

Ranking Tournaments: Local Search and a New Algorithm

TOM COLEMAN and ANTHONY WIRTH

The University of Melbourne

Ranking is a fundamental activity for organizing and, later, understanding data. Advice of the form “ a should be ranked before b ” is given. If this advice is consistent, and complete, then there is a total ordering on the data and the ranking problem is essentially a sorting problem. If the advice is consistent, but incomplete, then the problem becomes topological sorting. If the advice is inconsistent, then we have the feedback arc set (FAS) problem: The aim is then to rank a set of items to satisfy as much of the advice as possible. An instance in which there is advice about every pair of items is known as a tournament. This ranking task is equivalent to ordering the nodes of a given directed graph from left to right, while minimizing the number of arcs pointing left.

In the past, much work focused on finding good, effective heuristics for solving the problem. Recently, a proof of the NP-completeness of the problem (even when restricted to tournaments) has accompanied new algorithms with approximation guarantees, culminating in the development of a PTAS (polynomial time approximation scheme) for solving FAS on tournaments.

In this article, we reexamine many existing algorithms and develop some new techniques for solving FAS. The algorithms are tested on both synthetic and nonsynthetic datasets. We find that, in practice, local-search algorithms are very powerful, even though we prove that they do not have approximation guarantees. Our new algorithm is based on reversing arcs whose nodes have large indegree differences, eventually leading to a total ordering. Combining this with a powerful local-search technique yields an algorithm that is as strong, or stronger than, existing techniques on a variety of data sets.

Categories and Subject Descriptors: G.2.2 [Discrete Mathematics]: Graph Theory—*Graph algorithms*; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Computations on discrete structures, routing and layout*; G.1.6 [Numerical Analysis]: Optimization

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Local search, feedback arc set, approximation

ACM Reference Format:

Coleman, T. and Wirth, A. 2009. Ranking tournaments: Local search and a new algorithm. ACM J. Exp. Algor. 14, Article 2.6 (May 2009), 22 pages. DOI = 10.1145/1498698.1537601 <http://doi.acm.org/10.1145/1498698.1537601>

A preliminary version of this article first appeared in *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX) 2008*.

This work was supported by the Australian Research Council, through Discovery Project grant DP0663979.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2009 ACM 1084-6654/2009/05-ART2.6 \$10.00
DOI 10.1145/1498698.1537601 <http://doi.acm.org/10.1145/1498698.1537601>

1. INTRODUCTION

1.1 The Feedback Arc Set Problem

The feedback arc set (FAS) problem is a key combinatorial problem to rank items in a set given only advice about the correct way to order specific pairs. A ranking π of a set is simply a permutation of that set. Thus, the only information we have to help us to form our ranking is a set of statements of the form “ a should be ranked before b .”

If each statement is consistent with all others, the problem is simple to solve—just return a ranking that agrees with all the advice. However, difficulties arise when there is contradictory, or inconsistent, advice. For instance, given the three statements, “ a should be ranked before b ”, “ b should be ranked before c ” and “ c should be ranked before a ,” there is no ranking of a , b , and c , which agrees with all the statements.

The difficulty of the FAS problem, much like problems of clustering with advice—such as the correlation clustering problem [Bansal et al. 2004]—is in deciding which of the inconsistent advice to follow, and which to violate. The aim is to minimize the number of statements that are violated.

The natural graph representation of the problem uses a vertex for each item and a directed arc from a to b for each demand “ a should be ranked before b .” In this context, the aim is to order the vertices from left to right so that the number of arcs pointing left (back-arcs) is as small as possible.

Problem: FAS

Given a directed graph $G = (V, E)$, find an ordering π over V to minimize the number of back arcs, that is, the number of arcs

$$\{v \rightarrow w \in E \mid \pi(v) > \pi(w)\}$$

Given some ranking π , if we remove the set of back-arcs, we will eliminate all cycles in the graph. We call such a set a FAS. An equivalent formulation of the problem is therefore: Given a digraph G , find the smallest subset S of the arcs of G that intersects all cycles in G . The graph G' obtained by removing the arcs in S from G is acyclic—and thus a DAG—and admits a consistent ordering via a topological sort.

In this article, we focus on the special case of tournament graphs, in which there is an arc between each pair of nodes. We also consider weighted tournaments, in which the weights on arcs $u \rightarrow v$ and $v \rightarrow u$ must sum to 1.

1.2 Some Applications of FAS

Originally motivated by problems in circuit design [Johnson 1975], FAS has found applications in many areas, including computational chemistry [Klein and Randić 1987; Pachter and Kim 1998] and graph drawing [Eades and Wormald 1994].

Closely related to the FAS problem is the rank aggregation problem.

Problem: Rank Aggregation

Given a set Π of rankings over a set V , find a ranking σ to minimize $\sum_{\pi \in \Pi} K(\sigma, \pi)$, where $K(\sigma, \pi)$ is the Kemeny distance [1959]; defined to be the number of pairs $v, w \in V$ where $\pi(v) < \pi(w)$ and $\sigma(v) > \sigma(w)$.

Dwork et al. [2001] outline the problem and motivate it as a method for aggregating data from search engines. There is a significant body of work studying this problem, which is known as MetaSearch [Aslam and Montague 2001].

Rank aggregation is a special case of FAS on weighted tournaments. There is a fairly simple reduction: For each $v, w \in V$, if v is ranked above w in some fraction f of the input rankings, place an arc between v and w with weight f . The connection to rank aggregation is a principal reason for the attention paid to the FAS problem on tournaments.

1.3 The Linear Ordering Problem

Another similar problem to FAS is linear ordering.

Problem: Linear ordering

Given a square matrix (c_{ij}) for $i, j \leq n$, find a ranking π over $[n]$ to minimize:
$$\sum_{\substack{i,j \\ \pi(i) < \pi(j)}} c_{ij} .$$

The weighted tournament version of FAS is a special case of linear ordering, under the constraint that $c_{ij} = 1 - c_{ji}$ and $c_{ij} \in [0, 1]$.

1.4 Overview of Heuristics

The problem of FAS is familiar to organizers of sporting leagues—often each team plays each other once, and a ladder must be constructed, which ranks the teams as fairly as possible. In this context, it is natural to ask to minimize the number of upsets as a measure of “fairness.” The standard solution to this problem is—unsurprisingly—a simple one. We rank each team by the number of wins it has achieved. In the graph version of the problem, where a match is represented by an arc pointing from the loser to the winner, this is simply the indegree. This means that the best teams (vertices) will be placed on the right-hand side when we order by indegree. The indegree has traditionally been given another name, after Kendall [1955], the first to propose this algorithm to solve FAS. The Kendall score of a vertex v is simply v ’s indegree—that is the number of other vertices x , such that we have a statement of the form “ x should be ranked before v .”

The one complication to ordering by Kendall score is in resolving the issue of ties—what happens if two teams have the same number of wins? Sporting leagues tend to use domain-specific solutions to the problem (e.g., goal difference). Ali et al. [1986] and Cook et al. [1988] propose a more general

solution: For a set of vertices T of the same Kendall score, break the tie by recursing into the subgraph induced by T . We refer to this algorithm as the iterated Kendall algorithm.

