ELSEVIER

Information Processing Letters 51 (1994) 133–140

Information Processing Letters

# Approximations for the maximum acyclic subgraph problem

Refael Hassin *, Shlomi Rubinstein

*Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel*

Communicated by S. Zaks; received 9 May 1993; revised 9 February 1994

**Abstract**

Given a directed graph $G = (V, A)$, the maximum acyclic subgraph problem is to compute a subset, $A'$, of arcs of maximum size or total weight so that $G' = (V, A')$ is acyclic. We discuss several approximation algorithms for this problem. Our main result is an $O(|A| + d_{\max}^3)$ algorithm that produces a solution with at least a fraction $1/2 + \Omega(1/\sqrt{d_{\max}})$ of the number of arcs in an optimal solution. Here, $d_{\max}$ is the maximum vertex degree in $G$.

*Keywords:* Analysis of algorithms; Combinatorial problems

## 1. Introduction

Given a directed graph $G = (V, A)$, $V = \{1, \ldots, n\}$, with arc weights $w_{ij} > 0$, $(i, j) \in A$, the *maximum acyclic subgraph problem* is to find a subset $A' \subset A$ such that $G' = (V, A')$ is acyclic and $w(A') = \sum_{(i,j)\in A'} w_{ij}$ is maximized. An alternative statement of this problem (the *minimum feedback arc set* problem) requires to find a minimum weight subset $A'' \subset A$ such that every (directed) cycle of $G$ contains at least one arc in $A''$. The problem is NP hard [11]. It belongs to the class of "edge deletion problems" [17,21]. It has been shown to be complete for the class of permutation optimization problems, MAX SNP[$\pi$], defined in [19], that can be approximated within a fixed error ratio.

The problem is polynomially solvable when $G$ is planar ([4,12,15] and Chapter 8.4 in [8]). The best complexity of these algorithms is $O(n^3)$ [5] and $O(n^{5/2}\log(nW))$ where $W$ is the largest magnitude of an arc weight (and the weights are assumed to be integral) [6]. The problem is also polynomially solvable for the more general class of $K_{3,3}$-free graphs [18] and the classes of reducible flow graphs [20] and weakly acyclic graphs [7]. A variation of the problem in which the objective is to minimize the greatest outdegree of a vertex in the subgraph $(V, A'')$ can be solved in linear time [16].

The problem has a variety of applications such as ordering alternatives by group voting, determining of a hierarchy of the sectors of an economy, determining ancestry relationships, analysis of systems with feedback, and certain scheduling problems [3,14,20]. Flood [3]

* Corresponding author. Email: hassin@math.tau.ac.il.


*SSDI* 0020-0190(94)00086-E

used the relation of the problem to quadratic assignment for developing an efficient branch-and-bound algorithm. Jünger [14] studied the acyclic subgraph polytope.

The maximum acyclic subgraph and minimum feedback arc set problems are equivalent with respect to their optimal solution. However, bounded error polynomial approximations are known only for the maximum acyclic subgraph version. The simplest algorithm is the following [9]: Let $A_1 = \{(i,j) \in A \mid i < j\}$, $A_2 = \{(i,j) \in A \mid i > j\}$. Clearly, both $(V, A_1)$ and $(V, A_2)$ are acyclic and, since $A_1 \cup A_2 = A$, $\max\{w(A_1), w(A_2)\} \geqslant 0.5w(A)$. Therefore, it is a 0.5 approximation for the problem. Note that the algorithm has linear complexity.

Korte and Hausmann [13] proved that the greedy algorithm (i.e., construct a solution by repeatedly selecting the arc of maximum weight that does not form a directed cycle with the already chosen arcs) does not guarantee any fixed error ratio.

A more sophisticated algorithm for the unweighted problem was proposed by Berger and Shor [2]. They note that without loss of generality we can assume that $G$ has no cycles of length 2, since any bound that can be achieved under this assumption can also be achieved without it by a simple modification of the algorithm. They then develop an algorithm producing an acyclic subgraph of at least $(1/2 + \Omega(1/\sqrt{d_{\max}}))|A|$ arcs, where $d_{\max}$ is the maximum vertex degree of $G$. Even when cycles of length two exist, the solution contains at least a fraction $(1/2 + \Omega(1/\sqrt{d_{\max}}))$ of the number of arcs in an optimal solution. The running time of the algorithm is $O(|A| \cdot |V|)$.

