Approximations to the **NP**-complete Maximum Acyclic Subgraph Problem

Alex Francis

Berkeley, California, United States

Abstract

Maximum Acyclic Subgraph (MAS): Given a directed graph $G = (V = \{1, ..., n\}, E)$, find an ordering of nodes $r_1, ..., r_n$ (we call r_i the *rank* of player i) that maximizes the number of forward edges.

$$\text{(objective value achieved by ranking } \{r_1, ..., r_n\}) = \sum_{(i,j)\in E} \mathbb{1}\{r_i < r_j\}$$

The above problem is a known **NP**-complete reduction (proof below) with an abundance of practical applications (only one of which is discussed in detail here). The question, as with any problem in this daunting domain, is how to get reasonable results without spending unreasonable amounts of time computing. Approximation algorithms serve as a primary tool for tackling such problems. The performance of these approximations is discussed both when applied exclusively and in conjunction. The end of this brief paper serves as a cursory insight into the potential of MAS in solving the motivating problem, ranking teams in a tournament.

Keywords: Maximum Acyclic Subgraph, **NP**-completeness, Simulated Annealing, Approximation Algorithms

1. Proof of NP-Completeness

As with any problem of this nature, let's first prove that the problem lies in **NP**-complete via a reduction from the Vertex Cover problem.

Proof. We reduce from the vertex cover problem. Consider a vertex cover given by $\phi = (G, k)$. Recall that this instance has a solution if and only if there is some subset of vertices $S \subseteq V$ such that $|S| = k$ and every edge in

E has at least one endpoint in S. Formally, we define the maximum acyclic subgraph instance constructed by the reduction as $f(\phi) = (G' = (V', E'), k')$, where

$$\begin{aligned}
V' &= \bigcup_{v \in V} \{v^{(0)}, v^{(1)}\} \\
E' &= \Big[\bigcup_{u \in V} \{(v^{(0)}, v^{(1)})\} \Big] \cup \Big[\bigcup_{(u,v) \in E} \{(v^{(1)}, u^{(0)}), (u^{(1)}, v^{(0)}\} \Big] \\
k' &= |E'| - k
\end{aligned}$$

Hence, in G', we have two vertices for each vertex in G. We organize them into a bipartite graph, such that for each vertex $v \in V$, we have an edge $(v^{(0)}, v^{(1)})$ (which we will call a "right") edge in E'. Additionally, for each edge $e \in E$, we have two "left" edges $(u^{(1)}, v^{(0)})$ and $(v^{(1)}, u^{(0)})$ in E'. We observe that for every $v^{(0)} \in V'$, there is exactly one right edge leaving $v^{(0)}$ and for every $v^{(1)} \in V'$, there is exactly one right edge entering $v^{(1)}$. Furthermore, we observe that a ranking of the vertices with $|E'| - k$ forward edges exists if and only if it is possible to turn G' into a directed acyclic graph by removing k edges from the graph (since all non-forward edges can be removed to make the graph a DAG). Thus, to prove that this reduction holds, we must show that G has a vertex cover of size k if and only if it is possible to remove k edges from G' to make it a DAG. We prove the two directions separately. Note that we are reducing vertex cover to the search formulation of the Maximum Acyclic Graph problem, which in turn reduces to the optimization version via binary search in polynomial time.

If G has a vertex cover of size k, then let $S = \{v_1, v_2, ..., v_k\}$ be that vertex cover. We claim that by removing the edges $(v_1^{(0)}, v_1^{(1)}), (v_2^{(0)}, v_2^{(1)}), ..., (v_k^{(0)}, v_k^{(1)})$, we make G' acyclic. To show this, we will prove that every cycle in G' contains at least one of these edges. This is true because any cycle in G' must contain some "right" edge, since the graph is bipartite. Consider any "left" edge in a cycle. which must be of the form $(u^{(1)}, v^{(0)})$. Since the only edge coming into $u^{(1)}$ is $(u^{(0)}, u^{(1)})$, and the only edge coming out of $v^{(0)}$ is $(v^{(0)}, v^{(1)})$, we know that both of these edges must be in the cycle. The existence of this edge in E' implies that $(u, v) \in E$. Since S is a solution to the vertex cover problem, we know that either $u \in S$ or $v \in S$. This means that we removed either $(u^{(0)}, u^{(1)})$ or $(v^{(0)}, v^{(1)})$. In either case, the resultant graph does not contain this cycle. Since this is true for every cycle, the resultant graph is acyclic.

