Ranking Tournaments: Local Search and a New Algorithm*

Tom Coleman† Anthony Wirth‡

Abstract

Ranking data is a fundamental organizational activity. Given advice, we may wish to rank a set of items to satisfy as much of that advice as possible. In the Feedback Arc Set (FAS) problem, advice takes the form of pairwise ordering statements, 'a should be ranked before b'. Instances in which there is advice about every pair of items is known as a *tournament*. This task is equivalent to ordering the nodes of a given directed graph to minimize the number of arcs pointing in one direction.

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 paper we re-examine many of these existing algorithms and develop some new techniques for solving FAS. The algorithms are tested on both synthetic and Rank Aggregation-based 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 beats existing techniques on a variety of data sets.

1 Introduction

1.1 The Feedback Arc Set Problem The Feedback Arc Set (FAS) problem is a key ranking problem, asking us to rank items in a set given only advice about the best way to order specific pairs. A ranking π 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'. These statements may be contradictory: the aim of the FAS problem is to produce a π that violates as few of these pairwise statements as possible.

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.

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 'Feedback Arc Set'. An equivalent formulation of the problem is therefore: given a digraph G, find the smallest subset of the arcs of G that intersects all cycles in G (whose removal therefore renders G acyclic). In this paper we focus on the special case of *tournament* graphs, in which there is an arc between every pair of nodes. We also consider *weighted* tournaments, in which the weights on arcs $u \to v$ and $v \to u$ must sum to 1.

Originally motivated by problems in circuit design [13], FAS has found applications in many areas, including computational chemistry [21, 19], and graph drawing [12]. An application to Rank Aggregation in particular has seen much focus solving the problem on directed complete graphs (tournaments).

1.2 The Rank Aggregation Problem Closely related to the FAS problem is the Rank Aggregation problem. Given a set of rankings, we must find a single ranking that *best* represents the consensus. Dwork et al. [10] 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 [5].

Precisely what *best* means is a difficult issue, and it relates to how we interpret the similarity of rankings. The Kemeny distance [16] $K(\pi, \sigma)$ between two rankings, π and σ, is defined as the number of pairs i, j for which π ranks i above j, yet σ ranks j above i. In this paper, the Rank Aggregation problem seeks a ranking τ that minimizes the *sum* of the Kemeny distances from τ to each of the input rankings.

This problem is a special case of FAS on weighted tournaments. There is a fairly simple reduction from Rank Aggregation to FAS: for each i, j, if i is ranked above j in some fraction w of the input rankings, place an arc between i and j with weight w.

1.3 The Linear Ordering Problem A very similar problem to the Feedback Arc Set problem is the Linear Ordering problem. Here we have a matrix (c_{ij}), and we want to choose an ordering σ such that $\sum_{i <_\sigma j} c_{ij}$ is minimized. The weighted tournament version of FAS is a special case of this, under the

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

†The University of Melbourne

‡The University of Melbourne

constraint that $c_{ij} = 1 - c_{ji}$.

1.4 Hardness of these problems The FAS problem is one of Karp's [15] original NP-complete problems (in the general case). Dwork et al. [10] prove the RANK AGGREGATION problem is NP-hard, even with as few as 4 input rankings. Recently, the FAS problem has been shown to be NP-hard even on unweighted tournaments [7]. 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.

1.5 Algorithms with Guarantees The first constant factor approximation algorithm for FAS, designed primarily for tournaments, was the pivoting algorithm of Ailon et al. [1]. Intuitively similar to quicksort, it uses on average $O(n \log n)$ comparisons for an expected 3-approximation on *tournament graphs*.

Coppersmith et al. [9] show that ordering the nodes by their indegrees is a 5-approximator on tournament instances, regardless of how ties are broken. The vast majority of sporting leagues use this heuristic to rank teams (though often with elaborate tie-breaking mechanisms).

Finally, Kenyon-Mathieu and Schudy completed the approximation picture for tournaments picture with a PTAS (polynomial-time approximation scheme) [18]. This scheme comprises a simple moves-based local search heuristic—which we investigate in isolation below—and the PTAS of Arora, Frieze and Kaplan [3] for the MAXIMUM ACYCLIC SUBGRAPH problem.[1]

1.6 Heuristics Viewing the input as a directed graph, we define the *Kendall score* of a node to be its indegree. One of the simplest and earliest heuristics developed for the FAS problem ranks the items according to their *Kendall scores*, breaking ties arbitrarily [17]. Ali et al. [2] and Cook et al. [8] improve this technique by considering various methods of breaking ties. Let us define the ITERATED KENDALL algorithm to be:

> Rank the nodes by their Kendall scores. 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.

Eades et al. [11] created an algorithm that is in fact quite similar to ITERATED KENDALL, possibly inspired by selection sort. We define the EADES algorithm to be:

> 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.

In retrospect, the tie breaking could be done in a more sophisticated way.

Chanas and Kobylanski [6] present an algorithm for the LINEAR ORDERING problem 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 [23] 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.

