Approximation Algorithms for the Maximum Acyclic Subgraph Problem

Bonnie Berger[*][+]
Peter W. Shor[#]

Abstract

Given a directed graph $G = (V, A)$, the *maximum acyclic subgraph problem* is to find a subset A' of the arcs such that $G' = (V, A')$ is acyclic and A' has maximum cardinality. In this paper, we present polynomial-time and RNC algorithms which, when given any graph G without two-cycles, find an acyclic subgraph of size at least $(1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$, where $\Delta(G)$ is the maximum degree of G. This bound is tight, in terms of $|A|$, since there exists a class of graphs without two-cycles for which the largest acyclic subgraph has size at most $(1/2 + O(1/\sqrt{\Delta(G)}))|A|$. For the common case of low degree graphs, our algorithms provide an even better improvement over the known algorithms that find an acyclic subgraph of size $\frac{1}{2}|A|$. For example, for graphs without two-cycles, our algorithms find an acyclic subgraph of size at least $\frac{2}{3}|A|$ when $\Delta(G) = 2$ or 3, and $\frac{19}{30}|A|$ when $\Delta(G) = 4$ or 5. For $\Delta(G) = 2$, this bound is optimal in the worst case. Curiously, for 3-regular graphs, we can achieve $\frac{13}{18}|A|$, which is slightly better that $\frac{2}{3}|A|$. As a consequence of this work, we find that all graphs without two-cycles contain large acyclic subgraphs, a fact which was not previously known.

1 Introduction

Given a directed graph $G = (V, A)$ with no self-loops or multiple arcs, the *maximum acyclic subgraph problem* is to find a subset A' of the arcs such that $G' = (V, A')$ is acyclic and A' has maximum cardinality. An alternative formulation of this problem is the *feedback arc set problem* which is the problem of determining, in a directed graph, a minimum cardinality set of arcs that breaks all cycles. Whichever formulation is chosen, the above problem is important in the study of systems with feedback. A frequently used technique in the analysis of such systems is to model the structure of the system as a directed graph, and to make the system feedback-free by removing a small number of arcs. This feedback-free system can now be analyzed using standard methods. The feedback arcs can be reintroduced at a cost that is generally minimized if the size of the feedback arc set is small [Ram].

The maximum acyclic subgraph problem is NP-complete ([K],[GJ]). The standard techniques we have for dealing with such problems is to find either exact solutions for certain cases or approximation algorithms. Ramachandran [Ram] has made headway in the first approach by developing a poly-time algorithm for finding a minimum feedback arc set in reducible flow graphs. We take the second approach and find approximation algorithms for the maximum acyclic subgraph problem. Our algorithms take as input a directed graph $G = (V, A)$ and find a subset $\hat{A}$ of the arcs such that $\hat{G} = (V, \hat{A})$ is acyclic but $\hat{A}$ is not necessarily of maximum cardinality. The *performance ratio* of an approximation algorithm is $|\hat{A}|/\beta(G)$, where $\beta(G)$ is the optimal number of arcs in an acyclic subgraph of G. The *performance guarantee* is the worst case performance ratio over all possible graphs G.

Relatively little has been discovered about the approximability of the maximum acyclic subgraph problem. The known poly-time algorithms have performance ratio 1/2 because they find an $\hat{A}$ such that $|\hat{A}| \geq \frac{1}{2}|A|$. For graphs which contain two-cycles, this performance is the best possible in terms of $|A|$, since there are graphs with $|\hat{A}| = \frac{1}{2}|A|$. For the class of graphs which contain no two-cycles, however, the known algorithms still achieve no better than $\frac{1}{2}|A|$ in the worst case. Recently, Papadimitriou and Yannakakis [PY89] showed that the maximum acyclic subgraph problem is complete for MAX SNP[π]. The reader may refer to their paper for the definition of this class; however, this result seems to imply that one will not be able to obtain a poly-time algorithm guaranteed to achieve better than a constant factor time op-

[*]Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139

[#]AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

[+]The first author was supported in part by an ARO Graduate Fellowship, Air Force Grant OSR-86-0076, and NSF PYI Grant CCR-8657688 with matching support from UPS, IBM and Sun. This is work done in part when the first author was at AT&T Bell Labs, Murray Hill NJ.

timal, since if one could obtain such an algorithm for this problem, one could get such an algorithm for any problem in MAX SNP[π].