If G' can be made into a DAG by removing k edges, then let those edges be $\{e_1, ..., e_k\}$. For each e_i of the form $(u^{(1)}, v^{(0)}$, we know that any cycle through e_i must also go through $e'_i = (v^{(0)}, v^{(1)})$, as $v^{(0)}$ has only one outgoing edge. Thus, if we remove e'_i instead of e_i, the resultant graph is still acyclic. This means that we can make G' into a DAG by removing k edges of the form $(v^{(0)}, v^{(1)})$. Now, we can construct a vertex cover solution S by taking each v such that we removed the edge $(v^{(0)}, v^{(1)})$. To show that this is a valid vertex cover solution, we observe that every edge (u, v), produces the cycle $(u^{(0)}, u^{(1)}), (u^{(1)}, v^{(0)}), (v^{(0)}, v^{(1)}), (v^{(1)}, v^{(0)})$ in G'. Since we know that we can make G' acyclic by removing edges k of the form $(v^{(0)}, v^{(1)})$, it must be the case that we remove either $(u^{(0)}, u^{(1)})$ or $(v^{(0)}, v^{(1)})$. By construction of our vertex cover solution, that means that either u or v is in the vertex cover S. Since this is true for every $(u, v) \in E$, every edge has at least one endpoint in the vertex cover. $\square$

2. Motivating Question

It is often difficult to bridge the gap between theoretical formulations like that of the Maximum Acyclic Subgraph problem and applications of computer science and statistics. However, for MAS there is an intuitive motivating example that provides us with a glimpse of the usefulness of solving this problem quickly and effectively. Naturally, MAS provides an intuitive ranking of nodes based on inherent transitive properties in the connectivity of the graph itself. Specifically, consider a tournament of k players with a history of games between all pairs (i, j) in $\{1, ..., k\}$. For the two players in this pair, you know that i has defeated j, j has defeated i, both, or neither. How can we rank the players based on this history? *Comment: As an extension, consider edge weights representing the playing history of the two teams, where the number of games u has won against v is represented by the integer edge weight $w(u, v)$, and the number of games v has won against u is represented by the integer edge weight $w(v, u)$.*

From the perspective of a sports fan, this problem seems to perform an interesting simplification over the existing methods of tiebreaking, like using measures of strength of schedule, and using specific games as a determining

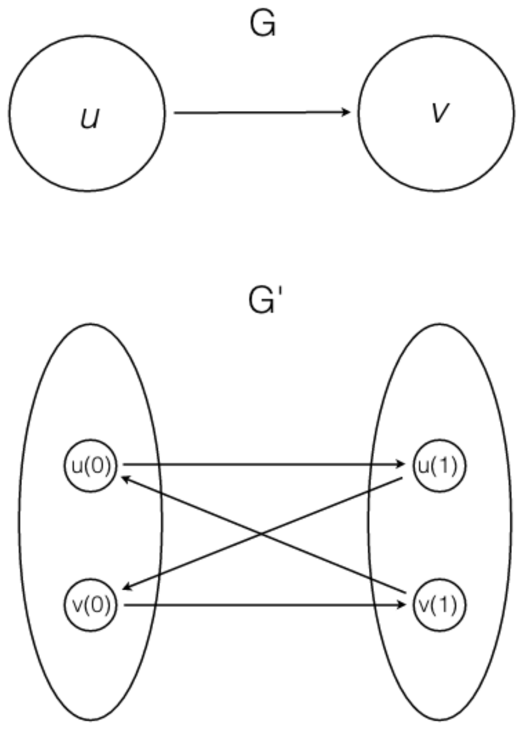

Figure 1: Bipartite graph transformation $G \to G'$

factor. Indeed, certain aspects of this ranking formulation are discussed in detail after the analysis of the algorithms themselves. Finally, the rankings of the 32 NFL teams in the context of the 2016 season is considered, and certain characteristics of the Maximum Acyclic Subgraph problem are proved based on the illuminating experimental results.

3. Deterministic Algorithms

Though this problem lies in the **NP**-complete domain, there are a variety of special cases of G that are solvable in a reasonable amount of time (sometimes polynomial).

3.1. Topological Sort on Directed Acyclic Graphs

The most obvious, of course, is the case in which G is a directed acyclic graph. The algorithmic tool is topological sort, a simple depth first search in which nodes are processed in reverse sorted order of postorder number. Pseudocode is excluded in all cases here, since the project itself can be found on GitHub (`MAS-Solver`). The runtime of topological sort on DAGs of different sizes is $O(|V| + |E|)$, and the benchmarks below show the linearity of this runtime.