In this paper we examine a variety of algorithms for the problem. Our main contribution is an algorithm for the unweighted problem that guarantees a bound similar to that achieved by Berger and Shore, but with time complexity $O(|A| + d_{\max}^3)$ which is better than $O(|A| \cdot |V|)$ in certain cases.

## 2. Inducing a solution from a permutation

Call an acyclic subgraph of $G$ *maximal* if it is not strictly contained in another acyclic subgraph of $G$. Since we assume that the arc weights are positive, the maximum acyclic subgraph is also maximal.

A permutation $\pi$ of $\{1, \ldots, n\}$ *induces* an acyclic subgraph $G_\pi = (V, A_\pi)$, where $A_\pi = \{(i,j) \in A \mid \pi(i) < \pi(j)\}$. Note that $G_\pi$ may not be maximal if it is not connected. However every maximal acyclic subgraph of $G$ is induced by some permutation. (One can always renumber the vertices of an acyclic graph so that each arc $(i,j)$ in it satisfies $i < j$, and if the graph is maximal then it is the one induced by this permutation.) Therefore, the maximum acyclic subgraph problem is exactly the problem of computing a permutation whose induced subgraph is of maximum weight.

We first describe a prototype algorithm that generates a permutation, $\pi$, such that $w(A_\pi) \geqslant 0.5w(A)$. For $i \in V$ and $S \subset V$, let $w_i^{\text{in}}(S) = \sum_{j \in S} w_{ji}$, $w_i^{\text{out}}(S) = \sum_{j \in S} w_{ij}$.

**Algorithm 1.**

1. Set $S = V$, $l = 1$, $u = n$.
2. Choose $i \in S$. Set $S \leftarrow S \setminus \{i\}$. If $w_i^{\text{in}}(S) \leqslant w_i^{\text{out}}(S)$, set $\pi(i) = l$, $l \leftarrow l + 1$. If $w_i^{\text{in}}(S) > w_i^{\text{out}}(S)$, set $\pi(i) = u$, $u \leftarrow u - 1$.
3. If $u \geqslant l$ go to Step 2. Else, stop and output $\pi$.

The weight of the arcs preserved by the algorithm is at least one half of the total weight of $A$ since this property holds in every iteration with respect to the arcs incident with vertex $i$ in the subgraph induced by $S$.

## 3. Randomization

The difficulty in obtaining a bound of more than 0.5 on the error while applying simple constructive algorithms arises when $w_i^{\text{in}}(S) \approx w_i^{\text{out}}(S)$ for many vertices. We will try to overcome the difficulties associated with such a situation and improve the bound for the unweighted

problem by treating the maximum vertex degree in the graph as a parameter.

We start by presenting a randomized algorithm. It is different from the one suggested by [2] but achieves a similar bound. We assume, without loss of generality, that $G$ contains no cycles of length 2. If it contains cycles of length 2 then, as done in [2], they can be temporarily removed before the algorithm is executed. Any permutation will induce a graph with exactly one arc from each such cycle.

**Algorithm 2.**

1. Partition $V$ into two subsets $V_1, V_2$ by assigning each vertex to each subset with probability 0.5. Let $A_r = \{(i,j) \mid i,j \in V_r$. Execute Step 2 for $r = 1, 2$.
2. Form a permutation $\pi_r$ of the vertices in $V_r$ by applying Algorithm 1 to $(V_r, A_r)$. The vertices are selected in increasing order of their indices.
3. Define the final permutation by choosing between $(\pi_1, \pi_2)$ and $(\pi_2, \pi_1)$ the permutation inducing a subgraph with the larger number of arcs.