In this paper, we provide new poly-time and RNC algorithms which can be shown to do better than $\frac{1}{2}|A|$ for graphs without two-cycles through a substantially more complicated analysis. More specifically, we give a linear-time randomized algorithm, an $O(|V||A|)$-time deterministic algorithm, and a logarithmic-time, linear-processor RNC algorithm for finding an $\hat{A}$ such that $|\hat{A}| \geq (1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$, where $\Delta(G)$ is the maximum total degree of V in G. This result generalizes the corresponding $(1/2 + \Omega(1/\sqrt{|V|}))|V|^2$ bound for tournaments discovered by Spencer [S] using different techniques. Interestingly, our bound is tight, in terms of $|A|$, since the existence of a class of graphs without two-cycles for which the largest acyclic subgraph has size at most $(1/2 + O(1/\sqrt{\Delta(G)}))|A|$, for any $\Delta(G)$, follows from a result of Spencer [S] and de la Vega [D]. For the common case of low degree graphs, our algorithms provide an even better improvement over the known algorithms. For example, for graphs without two-cycles, our algorithms find an $\hat{A}$ of size at least $\frac{2}{3}|A|$ when $\Delta(G) = 2$ or 3, and $\frac{19}{30}|A|$ when $\Delta(G) = 4$ or 5. For $\Delta(G) = 2$, this bound is optimal in the worst case. Curiously, for 3-regular graphs, we can achieve $\frac{13}{18}|A|$, which is slightly better. As a consequence of this work, we find that all graphs without two-cycles contain large acyclic subgraphs, a fact which was not previously known.

Furthermore, we show that considering graphs without two-cycles is sufficient since two-cycles can be dealt with optimally; *i.e.*, we give a procedure for including exactly one arc from each two-cycle in any $\hat{A}$ found for a graph $G = (V, A)$ with two-cycles removed. This procedure will turn an approximation algorithm for graphs without two-cycles into an equally good approximation algorithm for graphs with two-cycles.

Combining the methods in this paper with the techniques of Leighton and Rao [LR88], it appears that Leighton and Rao [LR89] have obtained a $\log^2 |V|$ times optimal approximation algorithm for the feedback arc set problem. For some graphs with small feedback arc sets, the Leighton/Rao algorithm will perform better than those described here, at a large polynomial cost in running time, but for graphs with large feedback arc sets it will not do as well.

In this paper, we first review the known 2-approximation algorithms. Then in Section 3, we introduce a new technique and show how it can easily be used to give an alternative 2-approximation algorithm. In Section 4, we describe our technique for 2-cycle removal and reintroduction. As this technique deals optimally with two-cycles, in the remaining sections of the paper, we focus solely on algorithms which work on graphs without two-cycles. In Section 5, we describe poly-time randomized and RNC algorithms, which are based on the technique of Section 3 and achieve the better bound through a substantially more complicated analysis. The poly-time randomized algorithm is transformed into a deterministic one in Section 6. The hard part of this transformation is calculating the conditional expectations exactly and efficiently. We conclude in Section 7 with a description of the upper bound construction.

2 Review of Known 2-Approximation Algorithms

In this section, we review two algorithms for the maximum acyclic subgraph problem. By the nature of their approach, these algorithms do not seem amenable to achieving better than twice optimal (half the arcs) in the worst case, even for graphs without two-cycles.

The first approach uses a greedy strategy for arc inclusion in an approximate acyclic subgraph. The algorithm maintains two sets S and T. It greedily goes through all arcs of the graph $G = (V, A)$, adding an arc to S if with its addition $G_S = (V, S)$ remains acyclic, and otherwise adding the arc to T. When all arcs have been processed, the larger of S and T is selected for the approximate acyclic subgraph. It is easy to show that since $G_S = (V, S)$ is acyclic, $G_T = (V, T)$ is acyclic. Note that there are graphs and orderings of the chosen arcs for which this technique gives $|S| = |T|$.

The second algorithm works as follows. It arranges the vertices of the graph $G = (V, A)$ in some order from left to right on a line and then adds in all directed arcs of the graph. Those arcs directed from left to right are put in set S and those directed from right to left are put in set T. The larger of these two sets is chosen for the approximate acyclic subgraph. It should be obvious that neither $G_S = (V, S)$ or $G_T = (V, T)$ contains a cycle. Here again, note that there are graphs and orderings in which the vertices are laid out for which this technique gives $|S| = |T|$.

3 A New 2-Approximation Algorithm

Here we introduce a new technique and show how it can be used to give an alternative 2-approximation algorithm for the maximum acyclic subgraph problem.

We will call our general technique *Algorithm A(G)*, which takes the graph $G = (V, A)$ as its input. *Algorithm A(G)* processes the vertices of G, in any order, as follows. When processed, if a vertex has more incoming arcs than outgoing ones, the incoming arcs are removed from the graph and added to set $\hat{A}$ and the outgoing arcs are removed from the graph and thrown in the trash. In the case where a vertex has at least as many outgoing arcs as incoming ones, we add the outgoing arcs to the set $\hat{A}$ and discard the incoming ones. When all vertices are processed, $\hat{A}$ is returned.

