_adjlist(sys.stdin)  # input graph from networkx
n=g.order(); m=g.size()
print "Graph", [(int(u),int(v)) for (u,v) in g.edges()]
```

Our first sage algorithm to decide (via combinatorial integer partition search) if a graph is AP-labelable is given next.

```
s=(2*n*a+(n-1)*n*d)/4      # desired sum of edges

for x in partitions_greatest_eq(s,m):
  for y in permutations(partition_associated(x)):
     label=[0]*n; i=0
     for (u,v) in g.edges():
        label[int(u)]+=y[i]; label[int(v)]+=y[i]; i+=1
     if magic(sorted(label)): print label, y; sys.exit(0)
```

A more practical algorithm to decide (via Integer Programming) if a graph is AP-labelable is presented next.

```
for p in permutations(n):
    B = [a+(p[i]-1)*d for i in range(n)]
    p=MixedIntegerLinearProgram()
    b=p.new_variable()
    for i in range(n):
        p.add_constraint(sum([b[(u,v)] for (u,v) in g.edges(labels=None) \
            if i==int(u) or i==int(v)]),min=B[i],max=B[i])
    for e in g.edges(labels=None):
        p.add_constraint(b[e],min=1)
```

```
    p.set_objective(sum([b[x] for x in g.edges(labels=None)]))
    p.set_integer(b)
    try:
        p.solve(solver="Coin")
    except sage.numerical.mip.MIPSolverException as e:
        pass
    else:
        print p.get_values(b).items(); sys.exit(0)
```

Below is an alternative, but slightly slower, IP algorithm using the Singular.

```
M=Graph(g).incidence_matrix()
for r in range(0,n):
   for t in range(0,m): M[r,t]=abs(M[r,t]) # make undirected

singular.LIB("intprog.lib")
singular.eval('intmat M[%s][%s]=%s'%(n,m,str(M.list())[1:-1]))
I=vector(ZZ,[1]*m); M2=singular("M");
C2=singular(('0,'*m)[:-1],'intvec')

for p in permutations(n):
    B=vector(ZZ,[a+(p[i]-1)*d for i in range(n)])
    E=B-(M*I)
    if min(E)<0: continue
    E2=singular(str(E.list())[1:-1],'intvec')
    N2=singular.solve_IP(M2,E2,C2,'"pct"')
    if N2: print 'vertices =',B, 'edges =',list(N2+I); sys.exit(0)
```

Finally, we present our fastest algorithm based on improving walks.

```
import networkx as nx
import numpy as np

n=0                  # order of input graph
STree=nx.DiGraph()   # bipartite search graph
unsat=set()          # current nodes not saturated

def set_search_tree(G,g):
    global STree
    STree=nx.DiGraph()
    STree.add_nodes_from(range(2*n))
    for x in range(n):
        for y in G[x]:
            STree.add_edge(x,y+n)
            if g[min(x,y),max(x,y)]>1: STree.add_edge(y+n,x)
    return
```

```
def improve_walk(v,LoopFlag):
    path=nx.single_source_shortest_path(STree,v)
    endpoint=unsat
    if LoopFlag: endpoint=unsat|set([v])
    for w in endpoint:
        if w+n in path: return [z%n for z in path[w+n]]
    return None

def membership(G,f):
    """
      See if connected graph G can have an edge labeling (g[]) such that
      the sum of incident edge labels is the value corresponding to the
      f[] vertex labeling.
    """
    global n, unsat
    n=G.order()
    g=np.zeros((n,n),int)              # current edge labels (adjacency matrix)
    for (u,v) in G.edges(): g[u,v]=1
    s=np.zeros((n),int)                # current saturation values (vector)
    for v in G:
        s[v]=len(G[v])
        if s[v]>f[v]: return None
    unsat=set([i for i in range(n) if s[i]<f[i]])
    while unsat:                       # line L1 of pseudocode on page 11
        set_search_tree(G,g)
        v=unsat.pop()
        W=improve_walk(v,s[v]!=f[v]-1) # 2nd argument detects our "special case"
        if W:
            for i in range(len(W)-1):  # augment the edges g[] of the improving walk
                if W[i]<W[i+1]: g[W[i],W[i+1]]+=2*((i+1)%2)-1
                else:           g[W[i+1],W[i]]+=2*((i+1)%2)-1
            w=W[-1]; s[v]+=1; s[w]+=1  # and update the saturations s[]
            if s[v]!=f[v]: unsat.add(v)
            if v!=w and s[w]==f[w]: unsat.remove(w)
        else: return None
    return [((u,v),g[u,v]) for (u,v) in G.edges()]
```

B AP-labelings for AP-graphs of Order 5

On the next two pages the twenty AP-graphs with 5 vertices are listed (in geng enumeration order) as incident matrices and AP-labelings with $a = 6$ and small d.