1.7 Our Contributions / Paper Organization In Section 2 we provide details of the algorithms we will test. Some are existing procedures (especially local search approaches), others are our improvements to them, still others are based on our scheme of reversing arcs (in an organized 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 three sets of FAS problems generated from RANK AGGREGATION data. 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

2.1 Improving the Eades Algorithm The EADES algorithm focuses on the left side of the ranking. We improve this by allowing the selection of a node to either end of the ranking. Let $\text{In}(v)$ stand for the indegree of node v, with $\text{Out}(v)$ its outdegree. We define the algorithm EADES IMPROVED to be:

> Select the node u that maximizes $|\text{In}(u) - \text{Out}(u)|$ and place it at the appropriate end of the ranking. Recurse on the subgraph induced by removing u. Again, break ties arbitrarily.

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

2.2 Sorting Algorithms As suggested above, Ailon et al.'s algorithm [1] is much like quicksort. For this paper QUICKSORT is defined to be:

> Choosing pivots uniformly at random, run the quicksort algorithm with the *comparison* function: $u < v$ if and only if $u \to v$.

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*.

Cook et al. [8] focus on ensuring a Hamiltonian path exists in along the final ordering of the nodes; any sensible algorithm should achieve this. The method they use to reach this is in effect a BUBBLESORT of the tournament, using the same comparison function as QUICKSORT.

To our knowledge, no paper has studied the MERGESORT approach. In this paper we test its performance.

As noted above, the EADES algorithm is the obvious analogy to selection sort (with our improved version being a two-sided selection sort). Chanas and Kobylanski [6] apply an insertion technique to the LINEAR ORDERING problem that is more involved than the usual insertion sort. Their 'SORT' procedure is:

> Make a single pass through the nodes from the left to the right. As each node is considered, it is moved to the left, inserted into the position that minimizes the number of back-arcs.

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 (SORT* $\circ$ REVERSE)*, which Chanas and Kobylanski show outperforms the original version in practice.

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 the k-median [4] and k-means [14] problems, along with related problems. 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 paper we show that neither algorithm can provide an approximation guarantee. We found that the MOVES heuristic performs well in practice, but that SWAPS does not: so the latter was omitted from our experiments.

One particular advantage of a local search techniques 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.

Chanas 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 in the order is possible. This is essentially a scheme for selecting which MOVES-style changes to make. The operation SORT $\circ$ 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, termed CHANAS BOTH: the SORT procedure 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 only has a cycle if it has a directed triangle ($\vec{\triangle}$). We therefore considered 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 $\vec{\triangle}$s. Our algorithms work in the following way:

> While the tournament is not acyclic, *choose* an arc and reverse its orientation. Once the tournament is acyclic, use the ordering of the nodes 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 $\vec{\triangle}$s an arc is involved in its *triangle count*. The first algorithm, TRIANGLE COUNT, simply chooses the arc with the highest triangle count. There is a pitfall here though: reversing an arc can create a new $\vec{\triangle}$ that did not previously exist.

To avoid this problem we instead can choose the arc for which the number of $\vec{\triangle}$s after the reversal is least.

We call this heuristic TRIANGLE DELTA. A potential problem with TRIANGLE DELTA could be the existence of a tournament that was not acyclic (and thus still had triangles), yet contained no arcs whose reversal would lower the number of $\vec{\triangle}$s. Lemma 2.1 proves that this situation is impossible.

LEMMA 2.1. *Let T be a tournament. Then if T has a cycle, there exists an arc $e \in T$ such that reversing e will reduce the triangle count of T.*

Proof. See Appendix A.

We also considered a third approach, called TRIANGLE BOTH: choose the arc with the highest triangle count, provided that it reduces the number of $\vec{\triangle}$s. Note that calculating the triangle count, and the change in $\vec{\triangle}$s, for every arc of the digraph requires $O(n^3)$ operations.

In a weighted tournament, the weight of a $\vec{\triangle}$ 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 $\vec{\triangle}$s it is involved in.

2.5 Degree Difference Algorithms We designed a new algorithm DEGREE DIFFERENCE, that selects an arc to reverse based on a criterion that may be related to the triangle count. We select the arc $u \to v$ 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 (which we use as our solution), as the value of $\sum_v \mathrm{In}(v)^2$ increases whenever an arc from a higher-degree to a lower-degree node is reversed.

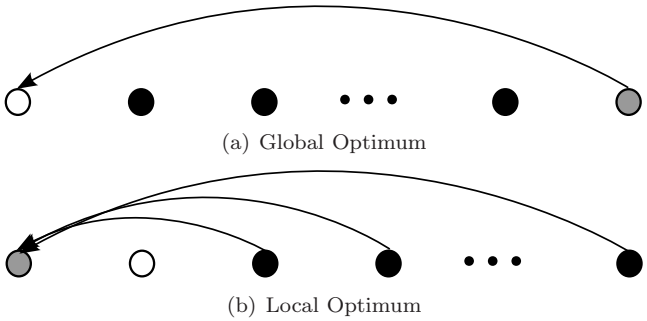
In an effort to speed up the DEGREE DIFFERENCE algorithm, we used a sampling technique. We sample $\log n$ vertices (favoring high indegree) to potentially be the 'tail' of the arc, and another $\log n$ (favoring low indegree) to potentially be the 'head'. We then check each of the $\log^2 n$ arcs between sampled vertices, choosing the back-arc of highest degree difference. We resample if no back-arc of non-negative degree difference is found. This algorithm is called DEGREE DIFFERENCE SAMPLED 1, and it takes approximately $O(n^2 \log^2 n)$ time on average.