Theorem 1 *Algorithm A(G) selects an $\hat{A}$ such that $|\hat{A}| \geq \frac{1}{2}|A|$.*

Proof *Algorithm A(G)* examines each arc in G once and always adds at least as many arcs to the approximate acyclic subgraph as it throws away. □

Theorem 2 *Algorithm A(G) selects an $\hat{A}$ such that $\hat{G} = (V, \hat{A})$ is acyclic.*

Proof Suppose $\hat{G}$ contains a cycle $x_1, \ldots, x_k, x_1$. Without loss of generality, let x_1 be the first vertex in this cycle processed by *Algorithm A(G)*. Then either arc $<x_k, x_1>$ or arc $<x_1, x_2>$ would have been thrown in the trash. But then, $x_1, \ldots, x_k$ could not form a cycle in $\hat{G}$, a contradiction. □

Theorem 3 *The running time of Algorithm A(G) is $O(|V| + |A|)$.*

Remark 1: Observe that *Algorithm A(G)* may discard an arc not contained in any cycle. To prevent this, we can add an extra step before processing each vertex of G. We run a strongly connected components algorithm on the graph $\hat{G} = (V, \hat{A})$ (everything we haven't thrown in the trash) and put all arcs that go between strongly connected components into $\hat{A}$. This modification may also be made to the algorithms based on *Algorithm A(G)* discussed later in this paper. It adds $O(|V| \cdot |A|)$ to the running time of an algorithm, and although it does not improve our theoretical results, it may improve the performance of the algorithms in practice.

4 A Technique for Two-Cycle Removal and Reintroduction

In this section, we introduce our technique for 2-cycle removal and reintroduction. Basically, the technique works as follows: given any graph with two-cycles, we remove all two-cycles from the graph, run an algorithm for the maximum acyclic subgraph problem on the remaining graph which works on graphs without two-cycles, and then reintroduce one arc from each two-cycle back into the acyclic subgraph while keeping it acyclic. We will prove that the performance guarantee we get for the graph including two-cycles is at least as good as the one we get for the graph with two-cycles removed.

We define *Algorithm C(X,G)*, which takes as input *Algorithm X* (any algorithm which finds an acyclic subgraph for graphs with no two-cycles) and graph $G = (V, A)$ and which outputs $\hat{A}$, as follows:

1. Construct $G_2 = (V, A_2)$ such that $A_2 \leftarrow \{<u,v> \mid <u,v> \in A,\ <v,u> \notin A\}$
(*i.e.*, G_2 is G with two-cycles removed.) This step takes time $O(|A|)$.

2. Run *Algorithm* $X(G_2)$ on input graph G_2 and call its output $\hat{A}_2$.

3. Label the vertices in V with $v_1, \ldots, v_n$ such that if $< v_i, v_j > \in \hat{A}_2$, then $i < j$. Note that such a labeling is always possible from the assumption that *Algorithm* $X(G_2)$ returns an acyclic subgraph $\hat{A}_2$. This step is merely topological sort which takes time $O(|V| + |A|)$.

4. Construct $\hat{G} = (V, \hat{A})$ such that $\hat{A} \leftarrow \hat{A}_2 \cup \{<v_i, v_j> \mid <v_i, v_j> \in A,\ <v_j, v_i> \in A,\ i < j\}$
(*i.e.*, one arc from each 2-cycle in A is added to those arcs in $\hat{A}_2$.) This step takes time $O(|A|)$.

   Observe that all arcs in $\hat{A}$ are from v_i to v_j where $i < j$, so $\hat{G}$ must be acyclic.

Theorem 4 *The performance guarantee of Algorithm C(X,G) is at least as good as the performance guarantee of Algorithm $X(G_2)$.*

Proof Let $\psi(G)$ be the number of two-cycles in G. Recall that G_2 is G with two-cycles removed. The effect of *Algorithm C(X,G)*, Step 4 is to add $\psi(G)$ arcs to $\hat{A}_2$. Furthermore, $\psi(G)$ is the optimal number of arcs which can be added to $\hat{A}$ from the $\psi(G)$ two-cycles so that $\hat{G}$ contains no two-cycles; hence $\beta(G) = \beta(G_2) + \psi(G)$. Therefore,

$$\frac{|\hat{A}|}{\beta(G)} = \frac{|\hat{A}_2| + \psi(G)}{\beta(G_2) + \psi(G)} \geq \frac{|\hat{A}_2|}{\beta(G_2)}. \quad \square$$

Remark 2: All the algorithms in the remaining sections of this paper will take graphs without two-cycles as input since by Theorem 4, we can always pass these algorithms to *Algorithm C(X,G)* to achieve at least as good a performance guarantee for graphs which contain two-cycles.

5 New Randomized Algorithms

5.1 A Poly-Time Randomized Algorithm

In this section, we present an $O(|V| + |A|)$-time randomized algorithm for the maximum acyclic subgraph problem which, when given a graph $G = (V, A)$ with no two-cycles, finds an $\hat{A}$ such that $|\hat{A}| \geq (1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$. This result generalizes the corresponding $(1/2 + \Omega(1/\sqrt{|V|}))|V|^2$ bound for tournaments discovered by Spencer [S] using different techniques.