Eades et al. [1993] created an algorithm that is in fact quite similar to iterated Kendall, possibly inspired by selection sort. Again, we choose the vertex v of lowest Kendall score to place on the left-hand side of the ranking. The difference is that we then recompute Kendall scores for the remaining vertices by considering the subgraph remaining after dropping v .

Chanas and Kobylanski [1996] presented an algorithm for the linear ordering problem (and thus also for FAS) based on repeated application of a procedure analogous to insertion sort. In Section 2.2, we outline the algorithm and compare it to other sort-based methods.

Saab [2001] presents an algorithm using a divide-and-conquer approach. The idea is to split the input into two halves, minimizing the number of back-arcs between the halves and then recurse on each half. However, this minimization task is a difficult one: It is a directed version of the min bisection problem, and solving it would solve FAS. We do not explore this algorithm further.

1.5 Overview of Hardness and Approximation

The FAS problem is one of Karp's [1972] original NP-complete problems (in the general case). Dwork et al. [2001] proved the rank aggregation problem is NP-hard, even with as few as 4 input rankings. The FAS problem has now been shown to be NP-hard, even on unweighted tournaments [Charbit et al. 2006]. Given this, it is natural to ask for an approximation algorithm, which runs in polynomial time, yet is guaranteed to differ in cost from the optimal solution by a small factor.

Recently, there has been a flurry of activity in the approximation algorithms community focused on the FAS problem for tournaments. The first constant factor approximation algorithm for FAS, designed primarily for tournaments, was the pivoting algorithm of Ailon et al. [2005]. Intuitively similar to quicksort, it uses on average $O(n \log n)$ comparisons for an expected 3-approximation on tournament graphs.

Coppersmith et al. [2006] showed that ordering the nodes by their Kendall score is a 5-approximator on tournament instances. This of course includes the iterated Kendall algorithm. Interestingly, in terms of approximation guarantees, the method chosen to resolve ties is irrelevant. So an arbitrary choice is as good as the full recursive nature of the iterated Kendall algorithm.

Kenyon-Mathieu and Schudy [2007] completed the approximation picture for tournaments with a PTAS (polynomial-time approximation scheme).¹ This scheme comprises a local search heuristic—which we call MOVES, and investigate in isolation below—and the PTAS of Arora, Frieze, and Kaplan [2002] for dense instances of the maximum acyclic subgraph problem.

¹We did not implement this algorithm during this experimental evaluation. Although theoretically very powerful, the algorithm is complicated and impractical to implement.

1.6 Our Contributions/Article Organization

In Section 2, we provide details of the algorithms we will test. Some are existing procedures, others are our improvements to them, still others are based on our scheme of reversing arcs (in an organised manner) to destroy directed triangles, and thus produce a total ordering.

We then show in Section 3 that some of the existing heuristic algorithms are not guaranteed approximators for the FAS problem.

We tested all algorithms on not only synthetic data (as has generally been the method of testing heuristics in the past), but also on two sets of FAS problems generated from rank aggregation data. Our experiments focus on both the performance of the algorithms and on their running times. This work is outlined in Section 4.

The results of these tests and further analyses are presented in Section 5. We conclude and supply ideas for further work in Section 6.

2. ALGORITHM DETAILS

In this section, we provide detailed descriptions of the algorithms that we will investigate, some of which we have already discussed.

2.1 Kendall Score-Based Algorithms

We remind the reader that the Kendall score of a vertex is simply its indegree. Two fundamental algorithms are:

Algorithm: Iterated Kendall

Rank the nodes by their Kendall scores, with lowest Kendall score on the left. If there are nodes with equal score, break ties by recursing on the subgraph defined by these nodes. If there is a subgraph whose elements all have equal indegree, rank them arbitrarily.

Algorithm: Eades

Select the node that has the smallest Kendall score and place it at the left, breaking ties arbitrarily. Recurse on the remainder of the nodes, having recomputed the indegrees on the remaining subgraph.

The Eades algorithm focuses on the left side of the ranking. We improve this by allowing the selection of a vertex to either end of the ranking. For instance, if there is a vertex v of extremely low outdegree, but no vertex of low indegree, it makes sense to decide to place v on the right-hand side of the ordering, rather than a less appropriate vertex on the left-hand side.

Let $\text{In}(v)$ stand for the indegree of node v , with $\text{Out}(v)$ its outdegree. Our improved algorithm is:

Algorithm: Eades Improved

Select the node u that maximizes $|\text{In}(u) - \text{Out}(u)|$ and place it at the left end if $\text{In}(u) < \text{Out}(u)$, otherwise, the right end. Recurse on the subgraph induced by removing u . Again, break ties arbitrarily.

2.2 Sorting Algorithms

As suggested earlier, Ailon et al.'s [2005] algorithm is much like the classic quicksort algorithm for sorting. In fact, it is possible to define a FAS algorithm, which is analogous to almost any sorting algorithm. Unlike traditional sorting problems, in which we assume there is a total order on the data, the difficulty in FAS is the lack of transitivity, which sorting algorithms are designed to exploit. Nevertheless, sorting algorithms provide schemes for deciding which of the advice to believe.

So we can define a general strategy for FAS based on some sorting algorithm S ; run S over the vertices of G , using as the comparison function “ $u < v$ if and only if $u \rightarrow v$.”

For instance, using this strategy, quicksort is defined as follows:

Algorithm: Quicksort

Choose a pivot $p \in V$, uniformly at random. Let $L \subseteq V$ be all vertices v such that $v \rightarrow p$, and let $R = V \setminus (L \cup \{p\})$. Also, let π_L be the ordering of L obtained by quicksort, and π_R be the analogous ordering of R . Output (π_L, v, π_R) , the ordering resulting from placing vertices in L on the left (ordered by π_L), etc.

Cook et al. [1988] focus on ensuring a Hamiltonian path exists along the final ordering of the nodes; any sensible algorithm should achieve this. The method they use to achieve this is in effect a bubble sort of the tournament. As noted previously, the Eades algorithm is the obvious analogy to selection sort, with our improved version being a two-sided selection sort. To our knowledge, no article has studied the MERGESORT approach; in this article, we test its performance.

Chanas and Kobylanski [1996] apply an insertion technique to the linear ordering problem that is more involved than the usual INSERTIONSORT. As a subroutine, they use what is in effect insertion sort, which they name *sort*. It is defined as follows:

Algorithm: Sort

Make a single pass through the nodes from the left to the right. As each node is considered, it is moved to the position to the left of its current position that minimizes the number of back-arcs (if that number of back arcs is fewer than its current position).

Since executing *sort* cannot increase the number of back-arcs, the authors first propose an algorithm *sort**, which repeatedly applies *sort* until there is no improvement in the number of back-arcs. They also show that the composition of two steps *sort* \circ *reverse* (where *reverse* simply reverses the order of the nodes) cannot increase the number of back-arcs. The Chanas algorithm is therefore $(\text{sort}^* \circ \text{reverse})^*$, which the authors have demonstrated outperforms *sort** alone on general graphs.