$\|V\|$	$\|E\|$	Score	Time (s)
4	3	3	0.006
7	8	8	0.01
20	46	46	0.0003
100	1501	1501	0.0037
1000	149517	149517	0.3843

3.2. Brute Force

The brute force implementation is surprisingly useful on small graphs. The runtime of this algorithm is $O(n^2 \cdot n!)$, by a simple combinatorial argument. The Python `itertools` library is utilized to loop through all permutations of vertices ($n!$ of them), and the scoring function runs in $O(n^2)$. The resulting function is not particularly useful on inputs above a size of ten, as the benchmarks (and runtime analysis explained above) indicate quite clearly.

$\|V\|$	$\|E\|$	Score	Time (s)
2	1	1	0.0001
4	4	4	0.0002
8	16	14	0.6815
10	31	23	57.4857

3.3. Planar Graphs

Finally, a more sophisticated argument can be used to prove that the Maximum Acyclic Subgraph is in **P** when the graph is planar. First, let's introduce the concept of a planar graph.

Definition A *planar* graph is any graph that can be drawn in a plane without graph edges crossing.

To explore the connection between planarity and linearizing a graph, consider some significant properties of planar graphs.

3.4. Solving the Minimum Feedback Arc Set Problem via Integer Linear Programming

There exists an dual problem to MAXIMUM ACYCLIC SUBGRAPH, called MINIMUM FEEDBACK ARC SET, that is useful in solving this problem. Obviously, this problem is also **NP**-complete, but it is interesting to compare the deterministic algorithms above (particularly brute force) with another strategy for solving MAXIMUM ACYCLIC SUBGRAPH. The MINIMUM FEEDBACK ARC SET is posed in the following manner.

> In a directed graph, $G = (V, E), F \subset E$, F is called a feedback arc set if $G \backslash F$ is a directed, acyclic graph. The feedback arc set with minimum $|F|$ is the minimum feedback arc set.

A known deterministic algorithm exists for the MAXIMUM ACYCLIC SUBGRAPH that involves another **NP**-complete problem - INTEGER LINEAR PROGRAMMING. This problem is stated below.

> In the INTEGER LINEAR PROGRAMMING problem, we are given a system of inequalities, including a matrix of coefficients, A, with a vector of constraint coefficients $\mathbf{b}$, an objective function with coefficients $\mathbf{c}^T$, and a slack vector[1], $\mathbf{s}$. We're asked to find the value of a vector $\mathbf{x} = \{x_0, ..., x_n\}$. In standard form[2],
>
> $$\begin{aligned} \text{max} \qquad & \mathbf{c}^T x \\ \text{with constraints} \qquad & A\mathbf{x} + \mathbf{s} = b \\ & s \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

INTEGER LINEAR PROGRAMMING reduces to this analogous problem, confirming that this algorithm is **NP**-complete, and also providing context for the algorithm used for computation in this case.

Proof. We would like to reduce the INTEGER LINEAR PROGRAMMING problem to finding the MINIMUM FEEDBACK ARC SET on a general graph G. $\square$

[1]Slack vectors are specific to the process of standardizing an ILP from a system of inequalities to a system of equations, and is not explained in more depth here, as it has little relevance to the analysis.

[2]Since all linear programs can be reduced to this standardized form, this generalization permeates through the remainder of this section.

The speeds of this algorithm are comparable to brute force, and are documented below. Once again, the algorithm is extremely inefficient on graphs with size greater than 10.

$\|V\|$	$\|E\|$	Score	Time (s)
2	1	1	0.0001
4	4	4	0.0002
8	16	14	0.6815
10	31	23	57.4857

4. Approximations

A variety of approximation algorithms, with potentially non-optimal solutions and polynomial runtimes, are quite useful in solving this **NP**-complete problem.

4.1. Two Approximation

A variety of trivial approximation algorithms with an approximation ratio of $\frac{1}{2}$ exist for this problem. Two of them are implemented in this project. They provide fairly good approximations for most graphs. Unfortunately, the literature suggests that this is the best approximation possible for this **NP**-complete problem, assuming $\mathbf{P} \neq \mathbf{NP}$. The proof is excluded, but can be found in the reading in the footnote. The approximation algorithms utilized to solve this problem are proved, with ratio and the runtime, in the following sections.