We define *Algorithm R(G)*, which takes graph $G = (V, A)$, with no two-cycles, as input and which outputs $\hat{A}$, by modifying *Algorithm A(G)* as follows: instead of processing the vertices of G in any order, randomly order the vertices at the start and process them in that order.

We want to show that the expected size of the $\hat{A}$ that *Algorithm R(G)* returns is at least $(1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$. We first consider the difference between the number of arcs contributed to $\hat{A}$ and the number of arcs thrown away, when a particular vertex v is processed. This difference can be modeled as a walk of length $d_{\text{out}}(v) + d_{\text{in}}(v)$, from $|d_{\text{out}}(v) - d_{\text{in}}(v)|$ to 0, where each time a neighbor of v is processed, a $+1$ or a -1 step is taken. The direction of each unit step of the walk is determined by whether the corresponding adjacent vertex processed was an in or out neighbor of v. Note that a vertex cannot be both an in and an out neighbor since G contains no two-cycles. The walk is of length $d_{\text{out}}(v) + d_{\text{in}}(v)$ since this is the number of neighbors of v. The value of the walk at the time when v is processed is the difference between the number of arcs contributed to $\hat{A}$ and the number thrown away. This walk is in fact a random walk since the neighboring vertices are processed in random order. Equivalently, we will look at the walk backwards as a random walk that starts at 0 and ends at $|d_{\text{out}}(v) - d_{\text{in}}(v)|$.

Now let us consider the random walk more closely. Let $w(i)$ be the value of the random walk w after i steps. Then, $w(0) = 0$ and $w(d) = t$, where d is the length of the walk and t is the ending point of the walk. Thus, $d = d_{\text{out}}(v) + d_{\text{in}}(v)$ and $t = |d_{\text{out}}(v) - d_{\text{in}}(v)|$. *Algorithm R(G)* chooses uniformly from all walks with steps of ± 1 that have length d, start at 0, and end at t. Since a walk is determined solely by the positions of the $+1$'s and -1's, each walk is generated by exactly $d_{\text{in}}(v)!\,d_{\text{out}}(v)!$ permutations of the neighbors of v. Hence, *Algorithm R(G)* generates each walk with equal probability.

After setting up the above formulation we can now prove the following two lemmas. Essentially, they show that when v is processed, the expected difference between the number of arcs added to $\hat{A}$ and the number thrown away is $\Omega(\sqrt{d})$. The first lemma states that conditioning on the random walk ending at t gives an expected difference no worse than half that given by conditioning on ending at the origin. The second lemma proves the desired result in terms of the latter smaller expected difference. In these two lemmas we assume that d is even. Once we have the result for even d, it is easy to show that it also holds for odd d.

Lemma 1 *$E(\,|w(r)|\mid w(d) = t) \geq \frac{1}{2}E(\,|w(r)|\mid w(d) = 0)$ for d and t even, $0 \leq r \leq d$, and $0 \leq t \leq d$.*

Proof We choose two correlated random walks, w_1 and w_2, of length d as follows, so that w_1 is uniformly distributed over random walks ending at t and w_2 is distributed uniformly over random walks ending at 0. Choose w_2 uniformly such that $w_2(d) = 0$. Construct w_1 from w_2 by choosing at random $t/2$ of the -1 steps in w_2 and changing them to $+1$ steps. Observe that w_1 is a walk ending at $w_1(d) = t$. Furthermore, w_1 is distributed uniformly among walks of length d starting at 0 and ending at t. This can be seen by observing that the same number of w_2's can give rise to each w_1. Now we have the following:

$$
\begin{aligned}
& E(\,|w_1(r)|\,) \\
&= \sum_{m}\sum_{l\geq m} |l| \Pr(w_1(r) = l \text{ and } w_2(r) = m) \\
&= \sum_{m\geq 0}\sum_{l\geq m} |l| \Pr(w_1(r) = l \text{ and } w_2(r) = m) \\
&\quad + \sum_{m<0}\sum_{l\geq m} |l| \Pr(w_1(r) = l \text{ and } w_2(r) = m) \\
&\geq \sum_{m\geq 0}\sum_{l\geq m} |m| \Pr(w_1(r) = l \text{ and } w_2(r) = m) \\
&= \sum_{m\geq 0} |m| \left[\sum_{l\geq m} \Pr(w_1(r) = l \text{ and } w_2(r) = m)\right] \\
&= \sum_{m\geq 0} |m| \Pr(w_2(r) = m) \\
&= \tfrac{1}{2}E(\,|w_2(r)|\,),
\end{aligned}
$$

since $\Pr(w_2(r) = m) = \Pr(w_2(r) = -m)$. □

Lemma 2 *$E(\,|w(r)| \mid w(d) = 0) \geq \Omega(\sqrt{d})$, when $\frac{d}{4} \leq r \leq \frac{3d}{4}$.*