2.3 Local Search Algorithms

One approach that has been used successfully for many optimization problems is to begin with some solution and then iteratively improve that solution until no further improvement is possible. Researchers have met with success in proving approximation bounds for local search schemes for clustering problems. These include k -median [Arya et al. 2004], k -means [Kanungo et al. 2004] and, recently, correlation clustering [Coleman et al. 2008], which is similar in structure to FAS.

The general local search scheme that we will consider is:

Given some solution π , consider all potential local improvements that could be made to π . Choose as a new order π' , the local improvement, which minimizes the cost, as long as that cost is strictly less than π 's. If no such improvement exists, return π —otherwise repeat.

The important point here is to decide which permutations are local improvements of some ranking π . Here, we consider two such local improvement schemes for FAS:

- The swaps heuristic, which swaps the position of two nodes in the order.
- The moves heuristic, which moves one node to any position in the order, leaving the relative order of the other nodes unchanged.

In this article, we show that neither algorithm can provide an approximation guarantee. On preliminary tests, we found that the moves heuristic performed well, but that swaps did not; the latter was omitted from further experiments.

One particular advantage of a local search technique is that the initial solution it is given can be the output of an approximation algorithm. Consequently, the local search approach inherits the approximation guarantee.

The Chanas algorithm. An application of the sort step of Chanas has the effect of checking, for each vertex of the graph, from left to right, if a move to the left is possible. This is essentially a scheme for selecting which moves-style changes to make. The operation $\text{sort} \circ \text{reverse}$ does the same thing, but with moves to the right. So, Chanas is simply a method for investigating moves in a particular order. We developed an alternative algorithm:

Algorithm: Chanas Both

Run CHANAS, but change the sort procedure, so it is allowed to move a node either left or right, to the position that results in the fewest back-arcs.

A consequence of this modification is that some nodes may be moved more than once in a single sort pass.

2.4 Triangle-Destroying Algorithms

A tournament has a cycle if and only if it has a directed triangle ($\vec{\Delta}$). This is due to the complete nature of a tournament graph—every (non- Δ) cycle must have a chord inside it that forms a strictly smaller cycle. We, therefore, considered

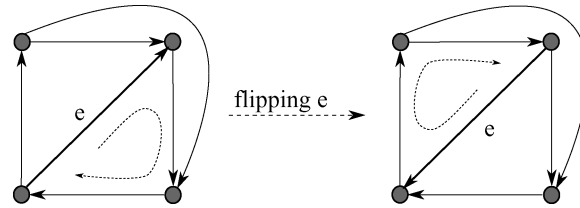


Fig. 1. An example where reversing creates a triangle while destroying another.

algorithms that destroy directed triangles by selecting arcs to reverse. It might seem more natural to delete arcs, but this would make the digraph no longer a tournament, creating the possibility of cycles without the presence of $\vec{\Delta}$ s. Our scheme works in the following way:

Algorithm: Triangle Deletion

While the tournament is not acyclic, choose an arc and reverse its orientation. Once the tournament is acyclic, use the topological sort of the vertices as the solution to the original problem.

The choice of arc to be reversed affects the performance and running time of this procedure; the remainder of this section examines various heuristics.

We call the number of $\vec{\Delta}$ s an arc is involved in its triangle count. The triangle count of a tournament is simply the number of $\vec{\Delta}$ s in that tournament. So, the first algorithm is:

Algorithm: Triangle Count

Run triangle deletion, choosing on each iteration the arc with highest triangle count.

There is a pitfall here though: Reversing an arc can create a new $\vec{\Delta}$ that did not previously exist. In Figure 1, we see that reversing the center arc does not actually reduce the triangle count of the graph, as a single triangle is destroyed and new triangle is created. The problem here is really the length 4 cycle surrounding the arc, and the solution is to reverse one of these outer arcs (one of which in fact has triangle count two).

However, there are (more complex) examples where reversing the arc with the highest triangle count does not change the triangle count of the tournament so that arc still has the highest triangle count, leading to an infinite loop.

To avoid this problem, we define the following algorithm:

Algorithm: Triangle Delta

Run triangle deletion, choosing the arc that causes the greatest net reduction to the tournament's triangle count.

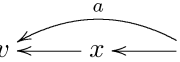
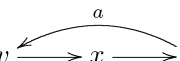
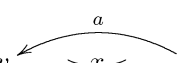
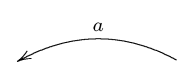
A potential problem with triangle delta could be the existence of a tournament that was not acyclic (and thus still had $\vec{\Delta}$ s, and thus a positive triangle count), yet contained no arcs whose reversal would lower the triangle count. Lemma 1 proves that this situation is impossible.

LEMMA 1. *Let T be a tournament. If T has a cycle, then there exists an arc $e \in T$ such that reversing e will reduce the triangle count of T .*

PROOF. Let σ be an ordering of the vertices of T that induces a minimum FAS. Let $a = v \leftarrow w$ be a back-arc of maximal length under σ , that is maximizing $\sigma(w) - \sigma(v)$. We claim that reversing a will lower the triangle count.

Firstly, we note that reversing a will not create any $\vec{\Delta}$ s of the form $v-w-x$, where x is to the right of both v and w , as this would imply a back-arc $v \leftarrow x$, that is “longer” than $v \leftarrow w$; this is impossible by our choice of a . Similarly, no $\vec{\Delta} x-v-w$ can be created where x is to the left of both vertices, so any $\vec{\Delta}$ created must involve an x between v and w .

Consider the four possibilities for a node x that is placed between v and w by σ :

- (1)  (reversing a will create a $\vec{\Delta}$). Say there are A such x 's.
- (2)  (reversing a will delete a $\vec{\Delta}$). B of these.
- (3)  (reversing a will have no effect). C of these.
- (4)  (no effect). D of these.

Since σ is optimal, moving v to the position after w will not reduce the back-arc count, so the number of back-arcs into v from such x 's must be less than the number of forward arcs from v to such x 's (strictly, as there is a back-arc from w to v). So we have

$$A + D < B + C.$$

Similarly, moving w before v will not improve the order, so we have

$$A + C < B + D.$$

Combining these gives

$$2A + D + C < 2B + C + D \implies A < B,$$

and, therefore, the number of $\vec{\Delta}$ s will decrease. \square

In practice, contrary to our expectations, the triangle count algorithm tended to outperform the triangle delta algorithm (except of course in the cases when it did not complete execution). So, we considered a third option, taking the best of both worlds.

Algorithm: Triangle Both

Run triangle deletion, choosing the arc with the highest triangle count, provided that it reduces the tournament's total number of $\vec{\Delta}$ s.

Note that the best algorithm we have for calculating the triangle count, and the change in $\vec{\Delta}$ s, for every arc of the digraph requires $O(n^3)$ operations.

In a weighted tournament, the weight of a $\vec{\Delta}$ is the sum of the weights of its arcs. Therefore, in such graphs, the triangle count of an arc is the sum of the weights of the Δ s it is involved in.