**Theorem 3.** *Let APX be the expected number of arcs in the solution computed by Algorithm 2. Then*

$$APX = \left(0.5 + \Omega\left(\frac{1}{\sqrt{d_{\max}}}\right)\right)|A|.$$

**Proof.** Consider a vertex $i \in V_r$. Let

$$D_i^{\text{in}} = |(j,i) \in A\colon j > i|,$$
$$D_i^{\text{out}} = |(i,j) \in A\colon j > i|,$$
$$d_i^{\text{in}} = |(j,i)\colon j \in V_r,\ j > i|,$$
$$d_i^{\text{out}} = |(i,j)\colon j \in V_r,\ j > i|.$$

The random variable $d_i^{\text{in}}$ is binomially distributed with parameters $(0.5, D_i^{\text{in}})$. The reason is that for each arc incident with $i$ there is a probability of 0.5 that its other end is also in $V_r$. Similarly, $d_i^{\text{out}}$ is binomial with parameters $(0.5, D_i^{\text{out}})$.

Without loss of generality assume that $D_i^{\text{in}} \geq D_i^{\text{out}}$. For $a \geq 0$,

$$P \equiv \Pr(|d_i^{\text{in}} - d_i^{\text{out}}| \geq a) \geq \Pr(d_i^{\text{in}} - d_i^{\text{out}} \geq a).$$

Since, by our assumption, $G$ contains no cycles of length 2, $d_i^{\text{in}}$ and $d_i^{\text{out}}$ are independent random variables. Consider two independent binomial random variables, $X_1, X_2$ each with parameters $(0.5, D_i^{\text{in}})$. Then,

$$\begin{aligned} P &\geq \Pr(X_1 - X_2 \geq a) \\ &\geq \Pr(X_1 \geq 0.5D_i^{\text{in}} + a,\ X_2 \leq 0.5D_i^{\text{in}}) \\ &= 0.5\Pr(X_1 \geq 0.5D_i^{\text{in}} + a), \end{aligned}$$

where the first inequality follows from our assumption that $D_i^{\text{in}} \geq D_i^{\text{out}}$. Setting $a$ to the standard deviation of $X_1$, $a = \frac{1}{2}(D_i^{\text{in}})^{1/2}$, we obtain that for some constant [1], $\beta > 0$,

$$\Pr(|d_i^{\text{in}} - d_i^{\text{out}}| \geq a) \geq \beta.$$

Let $D_i = D_i^{\text{in}} + D_i^{\text{out}}$, then, $D_i^{\text{in}} \geq D_i/2$ and $a = \Omega(\sqrt{D_i}) = \Omega(D_i/\sqrt{D_i}) = \Omega(D_i/\sqrt{d_{\max}})$.

Step 2 assigns vertex $i$ to the next lower or higher position in the permutation according to the sign of the difference $d_i^{\text{in}} - d_i^{\text{out}}$. The total number of arcs induced by $\pi_r$ is

$$\begin{aligned} &\sum_{i \in V_r} \max(d_i^{\text{in}}, d_i^{\text{out}}) \\ &\quad = \sum_{i \in V_r} \left( \left(\frac{d_i^{\text{in}} + d_i^{\text{out}}}{2}\right) + \frac{1}{2}|d_i^{\text{in}} - d_i^{\text{out}}| \right) \\ &\quad = 0.5|A_r| + 0.5 \sum_{i \in V_r} |d_i^{\text{in}} - d_i^{\text{out}}|. \end{aligned}$$

We conclude that the expected number of arcs induced by $\pi_r$ satisfies

$$APX_r = 0.5|A_r| + \Omega\left(\sum_i \frac{D_i}{\sqrt{d_{\max}}}\right).$$

Since, $\sum_i D_i = |A|$, it follows that

[1] For a binomial variable $X$ with parameters $(p,n)$ such that $p \leq 0.5$, the probability $p_n = \Pr(X - np \geq (np(1-p))^{1/2})$ is positive for all values $n$. It converges to a positive limit, by the central limit theorem, and hence the infimum of $p_n$ over $n = 1, 2, \ldots$ is positive.

$$APX_1 + APX_2 = 0.5(|A_1| + |A_2|) + \Omega\left(\frac{|A|}{\sqrt{d_{\max}}}\right).$$

Step 3 of the algorithm orders the two partial permutations so that the resulting induced subgraph contains at least half of the arcs connecting vertices in $V_1$ with vertices in $V_2$. The theorem now follows. □

**Remark 4.** The expected number of arcs induced by the algorithm is also $(0.5+\Omega(1/\bar{d}))|A|$ where $\bar{d} = 2|A|/n$ is the average degree in $G$. This bound results since the expected number of arcs, in addition to $0.5|A|$ gained in Step 3 is at least one per vertex. This bound can be better than the one stated in the theorem when the average degree is bounded while the maximum degree is not.