Proof Note that

$$\Pr(w(r) = l \mid w(d) = 0) \leq \Pr(w(r) = 0 \mid w(d) = 0)$$

since (by the hypergeometric distribution) we can explicitly calculate the probabilities and see that

$$\frac{\binom{r}{\frac{r+l}{2}}\binom{d-r}{\frac{d-r-l}{2}}}{\binom{d}{\frac{d}{2}}} \leq \frac{\binom{r}{\frac{r}{2}}\binom{d-r}{\frac{d-r}{2}}}{\binom{d}{\frac{d}{2}}}.$$

Also note that

$$
\begin{aligned}
\Pr(w(r) = 0 \mid w(d) = 0) &= \frac{\binom{r}{\frac{r}{2}}\binom{d-r}{\frac{d-r}{2}}}{\binom{d}{\frac{d}{2}}} \\
&\leq c\sqrt{\tfrac{d}{r(d-r)}} \\
&\leq \frac{c'}{\sqrt{d}} \quad \text{for } \tfrac{d}{4} \leq r \leq \tfrac{3d}{4}.
\end{aligned}
$$

Thus,

$$
\begin{aligned}
& \Pr(|w(r)| \leq l \mid w(d) = 0) \\
&\leq (2l+1)\frac{c'}{\sqrt{d}} \quad \text{for } \tfrac{d}{4} \leq r \leq \tfrac{3d}{4} \\
&\leq \frac{1}{2} \quad \text{if } l \leq \tfrac{\sqrt{d}}{6c'}.
\end{aligned}
$$

Therefore,

$$E(\,|w(r)| \;|\; w(d)=0) \geq \frac{\sqrt{d}}{12c'} \quad \text{for } \tfrac{d}{4} \leq r \leq \tfrac{3d}{4}. \;\square$$

We will use the above two lemmas to obtain the following result.

Theorem 5 *Algorithm R(G) returns an $\hat{A}$ such that* $E(\,|\hat{A}|\,) \geq (1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$.

Proof Suppose that the time when v is processed corresponds to the time we have taken r steps on the random walk w and that $w(r) = l$. Let d be the degree of v. Let $\rho_{\hat{A}}(v)$ be the number of arcs contributed to $\hat{A}$ by v. This is the maximum of the number of in-arcs and the number of out-arcs adjacent to v when v is processed. Since in- and out-arcs correspond to ± 1 steps, and since we are walking backwards (i.e., the ith step of w corresponds to the $(d+1-i)$-th neighbor of v removed), $\rho_{\hat{A}}(v)$ is the maximum of the number of $+1$ steps and the number of -1 steps in the first r steps of the random walk. With this definition, it is easy to show that $\rho_{\hat{A}}(v) = (r+|l|)/2$. Taking expectations, we find

$$\begin{aligned} &E(\rho_{\hat{A}}(v)) \\ &= E\left(\frac{r+|l|}{2}\right) \\ &= \frac{E(r)}{2} + \frac{E(|l|)}{2} \\ &= \frac{d}{4} + \Omega(\sqrt{d}) \quad \text{by Lemmas 1 and 2} \\ &= \frac{d}{4} + \Omega\left(\frac{d}{\sqrt{\Delta(G)}}\right) \quad \text{since } \sqrt{d} \geq d/\sqrt{\Delta(G)}. \end{aligned}$$

Then the total number of arcs in $\hat{A}$ is

$$\begin{aligned} \sum_{v\in V} E(\rho_{\hat{A}}(v)) &= \sum_{v\in V} d(v)\left(\frac{1}{4} + \Omega\left(\frac{1}{\sqrt{\Delta(G)}}\right)\right) \\ &= 2|A|\left(\frac{1}{4} + \Omega\left(\frac{1}{\sqrt{\Delta(G)}}\right)\right) \\ &= |A|\left(\frac{1}{2} + \Omega\left(\frac{1}{\sqrt{\Delta(G)}}\right)\right). \;\square \end{aligned}$$

Remark 3: When G is a low degree graph, we can calculate the exact expectation of the size of $\hat{A}$ returned by *Algorithm R(G)* by going through all possible cases. For example, when the degree is 2, the possible random walks of length 2 starting and ending at 0 are UD and DU (here U means a $+1$ step and D means a -1 step), both of which give a bound of $(2/d)E(\frac{1}{2}(r+|l|)) = \frac{2}{3}$. This gives the $\frac{2}{3}|A|$ bound for degree 2 graphs, since it is easy to check (by comparing this case with the cases UU and DD) that the walks starting at 0 and ending at 0 give the smallest bounds. When the degree is 3, the worst case is walks starting at 0 and ending at 1. These walks are UUD, UDU, and DUU. Walks UDU and DUU give a value of $(2/d)E(\frac{1}{2}(r+|l|)) = \frac{2}{3}$, while walk UUD gives $\frac{5}{6}$. Averaging these values gives $\frac{13}{18}$, implying the bound of $\frac{13}{18}|A|$ for 3-regular graphs. This value is larger than the $\frac{2}{3}|A|$ we get for degree 2 because with odd degree, the walks cannot end at 0. For degree 4, the worst case again is walks ending at 0. We need only compute walks starting with U, because the ones starting with D are symmetric. The walk UUDD gives $(2/d)E(\frac{1}{2}(r+|l|)) = \frac{7}{10}$, while walks UDUD and UDDU give $\frac{6}{10}$. The average is thus $\frac{19}{30}$, implying $\frac{19}{30}|A|$ for degree 4 graphs. Calculations for degree 5 show that the constants are larger than degree 4 (again, because 5 is odd), so the $\frac{19}{30}|A|$ bound holds also for degree 5 graphs.