2.5 Degree Difference Algorithms

We designed a new algorithm, degree difference, which is again a triangle-deletion algorithm, but which selects an arc to reverse based on a criterion that is much simpler to compute than the full triangle count. However, the criterion seems empirically to be related to the triangle count.

Algorithm: Degree Difference

Run triangle deletion, choosing the arc $u \rightarrow v$ for which the difference between u 's indegree and v 's indegree is greatest.

Unfortunately, it may take $\Theta(n)$ time to find such an arc at each iteration. Nevertheless, this algorithm always makes progress towards a total ordering. The value of $\sum_v \text{In}(v)^2$ increases whenever an arc from a higher-degree to a lower-degree node is reversed and has a maximum value of $\sum_{i=0}^{n-1} i^2$ when the tournament has a total ordering.

In an effort to further speed up the degree difference algorithm, we used a sampling technique. We sample $\log n$ vertices (favoring high indegree) to potentially be the “tail” of the arc, and another $\log n$ (favoring low indegree) to potentially be the “head.” We then check each of the $\log^2 n$ arcs between sampled vertices, choosing the back-arc of highest degree difference. We resample if we find back-arcs only of nonpositive degree difference. This algorithm is called degree difference sampled 1, and it takes $O(n^2 \log^2 n)$ time, on average.

A further variation, degree difference sampled 2, maintains two lists: One of potential head nodes, and one of potential tail nodes. The idea is to try to push the quantity $\sum_v \text{In}(v)^2$ toward its maximum (which is reached when all indegrees are different).

With this in mind, a node v is a potential head if its indegree, $\text{In}(v)$, is not unique or there is no node of indegree $\text{In}(v) - 1$. Similarly, a node u is a potential tail if $\text{In}(u)$ is not unique or there is no node of indegree $\text{In}(u) + 1$. We sample $\log n$ nodes from each list uniformly and from those pairs select the arc with the largest indegree difference to reverse.

3. APPROXIMATION COUNTER-EXAMPLES

We now show that various algorithms for FAS cannot guarantee reasonable factor approximations.

All graphs shown are tournaments (complete graphs), but in the interest of readability, not all arcs are drawn. In the following figures, only back-arcs, with respect to the given ordering, are displayed. All pairs of nodes with no arc displayed are assumed to have a right-pointing arc between them.

We remind the reader that the cost of a configuration is simply the total number of back-arcs.

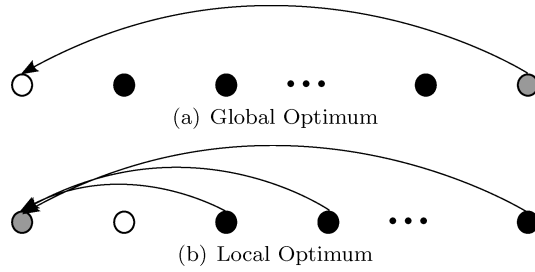


Fig. 2. Standard Bad Example: The global optimum has a single back-edge, while the local optimum (for the SWAPS heuristic) has $n - 2$. The only difference between the two is in the placement of the grey vertex—all other nodes are unchanged.

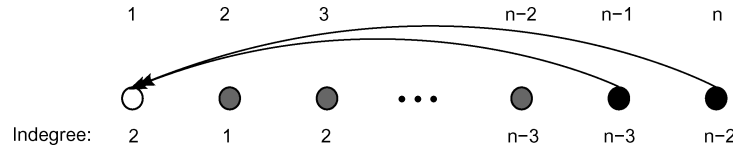


Fig. 3. The counterexample for the Eades algorithm. The global optimum is as pictured. The Eades algorithm will place the white node at the right-most position, creating $n - 3$ back-arcs, while removing only 2.

3.1 Standard Bad Example

This example consists of a completely transitive tournament of size n , with one minor perturbation—there is a single back-arc, from the last node (node n) to the first (node 1). Figure 2(a) shows this (global) optimum configuration; a local optimum for the swaps heuristic is shown in Figure 2(b), with cost $n - 2$.

Note also that there is no guarantee that bubblesort will start with the grey node placed after the white node. As bubblesort (like swaps) only ever exchanges adjacent nodes, if it starts from such a scenario, it will never reach a configuration with the white node before the grey, and thus will only reach a costly local optimum.

3.2 The Eades Algorithms

Figure 3 shows a modification to the standard bad example of Section 3.1, in which the optimum solution has two back-arcs: from nodes $n - 1$ and n to node 1. Displayed below each node is its indegree. The Eades and Eades algorithms both place node 2 at the left of their solution (as it has the lowest indegree); with that node removed, the induced subgraph is precisely the same as the original one, albeit one node smaller. The final order will, therefore, be $(2, 3, 4, 5, \dots, n - 1, n, 1)$, which has a cost of $n - 3$.

3.3 Moves and Chanas

The configuration of Figure 4(b) is a local optimum for the moves heuristic. The spacing of the back-arcs ensures that it is never an improvement to move a single node.

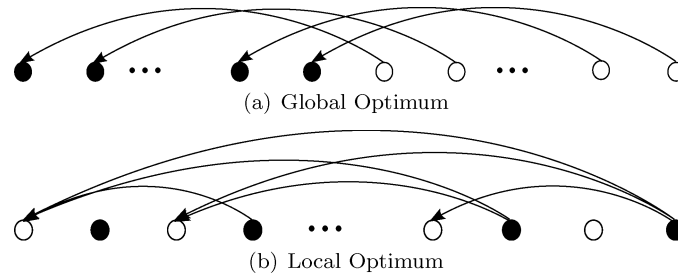


Fig. 4. Counterexample family (n even) for the moves local search heuristic (and thus for Chanas and Chanas both). Consists of two sets of $n/2$ vertices (black and white), each of which is internally transitive. (a) Shows the global optimum. The black vertices are placed before the white, and each white vertex has a single back-arc to the black vertex $n/2$ positions preceding it. (b) If we interleave the black and white vertices (each set is internally in the same order), we have a locally optimal solution. Yet now there is a back-arc from each black vertex to the white vertex that is 3, 5, 7, ... positions preceding it.

Following the discussion at the end of Section 2.3, neither the Chanas nor the Chanas both algorithms could escape from this configuration.

The cost of the local optimum is $n^2/8 - n/4$. The global minimum, shown in Figure 4(a), places all black nodes before all white nodes, without changing the relative order within the color group, thus incurring a cost of $n/2$. So, the locality gap here is in $\Omega(n)$.

4. EXPERIMENTS

We conducted a series of experiments to validate the empirical performance of these algorithms. All experiments were conducted on a 4-core Intel Xeon 3.2GHz machine, with 8GB of physical memory. All algorithms were compiled by gcc version 3.4.6 with the `-O3` optimization flag.

In order to investigate the significance of initial solution quality to the effectiveness of local search techniques, we first tested each algorithm in isolation—for the local search algorithms, this meant starting from a random ordering—and then passed the output of each algorithm into both the Chanas and moves algorithms.