## 4. Derandomization

We now describe the details necessary for an efficient execution of derandomization through the *method of conditional probabilities* (see, for example, [1]). Using this method we turn our randomized algorithm into a deterministic one with the same performance guarantee.

Instead of randomly generating $V_1, V_2$ in Algorithm 2, the method assigns successively a vertex at a time to one of the subsets so that the expected size of the solution obtained by continuing randomly from this point on is maximized. The reason is that the expected size of the random solution is the average of the two expectations obtained conditioned on the assignment of the current vertex. Assigning to the set giving the larger value guarantees at least the unconditional expected value.

Note, from the proof of Theorem 3, that the term $\Omega(1/\sqrt{d_{\max}}))|A|$ comes from the analysis of the expected number of arcs obtained *within* the sets $V_1$ and $V_2$. Step 3 of Algorithm 2 guarantees half of the arcs between the sets (and if we just consider expectations then both $(\pi_1, \pi_2)$ and $(\pi_2, \pi_1)$ are suitable for satisfying the theorem). We will now show how the above term can be guaranteed deterministically. With Step 3, the complete solution will have $(0.5 + \Omega(1/\sqrt{d_{\max}}))|A|$ arcs.

Suppose that each of the vertices $1, \ldots, k-1$ has already been assigned to $V_1$ or $V_2$. We will now show how to compute efficiently the expected number of arcs within these sets obtained by Step 2 of Algorithm 2, when $V_1, V_2$ are completed by randomly assigning the vertices $k, \ldots, n$. As in the proof of Theorem 3, we associate each arc with its lower index vertex.

Consider a given vertex $i \in \{1, \ldots, k-1\}$. Suppose that it has been assigned to $V_r$. Clearly, Algorithm 2 guarantees that the approximate solution will contain at least one half of the arcs within $V_r$ which are associated with $i$. We are interested now in computing the expected number of *additional such arcs* that will be contained in our solution, given the initial assignment of the first $k-1$ vertices. Let $E(x_i, y_i, z_i)$ denote the expected number of such additional arcs, where

$$x_i = |(j,i): j \in V_r,\ i < j < k| - |(i,j): j \in V_r,\ i < j < k|,$$
$$y_i = |(j,i): j \geqslant k|,$$
$$z_i = |(i,j): j \geqslant k|.$$

Then,

$$E(x,0,0) = 0.5|x|, \quad x = \ldots, -2, -1, 0, 1, 2, \ldots,$$

and the other values can be computed recursively by

$$E(x,y,0) = 0.5E(x+1, y-1, 0) + 0.5E(x, y-1, 0),$$
$$E(x,y,z) = 0.5E(x,y,z-1) + 0.5E(x-1, y, z-1), \quad z \geqslant 1.$$

The recursion can be applied by computing $E(x,y,0)$ for $y$ fixed and all values of $x$, starting with $y = 0$ and then $y = 1$ and so on. Then $E(x,y,z)$ is computed for $z$ fixed and all values of $x, y$, starting with $z = 1$ and then $z = 2$ and so on. The overall effort in computing these values for $x = -d_{\max}, \ldots, d_{\max}$, $y = 0, \ldots, d_{\max}$,

$z = 0, \ldots, d_{\max}$ is $O(d_{\max}^3)$, where $d_{\max}$ is the maximum vertex degree in $G$.

The expected gain at the arcs associated with a vertex $i \geqslant k$ is simply $E(0, y_i, z_i)$.

Note that the above expectations are valid independently of the order by which vertices are considered for joining the two sets.

The expected solution size, conditioned on a given partial assignment of vertices into the two subsets is $0.5|A| + \sum_{i=1}^{k-1} E(x_i, y_i, z_i) + \sum_{i=k}^{n} E(0, y_i, z_i)$.