5.2 An RNC Algorithm

Algorithm R(G) can be easily parallelized (*i.e.,* shown to be in the complexity class *RNC*). Once we have assigned a randomized ordering to the vertices of G, we can process all of them in parallel. When we process a vertex, we will only consider its neighbors with a higher number in the assigned ordering. We will place in $\hat{A}$ either all the out-arcs to or all the in-arcs from the higher-numbered neighbors, whichever set is larger. This procedure requires a linear number of processors and logarithmic time.

In fact, an RNC algorithm whose analysis depends on 5-wise independence can probably be shown to work through a messy analysis. Then the methods in [BR] can be used to convert it to an NC algorithm.

6 A New Deterministic Algorithm

Using the results of the previous section, we are able to give a poly-time deterministic algorithm for the maximum acyclic subgraph problem which, when given a graph $G = (V, A)$ with no two-cycles, finds an $\hat{A}$ such that $|\hat{A}| \geq (1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$. To achieve this result, we use a technique developed by Raghavan and Spencer [Rag,S] for transforming a randomized algorithm into a deterministic one. The hard part of applying this technique is calculating the conditional expectations exactly and efficiently.

Here we define *Algorithm D(G)*, which takes graph $G = (V, A)$, with no two-cycles, as input and which outputs $\hat{A}$. Let $E(|\hat{A}|)$ be the expected size of $\hat{A}$ for the randomized algorithm of Section 5.1. Let $\text{procd}(v_1, \ldots, v_k)$ mean that $v_1, \ldots, v_k$ have already been processed in the order in which they are listed. The algorithm is as follows:

For $i = 1$ to $|V|$ do:

```
1. At the ith iteration of this loop, assume vertices labeled $v_1, \ldots, v_{i-1}$ have already been processed in that order.
2. Choose the $v \in V - \{v_1, \ldots, v_{i-1}\}$ for which $E(|\hat{A}| \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v))$ is maximized.
3. Label $v$ with $v_i$.
```

Theorem 6 *Algorithm D(G) returns an $\hat{A}$ such that $|\hat{A}| \geq (1/2 + \Omega(1/\sqrt{\Delta(G)}))|A|$.*

Proof Note that $E(|\hat{A}|)$ for the randomized algorithm at step i is $E(|\hat{A}| \mid \mathrm{proc}(v_1, \ldots, v_{i-1}))$ which is $\frac{1}{|V|-(i-1)} \sum_{v \in V - \{v_1, \ldots, v_{i-1}\}} E(|\hat{A}| \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v))$. This expected value is an average. *Algorithm D(G)* picks

$$\max_{v \in V - \{v_1, \ldots, v_{i-1}\}} E(|\hat{A}| \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v)),$$

which must be at least as large as the average. Since at each step, *Algorithm D(G)* has at least as large an $E(|\hat{A}|)$ as *Algorithm R(G)*, then this relation carries over to the total expected values at the end of the algorithms. By Theorem 5, this completes the proof. □

Theorem 7 *$E(|\hat{A}| \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v))$ can be calculated exactly.*

Proof We know how many arcs all of the processed vertices contribute to $\hat{A}$. For the unprocessed vertices, we can calculate exactly the expected number of arcs they add to $\hat{A}$ by using the random walk formulation from Section 5.1. Suppose we want to compute $E(|\hat{A}| \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v))$. Let $\rho_{\hat{A}}(v)$ be the number of arcs contributed to $\hat{A}$ by v. Then,

$$\begin{aligned} & E(|\hat{A}| \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v)) \\ & = \sum_{v'' \in \{v_1, \ldots, v_{i-1}, v\}} \rho_{\hat{A}}(v'') + \\ & \qquad \sum_{v' \in V - \{v_1, \ldots, v_{i-1}, v\}} E(\rho_{\hat{A}}(v') \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v)). \end{aligned}$$