Note that passing the output of Chanas as input to Chanas is a surprising case. Chanas is a local search algorithm—which would imply that as the output of Chanas is locally optimal, a further call to Chanas would have no further effect. However, a single call to the `sort* ◦ reverse` step can significantly change the ordering without affecting the solution quality. That is, while deciding that it is at a local minimum, the algorithm can take a “sideways step” (one which changes the ordering, without decreasing the cost). So, it is possible that two calls to `sort* ◦ reverse` can lower the cost of a ranking, while one will not.

Repeated calls to Chanas can sometimes move the algorithm out of a local plateau. For this reason, Chanas + Chanas can sometimes improve on, and also take longer than, Chanas alone. However, it is difficult to predict when this is going to happen, and so no systematic method to take advantage of this is apparent.

4.1 Datasets

Biased. We tested the FAS algorithms on the following synthetic dataset. Starting with a total order from nodes 1 to n , we reverse each arc independently with probability p . In particular, with $p = 0.5$, we have a completely random tournament.

The following datasets provide a set of rankings to be aggregated—the rank problem on these rankings provides us with a FAS tournament as described in Section 1.2.

WebCommunities. Our colleague, Laurence Park, provided us with a set of 9 rankings of a large set of documents (25 million) [Park and Ramamohanarao 2007]. From this, we took 50 samples of 100 documents and considered the rankings of each of those subsets.

EachMovie. We used the EachMovie collaborative filtering dataset [McJones 1997] to generate tournaments of movie rankings. The idea here was to identify subgroups (we used simple age/sex demographics) of the users, and then generate tournaments that represented the “consensus view” of those groups.

The EachMovie dataset consists of a vote (on a scale of one to five) by each user for some set of the movies. To form a tournament from a group, we took the union of movies voted for by that group and then set the arc weight from movie a to movie b to be the proportion of users who voted a higher than b . For consistency, we sampled each tournament down to size 100.

5. DISCUSSION OF RESULTS

We tested all of the algorithms on a large number of datasets. We selected just four of the data sets to display in Table I: These show a variety of performance characteristics. We test three variants of each algorithm—the performance in isolation, as well as the performance as input to both moves and Chanas. Listed for each variant is the percentage error, as compared to our baseline algorithm—Chanas in isolation. Also listed is the percentage of “wins”—that is the number of times the algorithm does the best (of the variant). If k algorithms each produce the best solution on a given input graph, each is given $1/k$ wins. Finally, the total time taken in seconds is given.

We first note the striking performance of the Chanas local search procedure, which is rarely beaten. Despite coming with an approximation guarantee, the quicksort procedure performs relatively poorly, as does the MERGESORT algorithm. Bubblesort does surprisingly well on the WebCommunities dataset, although it seems from the results that the WebCommunities dataset is a particularly simple problem. Also, bubblesort only does well after the Chanas procedure is applied; otherwise, it is possibly the worst of the algorithms. The Eades and Eades improved algorithms are strong, but there should be a slight preference for the latter due to its lower running time.

As expected, the triangle both algorithm is very slow, though it is a strong performer when combined with local search. The degree difference algorithm is similar, though at a different point on the effectiveness/speed tradeoff. The

Table I. Each Algorithm is Tested on the Biased Data Set with $p = 0.6$ and 0.95 . We also Tested each Algorithm on the WebCommunities and EachMovie Data Sets. In addition to the basic algorithms, we ran moves and Chanas-style local search procedures on the outcomes of each of these algorithms. We report the average of the percentage (%) relative difference between the number of back-edges (Errors), compared to the Chanas heuristic, over all problem instances in the dataset. We also report the percentage (%) of times each algorithm wins (produces the best amongst the solutions generated), with the win split between algorithms that are equal first on a particular instances (Wins). Finally, the average running time (in seconds) of each procedure (including the local search cleanups) is reported separately Time. Note that both Biased datasets had 100 problem instances, WebCommunities 50, and EachMovie 146

	Variant	0.6			0.95			WebCommunities			EachMovie		
		Errors	Wins	Time	Errors	Wins	Time	Errors	Wins	Time	Errors	Wins	Time
It. KEND.	—	12.02	0.0	2.1	29.79	0.0	1.9	15.63	0.0	0.1	8.39	0.0	0.3
	MOVE	0.35	7.5	4.9	0.30	10.0	3.8	0.00	12.0	0.4	0.31	7.4	0.8
	CHAN	-0.35	10.9	6.7	0.00	14.5	4.5	0.00	13.6	0.3	-0.06	7.8	1.0
EAdES	—	9.88	0.0	4.7	62.02	0.0	4.7	31.38	0.0	0.2	7.47	0.0	0.7
	MOVE	0.56	4.0	7.5	0.31	5.5	6.9	0.01	5.3	0.6	0.25	11.5	1.1
	CHAN	-0.29	9.6	9.7	0.00	7.3	7.3	-0.00	6.7	0.4	-0.08	8.0	1.4
EAdES Imp.	—	8.65	0.0	1.9	52.10	0.0	1.9	19.29	0.0	0.1	6.37	0.0	0.3
	MOVE	0.60	3.2	4.7	0.32	5.2	4.7	0.00	8.6	0.3	0.37	2.2	0.8
	CHAN	-0.20	6.9	6.8	0.00	7.3	4.7	0.00	7.0	0.3	-0.02	6.3	1.0
CHANAS	—	0.00	74.0	5.6	0.00	27.8	2.8	0.00	23.6	0.1	0.00	83.8	0.7
	MOVE	-0.01	34.1	7.5	-0.00	12.8	5.3	-0.00	9.4	0.3	-0.03	54.5	1.1
	CHAN	-0.04	6.4	7.8	-0.00	7.3	5.5	-0.00	7.0	0.3	-0.04	9.7	1.1
BUBBLE.	—	35.25	0.0	1.6	728.46	0.0	1.6	74.46	0.0	0.1	31.07	0.0	0.3
	MOVE	0.75	4.0	4.6	0.21	7.0	3.4	0.00	5.4	0.3	0.33	1.5	0.8
	CHAN	-0.02	4.4	7.2	0.00	7.2	3.6	-0.00	8.9	0.3	-0.01	7.8	1.0
MERGE.	—	23.23	0.0	1.6	130.81	0.0	1.6	0.86	1.4	0.1	20.62	0.0	0.3
	MOVE	0.79	2.7	4.6	0.23	6.4	5.2	0.00	6.0	0.2	0.34	3.4	0.8
	CHAN	-0.02	5.6	7.2	0.00	7.2	4.5	0.00	6.6	0.3	-0.01	7.4	1.1