Whenever an unassigned vertex is considered, the two alternatives for assigning it are compared. Each alternative affects the $(x, y, z)$ values of its neighbors, and then the preferred assignment is made and the revised values of the neighbors are determined. The total effort for making these revisions and comparing the expected solution values associated with the two alternatives of each vertex at its turn are altogether $O(|A|)$.

Thus altogether, computing the $E$ values, determining the assignment to the two sets, and the execution of Steps 2 and 3 of Algorithm 2 take $O(d_{\max}^3 + |A|)$.

## 5. Constructive algorithms

We mentioned that a 0.5 approximation can be obtained by comparing the graphs defined by any permutation and its reverse. We can further show that a better bound cannot be guaranteed by selecting any polynomial set of permutations (independent of the particular instance of the problem) and then comparing the solutions they induce. Therefore, we now turn to investigate procedures that construct a permutation while taking into account the data of the given instance of the problem.

Our basic tool is Algorithm 1. Note that it does not specify the order by which vertices are examined in Step 2. We will consider some "attractive" rules for examining the vertices, and describe a "bad example" for each. All the bad examples describe unweighted instances.

Recall that $w_i^{\text{in}}(S) = \sum_{j \in S} w_{ji}$, $w_i^{\text{out}}(S) = \sum_{j \in S} w_{ij}$. Let $W_i(S) = \max\{w_i^{\text{in}}(S), w_i^{\text{out}}(S)\}$. Our first rule gains at each iteration the maximum possible weight:

**Algorithm 5.** In Step 2 of Algorithm 1, choose the vertex $i \in S$ with the largest $W_i(S)$.

**Example 6.** Let $n = 2k + 1$ and $A = \{(j, k + 1) \mid j = 1, \ldots, k\} \cup \{(k + 1, j) \mid j = k + 2, \ldots, 2k + 1\}$. The graph is acyclic and the optimal solution contains $A$. However, Algorithm 5 will start by choosing $k + 1$ and assign $\pi(k + 1)$ to either 1 or $n$, thus losing half of the arcs.

Let $w_i(S) = \min\{w_i^{\text{in}}(S), w_i^{\text{out}}(S)$. Our second rule is similar to that used by Lin and Sahni [16] to solves their bottleneck problem. It minimizes at each iteration the weight of lost arcs:

**Algorithm 7.** In Step 2 of Algorithm 1, choose the vertex $i \in S$ with the smallest $w_i(S)$.

**Example 8.** Let $n = k^2$, $V = V_1 \cup V_2 \cup \cdots \cup V_k$, where $V_i = \{(i - 1)k + 1, \ldots, ik\}$, $i = 1, \ldots, k$. Let $A = \{(p, q) \mid p \in V_i,\ q \in V_{i+1},\ i = 1, \ldots, k - 1\} \cup \{(p, q) \mid p \in V_k,\ q \in V_1\}$. Note that $A \setminus \{(p, q) \mid p \in V_k,\ q \in V_1\}$ is acyclic and the optimal solution contains $k^2(k - 1)$ arcs. However, the following sequence of selections by Algorithm 7 is possible. Initially, $S = V$ and $w_i(S) = k$ for all $i \in V$. Set $\pi(1) = 1$. Now, $S = V \setminus \{1\}$, $w_i(S) = k - 1$ for the vertices $i$ in $V_2$ and $V_k$, while $w_i(S) = k$ for all of the other vertices of $S$. Vertex $k + 1 \in V_2$ may be the next to be selected and $\pi(k + 1)$ be set to 2. The candidates for selection are now the vertices in $V_1$, $V_2$, $V_3$ and $V_k$. The algorithm may set now $\pi(2k + 1) = 3$, then $\pi(3k + 1) = 4$ and so on till $\pi((k - 1)k + 1)$ is set to $k$. There are $k - 1$ vertices left now in $S$ from each set. The algorithm may proceed selecting the vertices in the following order: $(2, k + 2, 2k + 2, \ldots, (k - 1)k + 2, 3, k + 3, \ldots)$. This way the solution obtained contains the $k^2(k + 1)/2$ arcs of the form $(ik + j, (i + 1)k + l)$ for $l \geqslant j$. Asymptotically this is just half the optimal number.