```
geng -c 5 -p2
[0 1 1 0]
[1 0 0 0]
[0 0 0 1]
[0 1 0 0]
[1 0 1 1]
vertices = (15, 6, 9, 12, 18)
edges = [6, 12, 3, 9]
```

```
geng -c 5 -p3
[0 1 1 0 0]
[1 0 0 0 0]
[0 0 0 0 1]
[0 1 0 1 0]
[1 0 1 1 1]
vertices = (14, 6, 10, 18, 22)
edges = [6, 13, 1, 5, 10]
```

```
geng -c 5 -p4
[0 0 1 1 0]
[1 1 0 0 0]
[0 0 0 0 1]
[1 0 1 0 0]
[0 1 0 1 1]
vertices = (6, 9, 15, 12, 18)
edges = [7, 2, 5, 1, 15]
```

```
geng -c 5 -p5
[0 1 1 0 0]
[1 0 0 0 0]
[0 0 0 0 1]
[1 1 0 1 0]
[0 0 1 1 1]
vertices = (6, 8, 10, 12, 14)
edges = [8, 3, 3, 1, 10]
```

```
geng -c 5 -p6
[0 0 1 1 0 0]
[1 1 0 0 0 0]
[0 0 0 0 0 1]
[1 0 1 0 1 0]
[0 1 0 1 1 1]
vertices = (6, 8, 10, 12, 14)
edges = [6, 2, 5, 1, 1, 10]
```

```
geng -c 5 -p7
[0 0 1 1 0 0]
[1 1 0 0 0 0]
[0 0 0 0 1 1]
[1 0 1 0 1 0]
[0 1 0 1 0 1]
vertices = (6, 9, 15, 12, 18)
edges = [1, 8, 1, 5, 10, 5]
```

```
geng -c 5 -p8
[0 0 1 1 0 0 0]
[1 1 0 0 0 0 0]
[0 0 0 0 1 0 1]
[1 0 1 0 1 1 0]
[0 1 0 1 0 1 1]
vertices = (6, 8, 10, 12, 14)
edges = [1, 7, 1, 5, 9, 1, 1]
```

```
geng -c 5 -p9
[0 0 1 1]
[1 1 0 0]
[0 0 1 0]
[1 0 0 0]
[0 1 0 1]
vertices = (12, 18, 6, 9, 15)
edges = [9, 9, 6, 6]
```

```
geng -c 5 -p10
[0 0 1 1 0]
[1 1 0 0 0]
[0 0 1 0 1]
[1 0 0 0 0]
[0 1 0 1 1]
vertices = (6, 8, 9, 7, 10)
edges = [7, 1, 3, 3, 6]
```

```
geng -c 5 -p11
[0 0 1 1 0 0]
[1 1 0 0 0 0]
[0 0 1 0 0 1]
[1 0 0 0 1 0]
[0 1 0 1 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [5, 2, 5, 1, 4, 3]
```

```
geng -c 5 -p12
[0 0 1 1 0]
[1 1 0 0 0]
[0 0 0 1 1]
[1 0 1 0 0]
[0 1 0 0 1]
vertices = (6, 7, 8, 9, 10)
edges = [4, 3, 5, 1, 7]
```

```
geng -c 5 -p13
[0 0 1 1 1 0]
[1 1 0 0 0 0]
[0 0 0 1 0 1]
[1 0 1 0 0 0]
[0 1 0 0 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [5, 2, 4, 1, 1, 7]
```

```
geng -c 5 -p14
[0 0 1 1 1 0 0]
[1 1 0 0 0 0 0]
[0 0 0 1 0 0 1]
[1 0 1 0 0 1 0]
[0 1 0 0 1 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [5, 2, 1, 4, 1, 3, 4]
```

```
geng -c 5 -p15
[0 1 1 1 0 0]
[1 0 0 0 0 0]
[0 0 1 0 1 1]
[0 1 0 0 1 0]
[1 0 0 1 0 1]
vertices = (6, 7, 8, 9, 10)
edges = [7, 3, 1, 2, 6, 1]
```

```
geng -c 5 -p16
[0 1 1 1 0 0 0]
[1 0 0 0 0 0 0]
[0 0 1 0 1 0 1]
[0 1 0 0 1 1 0]
[1 0 0 1 0 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [7, 3, 2, 1, 5, 1, 1]
```

```
geng -c 5 -p17
[0 0 1 1 1 0 0 0]
[1 1 0 0 0 0 0 0]
[0 0 0 1 0 1 0 1]
[1 0 1 0 0 1 1 0]
[0 1 0 0 1 0 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [1, 6, 2, 2, 2, 5, 1, 1]
```

```
geng -c 5 -p18
[0 0 0 1 1 1 0]
[1 1 1 0 0 0 0]
[0 1 0 0 1 0 1]
[1 0 0 1 0 0 0]
[0 0 1 0 0 1 1]
vertices = (6, 7, 9, 8, 10)
edges = [4, 1, 2, 4, 1, 1, 7]
```

```
geng -c 5 -p19
[0 0 0 1 1 1 0 0]
[1 1 1 0 0 0 0 0]
[0 1 0 0 1 0 0 1]
[1 0 0 1 0 0 1 0]
[0 0 1 0 0 1 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [1, 4, 2, 2, 3, 1, 6, 1]
```

```
geng -c 5 -p20
[0 0 0 1 1 1 0 0 0]
[1 1 1 0 0 0 0 0 0]
[0 1 0 0 1 0 1 0 1]
[1 0 0 1 0 0 1 1 0]
[0 0 1 0 0 1 0 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [1, 1, 5, 2, 1, 3, 5, 1, 1]
```

```
geng -c 5 -p21
[1 0 0 0 1 1 1 0 0 0]
[1 1 1 1 0 0 0 0 0 0]
[0 0 1 0 0 1 0 1 0 1]
[0 1 0 0 1 0 0 1 1 0]
[0 0 0 1 0 0 1 0 1 1]
vertices = (6, 7, 8, 9, 10)
edges = [1, 1, 1, 4, 1, 1, 3, 5, 2, 1]
```