4.1.1. Greedy Approximation

This algorithm is the most simple of the approximation algorithms, and has the worst average case results. The strategy proceeds as follows:

> Choose a random ordering of the n nodes in the graph. Then, reverse that random ordering. Score each and choose the best of the two. The better ordering is guaranteed to be a two approximation.

The proof of the approximation ratio is below.

Proof. Assume for the sake of contradiction that the greedy algorithm does not provide a two approximation on the graph G. If the algorithm does not provide a two approximation, the the sum of the forward edges in forward order is less than $x < 1/2|E|$. By symmetric logic, the sum of forward edges in the reverse order is also less than $y < 1/2|E|$. These edges must account for all $|E| \leq |V|^2$ edges in the graph, but $x + y < |E|$, which is a contradiction. □

The runtime of this algorithm is simply the runtime of the scoring function, which is $O(|V|^2)$, since the nature of the scoring function is to check all nodes forward from the currently examined node in the ordering provided to it, looping over the all of V in the outer scope.

4.1.2. Divide and Conquer Approximation

Examining the greedy approach, there are ways that we can improve upon the progress that it provided. Specifically, let's apply the same strategy in a divide and conquer fashion across G. The strategy is detailed below:

> Divide a graph $G = (V, E)$ into two non-empty sets A, B. Examine the edges between A and B and examine the edges between B and A. Call the magnitudes of the set of edges $|E_{a,b}|$ and $|E_{b,a}|$, respectively. Discard the edges in the smaller set. Recurse on the sides A and B until the topological sorting algorithm can be run.

The proof proceeds in two parts, first noting that the algorithm will always terminate (eventually, the graph must be a DAG), and then examining the approximation ratio,

Proof. First, let's prove that the algorithm will always terminate.

The approximation ratio is straightforward to prove. The optimal number of forward edges is at most $|E|$, the total number of edges in G. During each iteration of the divide and conquer algorithm, the edges between the cut. As a result, the resulting edge set, E', is at most halved. Therefore, for the algorithm $\mathcal{A}$,

$$\alpha_A = \max_I \frac{\mathcal{A}(I)}{\text{OPT}(I)} = \frac{|E'|}{|E|} = \frac{1/2|E|}{|E|} = 1/2$$

□

The results of using this algorithm are attached below. Note that the score for the divide and conquer approximation tends to be better than that of the greedy strategy for the same graphs, but tends to have significantly worse performance on large graphs.

$\|V\|$	$\|E\|$	Score	Time (s)
4	4	4	0.1324
25	170	120	0.7196
100	2922	1773	8.5191

Table 1: Simulated Annealing with 4000 iterations

$\|V\|$	$\|E\|$	Score	Time (s)
4	1	1	0.0001
25	4	4	0.0002
100	2922	14	0.6815

Table 2: Simulated Annealing with 10000 iterations

$\|V\|$	$\|E\|$	Score	Time (s)
4	1	1	0.0001
25	4	4	0.0002
100	2922	14	0.6815

4.2. Simulated Annealing

Simulated annealing is a search technique derived from the process of crystallization, acclaimed for avoiding non-absolute maxima despite its hill-climbing strategy. Specifically, simulated annealing initializes a "tempera- ture," T, which steadily decreases to 0 at a rate of $-\Delta T$. These constants are fixed in the code (initialized in the constructor), and have been determined by a series of experiments. Some of the results of this research are explained after the delineation of the algorithmic techniques used here.

> Pick an ordering originally and score it using the functions provided by the staff. The initial ordering will be generated by the library function that calculates an approximate solution to the feedback arc set problem and by the two approximation, which will be run 10000 times on each instance.
>
> Then we flip one or two nodes in the initial ordering s randomly, generating a new ordering s', and run the scoring function. Assign $\delta = \text{score}(s') - \text{score}(s)$. If δ is negative, the new path is worse. So we choose it with (low) probability $\mathbb{P}[\text{use } s'] = e^{-\delta/T}$. If δ is positive, the new path is better. So, we replace the old solution with the new one. All the while we keep track of our best solution and its score.

4.3. Solving the Minimum Feedback Arc Set Problem via Eades Approximation

5. Application of Maximum Acyclic Subgraph

Ranking teams in a tournament. The results of running the algorithms set forth previously in this document on real-world examples, and comparisons to ranking tools currently utilized by statisticians.

6. Further Research

Probabalistic models & more.