Let $\hat{w}_i(S) = |w_i^{\text{in}}(S) - w_i^{\text{out}}(S)|$. Our third

rule maximizes at each iteration the excess of gained weight over the lost weight:

**Algorithm 9.** In Step 2 of Algorithm 1, choose the vertex $i \in S$ with the largest $\hat{w}_i(S)$.

**Example 10.** Let $n = 2k + 3$, $A = \{(j, k+1) \mid j = 1, \ldots, k\} \cup \{(k+1, j) \mid j = k+2, \ldots, 2k+3\}$. Then vertex $k+1$ initially has $\hat{w}_{k+1}(V) = 2$ while all the other vertices have $\hat{w}_i(S) = 1$. Hence, $k+1$ will be the first to be chosen by Algorithm 9 and the number of lost arcs in the solution is asymptotically $|A|/2$.

Let

$$r_i(S) = \max\left\{\frac{w_i^{\text{out}}(S)}{w_i^{\text{in}}(S)}, \frac{w_i^{\text{in}}(S)}{w_i^{\text{out}}(S)}\right\}.$$

Our fourth rule maximizes at each iteration the ratio of gained weight to the lost weight:

**Algorithm 11.** In Step 2 of Algorithm 1, choose the vertex $i \in S$ with the largest $r_i(S)$.

**Example 12.** Consider again the graph of Example 8. An optimal solution consists of $k^2(k-1)$ arcs, for example $\{(i, j) \mid i \in V_l,\ j \in V_{l+1},\ l = 1, \ldots, k-1\}$. Algorithm 11 may choose a vertex from $V_1$, then from $V_2, V_3, \ldots, V_{k-2}$, and then in a cyclic order of the sets, starting from $V_{k-1}$ two vertices from each subset. The chosen permutation is described for $k = 6$ in the following table:

| $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 7 |
| 9 | 11 | 13 | 15 | 6 | 8 |
| 10 | 12 | 14 | 16 | 17 | 19 |
| 21 | 23 | 25 | 27 | 18 | 20 |
| 22 | 24 | 26 | 28 | 29 | 31 |
| 33 | 34 | 35 | 36 | 30 | 32 |

This permutation induces a solution with $k(k+1)/2$ arcs going from a vertex in $V_l$ to a vertex in $V_{l+1}$ for $l = 1, \ldots, k$. The total number of arcs chosen by Algorithm 11 is $k^2(k+1)/2$, and the ratio of the optimal to the approximate solutions is asymptotically 0.5.

We now describe a different approach to constructing a permutation. Instead of determining the exact value of $\pi(i)$ in Step 2, we only determine the relative location of $i$ with respect to the vertices that were already examined. Thus, at each stage we are given a (total) order on the set $Q = V \setminus S$. We choose $i \in S$, and select for it the best location with respect to that order. At any given point of the execution of the algorithm, let $\prec$ denote the order defined on the subset of $V$ scanned so far. In particular $k \preceq j$ means that $k$ is either $j$ or a vertex assigned to precede $j$.

**Algorithm 13.**

1. Choose $(i, j) \in A$. Set $Q = \{i, j\}$, $i \prec j$, $S = V \setminus Q$.
2. Suppose that this step is reached with an order $\prec$ of $Q$. Choose $i \in S$ and set $S \leftarrow S \setminus i$. Compute for each $j \in Q$

$$D_j = \sum_{k \mid k \preceq j} w_{ki} + \sum_{k \mid j \prec k} w_{ik}.$$

   Set $D_0 = \sum_{j \in Q} w_{ij}$. Let $D_l = \max\{D_j \mid j \in Q\}$. If $D_l \geq D_0$ extend $\prec$ by adding $i$ so that it is the immediate successor of $l$. If $D_l < D_0$, extend $\prec$ by adding $i$ as the first element.
3. If $S = \emptyset$, stop. Else, set $Q \leftarrow Q \cup \{i\}$ and go to Step 2.

**Example 14.** Let $A = \{(1, n)\} \cup \{(n, j) \mid j = 2, \ldots, n-1\} \cup \{(j, 1) \mid j = 1, \ldots, n-1\}$. If Algorithm 13 starts with $(1, n)$ then at least half of the other arcs must be lost. Thus the outcome is $n-1$ arcs while the optimum solution contains the $2(n-2)$ arcs in $A \setminus (1, n)$.