QUICK.	—	23.65	0.0	1.6	135.63	0.0	1.6	0.91	1.5	0.1	20.27	0.0	0.3
	MOVE	0.78	3.7	4.6	0.22	6.9	4.4	0.00	6.3	0.2	0.34	1.7	0.8
	CHAN	-0.01	4.9	7.1	0.01	7.2	4.5	0.00	6.5	0.2	0.01	8.1	1.0
TRI. BOTH	—	2.12	0.2	345.4	0.06	21.6	208.9	0.01	19.3	2.7	4.42	0.0	52.5
	MOVE	0.23	12.4	347.9	0.05	10.4	211.1	0.00	8.7	2.8	0.26	5.5	52.9
	CHAN	-0.40	13.4	349.8	0.03	6.8	211.8	-0.00	7.0	2.8	-0.07	10.1	53.2
D. D. SAM. 1	—	11.24	0.0	9.4	48.05	0.0	4.0	0.42	0.0	0.1	9.65	0.0	1.3
	MOVE	0.30	9.6	12.2	0.29	5.6	5.8	0.00	7.1	0.3	0.29	2.2	1.8
	CHAN	-0.40	11.8	14.0	0.00	7.2	6.1	-0.00	7.5	0.3	-0.05	6.6	2.0
D. D. SAM. 2	—	10.75	0.0	15.6	99.82	0.0	6.0	0.86	0.0	0.1	9.38	0.0	2.4
	MOVE	0.33	8.1	18.3	0.27	5.9	8.4	0.00	6.2	0.3	0.32	4.4	2.8
	CHAN	-0.37	11.6	20.1	0.00	7.2	8.7	-0.00	7.1	0.3	-0.07	11.1	3.1
MOVES	—	0.85	11.8	17.6	0.28	12.0	11.5	0.00	25.1	0.7	0.35	8.1	2.6
	MOVE	0.85	2.2	19.4	0.28	5.7	14.3	0.00	8.8	0.8	0.35	2.9	2.9
	CHAN	-0.07	3.9	21.6	0.03	6.7	14.2	-0.00	7.2	0.8	-0.08	9.6	3.1
CHAN. BOTH	—	0.77	14.0	3.0	0.21	14.3	3.6	0.00	18.5	0.2	0.31	8.1	0.5
	MOVE	0.77	3.6	4.8	0.21	6.7	5.4	0.00	6.7	0.3	0.31	3.0	0.8
	CHAN	-0.09	5.1	7.0	0.03	6.7	5.9	-0.00	7.4	0.3	-0.05	7.5	1.0

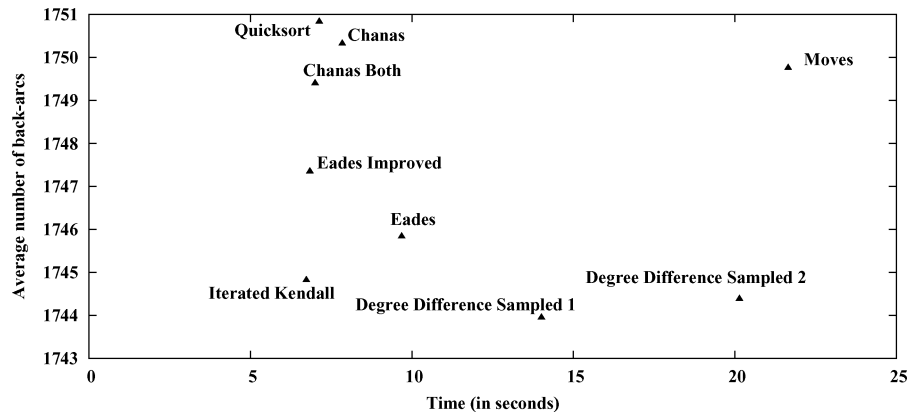


Fig. 5. The tradeoff between amount of time taken compared to the effectiveness of the various algorithms as inputs to Chanas. A point to the left indicates reduced running time; further downwards indicates fewer errors in the output ranking. The data shown is from the Biased dataset, $p = 0.6$, from a run of 1,000 instances of size 100. On the efficient frontier (the most valuable algorithms) are those that have no other algorithm both below and to the left—these include iterated Kendall, and degree difference sampled 1.

sampling methods for degree difference—degree difference sampled 1 and degree difference sampled 2—seem better compromises.

5.1 The Time-Effectiveness Tradeoff

Figure 5 highlights the tradeoff between speed and efficiency of selected algorithms. On the biased ($p = 0.6$) dataset, the two algorithms that cannot be said to be worse than others (as they are on the efficient frontier) are the hybrid of degree difference sampled 1 and Chanas and the hybrid of iterated Kendall and Chanas. Chanas by itself, not shown in this picture, unsurprisingly takes less time than these two algorithms. These three algorithms are, therefore, the subject of further study.

In Figures 6(a) and 6(b), we experimentally test the benefit of multiple runs of a randomized algorithm to see if there is an effective sacrifice of running time in comparison to solution quality.

In attempt to give each algorithm an equal amount of time to operate, we repeat each algorithm twice, four times, eight times, etc. On each run, we execute Chanas as a finishing step. In Figures 6(a) and 6(b), we plot the best result found over the set of repeats, along with the total time taken, in seconds. In Tables II and III, we display the numerical results, along with the average and worst result for each set of repeats.

There is a certain random component to all of these algorithms. For iterated Kendall, there is less randomness in the algorithm, and this is borne out in the results. This leads to a relative stagnation in its effectiveness, especially on the EachMovie data. The hybrid Chanas and degree difference sampled 1 algorithms perform similarly on both datasets, however, degree difference sampled 1 shows some advantage on the **Biased** dataset, while Chanas has a clear advantage on EachMovie. Naturally, one could run Wilcoxon-style [Wilcoxon

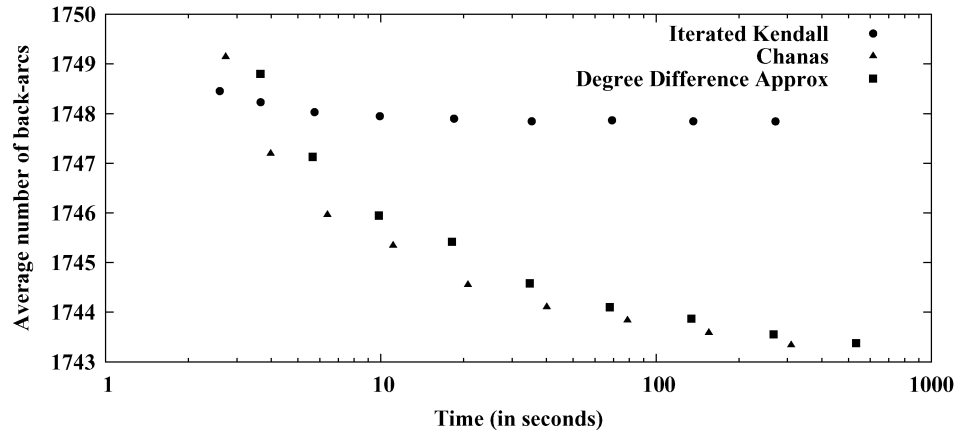
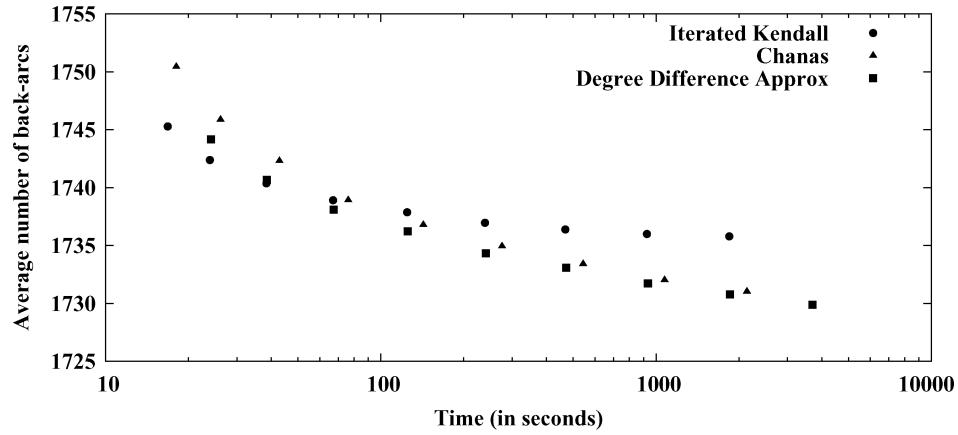