The first sum is easy to compute. For the second, consider the following. Let v' have $d_{\mathrm{out}}(v')$ out-arcs into the set $V - \{v_1, \ldots, v_{i-1}, v\}$ and $d_{\mathrm{in}}(v')$ in-arcs from the set $V - \{v_1, \ldots, v_{i-1}, v\}$. Notice that we are now using $d_{\mathrm{out}}(v')$ and $d_{\mathrm{in}}(v')$ for the out- and in-degree of v' with respect to the subgraph of G induced by the vertex set $V - \{v_1, \ldots, v_{i-1}, v\}$, and not the original graph G. Let $d(v')$ be $d_{\mathrm{out}}(v') + d_{\mathrm{in}}(v')$ and $t(v')$ be $|d_{\mathrm{out}}(v') - d_{\mathrm{in}}(v')|$. Let w be a random walk from 0 to $t(v')$ taking $d(v')$ steps and r be a variable uniformly taking integer values between 0 and $d(v')$ independent of w. Finally, let $l = w(r)$ be the random variable taking the value of w at r. By the argument on random walks in the previous section,

$$\begin{aligned} E(\rho_{\hat{A}}(v') \mid \mathrm{procd}(v_1, \ldots, v_{i-1}, v)) &= E\left(\frac{r + |l|}{2}\right) \\ &= \frac{E(r)}{2} + \frac{E(|l|)}{2} \\ &= \frac{d(v')}{4} + \frac{E(|l|)}{2}. \end{aligned}$$

We can now calculate $E(|l| \mid w(d) = t)$ using the following equality (which we argue is correct in the next paragraph):

$$\begin{aligned} & E(|l| \mid w(d) = t) \\ & = \frac{1}{d+1}\Big[t + \frac{d+t}{2} E(|l| \mid w(d-1) = t-1) + \\ & \qquad \frac{d-t}{2} E(|l| \mid w(d-1) = t+1)\Big]. \end{aligned}$$

Since this expectation depends solely on the values of d and t, at the outset of the algorithm, we can construct a table, *table1*, such that $\mathit{table1}[d,t] \leftarrow E(|l| \mid w(d) = t)$, for all $0 \leq t \leq d \leq \Delta(G)$. Although this table will be used for looking up expectations for different values of d and t throughout the execution of the algorithm, the table entries corresponding to given values of d and t never change.

Observe that the above equality holds by the following argument. First note that

$$E(|l| \mid w(d) = t) = \frac{1}{d+1} \sum_{r'=0}^{d} E(|w(r')| \mid w(d) = t),$$

since $l = w(r)$ and r is uniformly distributed between 0 and d. Two main cases arise:

1. $r' = d$:
$$\frac{1}{d+1} E(|w(d)| \mid w(d) = t) = \frac{1}{d+1} t.$$
2. $r' < d$: consider two subcases.
   (a) The class of walks where the last step is a $+1$ step:
   The probability that we are not sampling at $r' = d$ is $d/(d+1)$. Since we are conditioning the expectation on the last step being a $+1$ step, we multiply by the probability of the last step being $+1$, which is $\frac{d+t}{2d}$. (There are d steps, of which $\frac{1}{2}(d+t)$ are $+1$ steps.) Finally, $E(|l| \mid w(d-1) = t-1)$ is the average over all walks which take a $+1$ step in the last step.
   (b) The class of walks where the last step is a -1 step:
   This case is symmetric to Case 2a, although it is concerned with -1 steps rather than $+1$ ones. □

Theorem 8 *The running time of Algorithm D(G) is* $O(|A||V|)$.

Proof We want to be able to calculate $E(|\hat{A}| \mid \text{procd}(v_1,\ldots,v_{i-1},v))$ quickly. Recall that in the proof of Theorem 7, we introduced *table1* and explained that it could be constructed at the outset of the algorithm and would henceforth be read-only. The construction takes time $O((\Delta(G))^2)$, since the entries can be built up inductively, and lookup in *table1* takes time $O(1)$. We will also introduce another table, *table2*, such that, at the ith step of the algorithm, $table2[v,v']$ will contain

$$\begin{aligned} &E(\rho_{\hat{A}}(v') \mid \text{procd}(v_1,\ldots,v_{i-1},v)) \\ &= \frac{d(v')}{4} + \frac{E(|l| \mid w(d)=t)}{2}, \end{aligned}$$

for all $v, v' \in V$. At the outset of the algorithm, we initialize *table2* so that

$$table2[v,v'] \leftarrow E(\rho_{\hat{A}}(v') \mid \text{procd}(v)).$$

This takes time $O(|V|^2)$ since retrieving $E(|l| \mid w(d)=t)$ merely requires a lookup in *table1*. Furthermore, we will maintain an array *sum* such that, at the ith step of the algorithm, $sum[v]$ contains

$$\sum_{v''\in\{v_1,\ldots,v_{i-1},v\}} \rho_{\hat{A}}(v'') + \sum_{v'\in V-\{v_1,\ldots,v_{i-1},v\}} table2[v,v'],$$