## 6. Local search

The general idea of a local search involves a definition of a *neighborhood* for each feasible solution of the problem. Then, the current solution is replaced by a better solution from its neighborhood if such a solution exists. Else, the algorithm stops with the current ("locally optimal") solution.

We find it natural to define neighborhoods with respect permutations. Let $V_\pi$ denote the neighborhood of $\pi$.

**Algorithm 15.** Apply local search with $V_\pi$ defined as follows: $\pi' \in V_\pi$ if for some $j \neq k$,

$$\pi' = (\pi_1, \ldots, \pi_j, \pi_k, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_{k+1}, \ldots)$$

or

$$\pi' = (\pi_1, \ldots, \pi_{k-1}, \pi_{k+1}, \ldots, \pi_j, \pi_k, \pi_{j+1}, \ldots).$$

In other words, $\pi'_i$ is obtained from $\pi$ by changing the position of one vertex. For example, $(1,2,3,6,4,5,7)$ and $(1,3,4,5,2,6,7)$ are in the neighborhood of $(1,2,3,4,5,6,7)$.

Algorithm 15 is a 0.5 approximation. We prove this claim by showing that any solution that contains less then half of the total arc weights cannot be induced by a locally optimal permutation. Such a solution must have at least one vertex such that the total weight of the arcs incident with it in the induced subgraph is strictly less then one half of the total weight of the arcs incident with it in $G$. But in this case a better permutation can be obtained by moving this vertex to either the first or the last position (one of these options will add more weight then what is lost by the change). Thus, the proposed solution cannot be locally optimal.

**Example 16.** Let $A = \{(i, i+1) \mid i = 1, \ldots, n-1\}$. The graph is acyclic and the optimal solution contains all the $n-1$ arcs. However, the permutation $(n-1, n, n-3, n-2, \ldots, 5, 6, 3, 4, 1, 2)$ is locally optimal, and induces the subgraph with the $n/2$ arcs $(n-1, n), (n-3, n-2), \ldots, (5,6), (3,4), (1,2)$. Any local change in the permutation may add at most one arc but will surely loose one arc as well.

**Algorithm 17.** Apply local search with $V_\pi$ defined as follows: $\pi' \in V_\pi$ if for some $j < k$,

$$\pi' = (\pi_1, \ldots, \pi_{j-1}, \pi_k, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_j, \pi_{k+1}, \ldots).$$

In other words, $\pi'$ is obtained from $\pi$ by swapping two vertices. For example, the permutation $(1,6,3,4,5,2,7)$ is in the neighborhood of $(1,2,3,4,5,6,7)$.

Algorithm 17 is a 0.5 approximation. We prove this claim by showing that any solution that contains less then one half of the total arc weight cannot be induced by a locally optimal permutation. It is sufficient to show that the weight of the subgraph induced by a locally optimal permutation, $\pi$, is at least as good as the subgraph induced by the reverse permutation, $\bar{\pi}$ (since the sum of the two is equal to the total arc weight in $G$). The reverse permutation, $\bar{\pi}$ can be obtained from $\pi$ by a sequence of $n/2$ swaps: First swap $\pi(1)$ and $\pi(n)$, then $\pi(2)$ with $\pi(n-1)$, and so on. The effect of the second swap on the weight of the induced subgraph is independent of the first swap because it only affects arcs whose two ends are different from both $\pi(1)$ and $\pi(n)$. Similarly, each successive swap affects the solution in the same way as it will affect it if we do it directly on the original permutation $\pi$. Since $\pi$ is a local maximum, none of these swaps increases the induced subgraph's weight. This proves our claim.

**Example 18.** Consider again Example 16. The permutation described there is also locally optimal with respect to Algorithm 17.