Fig. 6. The effect of repeated calls to a hybrid of each algorithm and Chanas. We ran each algorithm once, twice, four times, etc., up to 256 times. Displayed for each run is the best-performing output over all repeats.

1945] nonparametric tests to show that one algorithm is significantly better than the other in a pure statistical sense. However, the difference may not be important, and it can be hard to compare algorithms that take slightly different running times. We leave the graphs themselves as the strongest evidence of the similar performance.

5.2 Time Taken as Problem Size Increases

We investigated the running times of the algorithms on larger data sets to infer something about the running time growth (see Figure 7). In the diagram, we have separated the algorithms into seven classes, each with roughly similar running times (generally within 10% for each sample). We can see immediately that triangle both and degree difference have very high running times,

dataset

Rep.	Time	Iterated Kendall				Chanas				Degree Difference Approx			
		Avg.	Best	Worst	Time	Avg.	Best	Worst	Time	Avg.	Best	Worst	Time
1	19.2	1,745.11	1,745.11	1,745.11	20.3	1,750.22	1,750.22	1,750.22	26.3	1,744.65	1,744.65	1,744.65	1,744.65
2	26.5	1,745.02	1,742.36	1,747.68	28.8	1,750.54	1,745.35	1,755.73	40.8	1,743.92	1,740.38	1,747.45	1,747.45
4	40.6	1,744.90	1,740.22	1,749.93	45.2	1,750.57	1,741.65	1,760.51	69.4	1,744.23	1,738.08	1,750.61	1,750.61
8	69.2	1,745.08	1,738.76	1,751.75	78.5	1,750.66	1,738.87	1,764.53	127.0	1,744.24	1,735.99	1,753.47	1,753.47
16	126.5	1,745.04	1,737.73	1,753.11	144.9	1,750.53	1,736.44	1,768.42	242.0	1,744.25	1,734.30	1,755.78	1,755.78
32	240.7	1,745.08	1,736.76	1,754.33	277.4	1,750.57	1,734.68	1,772.00	471.2	1,744.25	1,732.88	1,757.80	1,757.80
64	468.2	1,745.06	1,736.09	1,755.02	542.2	1,750.57	1,732.97	1,775.53	929.8	1,744.25	1,731.68	1,760.13	1,760.13
128	924.3	1,745.05	1,735.78	1,755.53	1,071.6	1,750.57	1,731.80	1,778.49	1,848.4	1,744.24	1,730.60	1,762.00	1,762.00
256	1,837.6	1,745.04	1,735.48	1,755.94	2,133.4	1,750.55	1,730.67	1,781.28	3,689.8	1,744.24	1,729.69	1,763.69	1,763.69

Table III. Similar Data to Table II, for the EachMovie Dataset

Rep.	Time	Iterated Kendall			Chanas			Degree Difference Approx				
		Avg.	Best	Worst	Time	Avg.	Best	Worst	Time	Avg.	Best	Worst
1	3.0	1,750.58	1,750.58	1,750.58	3.1	1,750.68	1,750.68	1,750.68	3.9	1,751.39	1,751.39	1,751.39
2	4.0	1,750.60	1,750.27	1,750.93	4.3	1,751.08	1,749.53	1,752.63	6.1	1,750.77	1,749.21	1,752.34
4	6.1	1,750.60	1,750.11	1,751.12	6.7	1,750.82	1,747.91	1,754.04	10.2	1,750.78	1,748.13	1,753.91
8	10.3	1,750.54	1,750.02	1,751.21	11.7	1,751.06	1,747.26	1,755.91	18.4	1,750.78	1,747.25	1,755.26
16	18.6	1,750.56	1,749.91	1,751.32	21.3	1,751.11	1,746.62	1,758.06	35.0	1,750.78	1,746.77	1,757.19
32	35.7	1,750.58	1,749.91	1,751.38	40.5	1,751.03	1,746.28	1,759.03	68.0	1,750.74	1,746.08	1,758.13
64	69.6	1,750.57	1,749.88	1,751.40	79.0	1,751.07	1,745.84	1,761.29	134.1	1,750.73	1,745.84	1,759.62
128	137.1	1,750.59	1,749.87	1,751.41	156.7	1,751.07	1,745.59	1,762.00	266.4	1,750.74	1,745.52	1,760.30
256	272.8	1,750.57	1,749.87	1,751.41	311.9	1,751.09	1,745.28	1,763.47	531.8	1,750.77	1,745.31	1,762.06

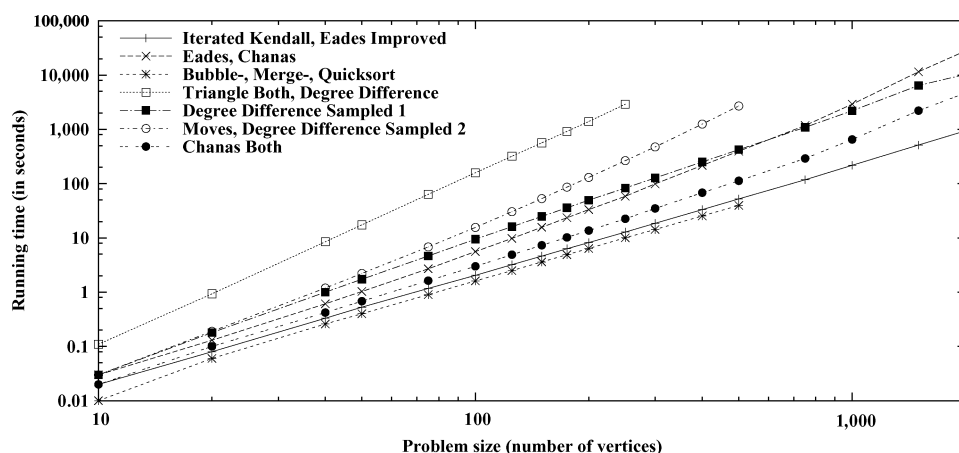


Fig. 7. The running time taken by a selection of the algorithms, as the problem size increases. Each plot represents a class of algorithms with similar running times. All experiments were run on the Biased dataset, $p = 0.6$.

corresponding to their cubic order of growth. The fastest class includes iterated Kendall and quicksort as well as Chanas both (but not Chanas) and eades improved (but not Eades), which appear to have quadratic growth. So, our improved algorithms deliver a speed increase.