for all $v \in V - \{v_1,\ldots,v_{i-1}\}$. At the outset of the algorithm,

$$sum[v] \leftarrow \rho_{\hat{A}}(v) + \sum_{v'\in V-\{v\}} table2[v,v'],$$

for all $v \in V$. This takes time $O(|V|^2)$. At the ith step of the algorithm, we will pick the $v \in V - \{v_1,\ldots,v_{i-1}\}$ for which $sum[v]$ is a maximum. Suppose we pick a given v at step i. We must then update various entries of *table2* and *sum*. For each v' adjacent to v and for each $v_j \in V - \{v_1,\ldots,v_{i-1},v\}$, we do the following: we must first update $table2[v_j,v']$; then, we must update $sum[v_j]$ by subtracting out the old value of $table2[v_j,v']$, adding in the new one, and adding or subtracting a 1 depending on whether v was a neighbor whose removal makes $\rho_{\hat{A}}(v_j)$ grow or shrink. Over all steps of the algorithm, updating takes time $O(|A||V|)$ altogether. Since this dominates the other terms required for initialization, our proof is complete. □

7 The Upper Bound Construction

Here we demonstrate the existence of a class of arbitrarily large graphs without two-cycles that have an acyclic subgraph of size at most $(1/2 + O(1/\sqrt{\Delta(G)}))|A|$, for any $\Delta(G)$, using a result of Spencer [S], which was later simplified by de la Vega [D].

Although they were not explicitly concerned with the maximum acyclic subgraph problem, Spencer and de la Vega have proven the following theorem about random tournaments.

Theorem 9 ([S],[D]) *A random tournament* $G = (V,A)$ *has an acyclic subgraph of size at most* $(1/2 + O(1/\sqrt{|V|}))|A|$, *with high probability. Note that here* $|A| = \binom{|V|}{2}$.

This shows that our bounds are tight when $\Delta(G) = |V| - 1$. We now use Theorem 9 to show they are tight for all $\Delta(G)$.

Theorem 10 *A class of arbitrarily large graphs exists that has an acyclic subgraph of size at most* $(1/2 + O(1/\sqrt{\Delta(G)}))|A|$, *for any* $\Delta(G)$.

Proof First find a random tournament on $\Delta(G)$ vertices, for any desired $\Delta(G)$. By Theorem 9, with high probability this tournament has an acyclic subgraph of size at most

$$(1/2 + O(1/\sqrt{\Delta(G)}))\binom{\Delta(G)}{2}.$$

Now for any $n \geq 1$ construct a graph $G_n = (V_n, A_n)$ which is the union of n disjoint copies of the above random tournament. Then, G_n has an acyclic subgraph of size at most $(1/2 + O(1/\sqrt{\Delta(G)}))|A_n|$, where $|A_n| = n\binom{\Delta(G)}{2}$. Since such a graph G_n can be generated for any $\Delta(G)$, we have proven the desired result. □

8 Acknowledgements

We thank Michelangelo Grigni, David Shmoys, Joel Spencer, and Éva Tardos for helpful discussions on this work. We also thank Éva Tardos for introducing us to the problem. We especially thank David S. Johnson, John Rompel, and David Shmoys for helpful critiques of drafts of this paper. We gratefully acknowledge Joel Spencer for pointing out that our initial upper bound of $(1/2 + O(\sqrt{\log\Delta(G)}/\sqrt{\Delta(G)}))|A|$ on the size of an acyclic subgraph of a random tournament had already been improved to $(1/2 + O(1/\sqrt{\Delta(G)}))|A|$.

References

[BR] Berger, B., J. Rompel, *Simulating* $(\log^c n)$*-wise Independence in NC*, 30th IEEE Symposium on Foundations of Computer Science, 1989.

[D] De la Vega, W.F., "On the maximum cardinality of a consistent set of arcs in a random tournament," *J. Combin. Theory, Ser. B* 35, 1983, pp. 328-332.

[GJ] Garey, M.R., D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1981.

[K] Karp, R.M., "Reducibility among combinatorial problems." In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, eds., Plenum Press, 1972, pp. 85-103.

[LR88] Leighton, T., S. Rao, *An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms* 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 422-431.

[LR89] Leighton, T., S. Rao, personal communication.

[PY89] Papadimitriou, C. H. and M. Yannakakis, "Optimization, Approximation, and Complexity Classes," manuscript. An earlier version, which does not contain the relevant result, appeared in *Proceedings 20th Annual ACM Symposium on Theory of Computing,* 1988, pp. 229-234.

[Rag] Raghavan, P., "Probabilistic construction of deterministic algorithms: approximating packing integer programs," *JCSS* 37, 1988, pp. 130-143.

[Ram] Ramachandran, V., "Finding a minimum feedback arc set in reducible flow graphs", J. Algorithms, to appear.

[S] Spencer, J., *Ten Lectures on the Probabilistic Method,* SIAM, Philadelphia, 1987.