Even if we allow a larger neighborhood in which up to $k$ swaps are allowed for some fixed $k$, an extension of Example 16 still applies: Let $V = V_1 \cup V_2 \cup \cdots \cup V_{2l}$, where $|V_i| = m \gg k$ for $i = 1, \ldots, 2l$. From each vertex in $V_i$, for $i$ even, there are arcs going to all of the $m$ vertices in $V_{i+1}$. From each vertex in $V_i$, for $i$ odd, there are $m-k$ arcs going to vertices in $V_{i+1}$. These arcs are selected so that for a vertex in $V_i$, $i$ even, there are exactly $m-k$ arcs entering from $V_{i-1}$. The graph is acyclic and has $lm^2 + lm(m-k)$ arcs. Consider a permutation such that the vertices from each $V_i$ are consecutive, and the order of the sets is $V_{2l-1}, V_{2l}, V_{2l-3}, V_{2l-2}, \ldots, V_3, V_4, V_1, V_2$. From each vertex in $V_i$, $i$ even, the solution induced by the permutation contains all of the $m$ arcs leaving it. The solution contains none of the arcs going from $V_i$ to $V_{i+1}$ for $i$ odd. Thus it contains $lm$ arcs, which is about one half of the optimal

value. Any change of location of any single index entails a loss of at least $k$ arcs. Hence, by relocating no more than $k$ vertices no gain is possible.

## Acknowledgement

The proof that Algorithm 17 is a 0.5 approximation is due to Nili Gutman.

## References

[1] N. Alon and J.H. Spencer, *The Probabilistic Method* (John Wiley and Sons, New York, 1992).

[2] B. Berger and P.W. Shor, Approximation algorithms for the maximum acyclic subgraph problem, in: *Proc. 1st Ann. ACM–SIAM Symp. on Discrete Algorithms* (1990) 236–243.

[3] M.M. Flood, Exact and heuristic algorithms for the weighted feedback arc set problem: A special case of the skew-symmetric quadratic assignment problem, *Networks* **20** (1990) 1–23.

[4] A. Frank, How to make a digraph strongly connected, *Combinatorica* **1** (1981) 145–153.

[5] H.N. Gabow, A representation for crossing set families with applications to submodular flow problems, *Proc. 4th Ann. ACM–SIAM Symp. on Discrete Algorithms* (1993) 202–211.

[6] H.N. Gabow, A framework for cost-scaling algorithms for submodular flow problems, 1993.

[7] M. Grötschel, M. Jünger and G. Reinhelt, On the acyclic subgraph polytope, *Mathematical Programming* **33** (1985) 28–42.

[8] M. Grötchel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization* (Springer, Berlin, 1988).

[9] B. Korte, Approximation algorithms for discrete optimization problems, *Ann. Discrete Math.* **4** (1979) 85–120.

[10] R. Kaas, A branch and bound algorithm for the acyclic subgraph problem, *European J. Oper. Res.* **8** (1981) 335–362.

[11] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations* (Plenum, New York, 1972) 85–103.

[12] A. Karazanov, On the minimal number of arcs of a digraph meeting all its directed cutsets, abstract, *Graph Theory Newsletters* **8** (1979).

[13] B. Korte and D. Hausmann, An analysis of the greedy heuristic for independence systems, *Ann. Discrete Math.* **2** (1978) 65–74.

[14] M. Jünger, *Polyhedral Combinatorics and the Acyclic Subgraph Problem* (Heldermann, 1985).

[15] C.L. Lucchesi, A minimax equality for directed graphs, Ph.D. Dissertation, University of Waterloo, Waterloo, Ontario, 1976.

[16] L. Lin and S. Sahni, Fair edge deletion problems, *IEEE Trans. Comput.* **38** 7(1989) 56–761.

[17] S. Miyano, Systematized approaches to the complexity of subgraph problems, *J. Inform. Process.* **13** (1990) 442–447.

[18] M. Penn and Z. Nutov, Minimum feedback arc set and maximum integral dicycle packing in $K_{3,3}$-free digraphs, 1993.

[19] C.H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *J. Comput. System Sci.* **43** (1991) 425–440.

[20] V. Ramachandran, Finding a minimum feedback arc set in reducible flow graphs, *J. Algorithms* **9** (1988) 299–313.

[21] M. Yannakakis, Node- and edge-deletion NP complete problems, in: *Proc. 10th ACM Symp. on Theory of Computing* (ACM, New York, 1978) 253–264.