The middle range of speeds has Chanas, and Eades, along with degree difference sampled 1 highlighting that degree difference sampled 1 runs in essentially similar time to Chanas, although for larger data sets degree difference sampled 1 seems to have an advantage. The final class includes degree difference sampled 2 and moves, which also appear to have cubic growth.

6. CONCLUSION

In this article, we outlined the operation of a number of algorithms for the FAS problem, extending them where possible and developing a variety of new algorithms. We demonstrated that some of the simple algorithms considered cannot be good approximators. These results complement existing results about algorithms, which are proven approximators [Ailon et al. 2005; Kenyon-Mathieu and Schudy 2007].

Additionally, we examined each algorithm from a practical perspective, testing its performance on two sets of real world data, as well as synthetic data. We found in practice Chanas is a very effective algorithm; however, using the output of a different algorithm as input to Chanas is more effective again.

The most effective algorithms to do this were iterated Kendall and degree difference sampled 1, with the first being faster, whereas the second was more effective. We gave these algorithms more time (by repeated application) and found that iterated Kendall did not gain much advantage from this approach, while degree difference sampled 1 became more effective, though only slightly better than Chanas alone.

Also, we experimentally tested the large scale performance of the algorithms, finding that degree difference sampled 1 has some small running time advantage over Chanas, while alternate algorithms, such as iterated Kendall, Chanas both, and Eades improved tend to much smaller running times.

6.1 Further Work

The Chanas both algorithm runs in significantly reduced time in comparison to Chanas. However, its performance is not as impressive. Perhaps there is a better order to search local moves, which will run faster, yet perform as well as Chanas.

The degree difference sampled 1 algorithm performs well, yet there are many other ways of sampling the vertices to check indegrees. These could be investigated—leading potentially to both better performing algorithms and theoretical results about them.

Most of the algorithms outlined here work unmodified on nontournament digraphs. However, some, for example triangle both and degree difference, will not work as currently specified, but perhaps analogous versions could be found that will. It would be profitable to test all these algorithms on nontournaments in much the same way as in this article.

ACKNOWLEDGMENTS

Many thanks to Laurence Park for providing the dataset for the WebCommunities set of experiments. Thanks to Compaq for the EachMovie dataset. Thanks to Peter Stuckey, Adrian Pearce, and Nick Wormald and the anonymous reviewers of preliminary versions, for helpful advice and comments.

REFERENCES

- AILON, N., CHARIKAR, M., AND NEWMAN, A. 2005. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC'05)*. ACM, New York, 684–693.
- ALI, I., COOK, W., AND KRESS, M. 1986. On the minimum violations ranking of a tournament. *Manage. Sci.* 32, 6, 660–672.
- ARORA, S., FRIEZE, A., AND KAPLAN, H. 2002. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Math. Program.* 92, 1, 1–36.
- ARYA, V., GARG, N., KHANDEKAR, R., MEYERSON, A., MUNAGALA, K., AND PANDIT, V. 2004. Local search heuristics for k -median and facility location problems. *SIAM J. Comput.* 33, 3, 544–562.
- ASLAM, J. AND MONTAGUE, M. 2001. Models for metasearch. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, 276–284.
- BANSAL, N., BLUM, A., AND CHAWLA, S. 2004. Correlation clustering. *Mach. Learn.* 56, 1, 89–113.
- CHANAS, S. AND KOBYLÁNSKI, P. 1996. A new heuristic algorithm solving the linear ordering problem. *Comput. Optim. Appl.* 6, 2, 191–205.
- CHARBIT, P., THOMASSÉ, S., AND YEO, A. 2006. The minimum feedback arc set problem is NP-hard for tournaments. *Comb. Probab. Comput.* 16, 01, 1–4.
- COLEMAN, T., SAUNDERSON, J., AND WIRTH, A. 2008. A local-search 2-approximation for 2-correlation-clustering. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA 08)*. Springer, Berlin, 308–319.
- COOK, W., GOLAN, I., AND KRESS, M. 1988. Heuristics for ranking players in a round robin tournament. *Comput. Operations Res.* 15, 2, 135–144.

- COPPERSMITH, D., FLEISCHER, L., AND RUDRA, A. 2006. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proceedings of the 17th Annual ACM SIAM Symposium on Discrete Algorithm (SODA'06)*. ACM, New York, 776–782.
- DWORK, C., KUMAR, R., NAOR, M., AND SIVAKUMAR, D. 2001. Rank aggregation revisited. In *Proceedings of the 10th International World Wide Web Conference (WWW'01)*. ACM, New York, 613–622.
- EADES, P., LIN, X., AND SMYTH, W. 1993. A fast and effective heuristic for the feedback arc set problem. *Inf. Process. Lett.* 47, 6, 319–323.
- EADES, P. AND WORMALD, N. 1994. Edge crossings in drawings of bipartite graphs. *Algorithmica* 11, 4, 379–403.
- JOHNSON, D. 1975. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.* 4, 1, 77–84.
- KANUNGO, T., MOUNT, D., NETANYAHU, N., PIATKO, C., SILVERMAN, R., AND WU, A. 2004. A local search approximation algorithm for k-means clustering. *Comput. Geom. Theory App.* 28, 2–3, 89–112.
- KARP, R. M. 1972. Reducibility among combinatorial problems. *Complexity Comput.*, 85–103.
- KEMENY, J. 1959. Mathematics without numbers. *Daedalus* 88, 571–591.
- KENDALL, M. 1955. Further contributions to the theory of paired comparisons. *Biometrics* 11, 1, 43–62.
- KENYON-MATHIEU, C. AND SCHUDY, W. 2007. How to rank with few errors. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC'07)*. ACM, New York.
- KLEIN, D. AND RANDIĆ, M. 1987. Innate degree of freedom of a graph. *J. Comput. Chem.* 8, 4, 516–521.
- MCJONES, P. 1997. Eachmovie collaborative filtering data set. DEC Systems Research Center, 249.
- PACHTER, L. AND KIM, P. 1998. Forcing matchings on square grids. *Discrete Math.* 190, 1-3, 287–294.
- PARK, L. AND RAMAMOHANARAO, K. 2007. Mining web multi-resolution community-based popularity for information retrieval. In *Proceedings of the 16th Annual ACM Conference on Information and Knowledge Management (CIKM'07)*. ACM, New York, 545–52.
- SAAB, Y. 2001. A fast and effective algorithm for the feedback arc set problem. *J. Heuristics* 7, 3, 235–250.
- WILCOXON, F. 1945. Comparisons by ranking methods. *Biometric Bull.* 1, 80–82.

Received March 2009; accepted March 2009