A further variation, DEGREE DIFFERENCE SAMPLED 2, maintains two lists: one of potential head nodes, and one of potential tail nodes. A node v is a potential head if its indegree, $\mathrm{In}(v)$, is not unique or there is no node of indegree $\mathrm{In}(v)-1$. Similarly, a node u is a potential tail if $\mathrm{In}(u)$ is not unique or there is no node of indegree $\mathrm{In}(u)+1$. The motivation for this is to increase $\sum_v \mathrm{In}(v)^2$. 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.

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



(a) Global Optimum

(b) Local Optimum

Figure 1: Standard Bad Example

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 1(a) shows this (global) optimum configuration; a local optimum for the SWAPS heuristic, shown in Figure 3.1, has cost $n-2$.

Note also that there is no guarantee that BUBBLESORT will start with node n placed after node 1. Without this, it will also reach the costly local optimum.

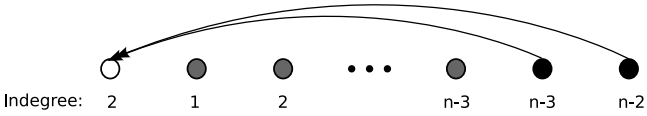
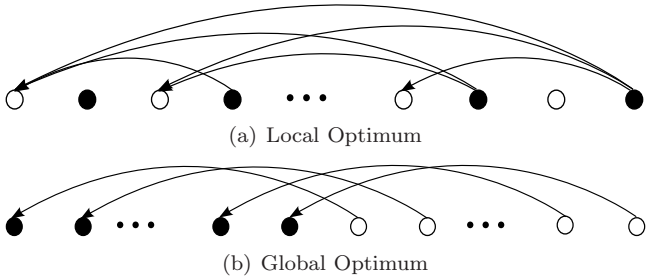


Figure 2: The Counterexample for the EADES algorithm.

3.2 The Eades algorithms Figure 2 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 IMPROVED 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 $(2, 3, 4, 5, \ldots, n-1, n, 1)$, which has a cost of $n-3$.



(a) Local Optimum

(b) Global Optimum

Figure 3: Counterexample family (n even) for the MOVES local search heuristic (and thus for CHANAS & CHANAS BOTH). (a) There is a back-arc from each even-numbered node (black) to each odd-numbered node (white) that is at least 3 positions preceding. (b) Shows the global optimum. The black nodes have the same relative ordering to one another, as do the white nodes.

3.3 Moves and Chanas The configuration on the left of Figure 3 is a local optimum for the MOVES heuristic. Following the discussion at the end of Section 2.3, it is also a configuration that CHANAS and CHANAS BOTH can be stuck in.

The spacing of the back-arcs ensures that it is never an improvement to move a single node. The cost of the local optimum is $n^2/8 - n/4$. The global minimum, shown in Figure 3(b), 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 4-core Intel Xeon 3.2GHz machine, with 8 gigabytes 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 CHANAS as input to CHANAS is an interesting case—although CHANAS is a local search algorithm (which would imply that a second run would have no further effect), the SORT* $\circ$ REVERSE step can significantly change the ordering without affecting the solution quality (in terms of number of back-arcs). So repeated calls to CHANAS can sometimes get the algorithm out of a local plateau. However, it is difficult to tell when this is going to happen.

4.1 Datasets We tested the FAS algorithms on the following synthetic dataset.

Biased 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 random tournament.

The following datasets are based around the idea of aggregating inconsistent rankings of a set of datapoints into a FAS-style tournament.

WebCommunities Our colleague, Laurence Park, provided us with a set of 9 different complete rankings of a large set of documents (25 million) [22]. 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 [20] 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 data sets. We selected just four of the data sets to display in Table 1: these show a variety of performance characteristics. We first note the striking performance of the CHANAS local search procedure. 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, but only 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

Table 1: Each algorithm is tested on the Biased data set with $p = 0.6$ and 0.95. We also tested them on the WebCommunities and EachMovie data sets, as described above. 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**.

		0.6			**0.95**			**laurence**			**eachmovie**		
	Variant	**Errors**	**Wins**	**Time**	**Errors**	**Wins**	**Time**	**Errors**	**Wins**	**Time**	**Errors**	**Wins**	**Time**
	—	12.02	0.0	2.1	29.84	0.0	1.9	15.70	0.0	0.1	28.96	0.0	0.3
IT. KEND.	MOVE	0.37	6.7	4.9	0.30	5.2	3.8	0.00	3.3	0.4	0.40	9.8	0.6
	CHAN	-0.31	9.7	6.7	0.00	7.3	4.5	0.00	3.7	0.3	-0.01	9.9	0.8
	—	9.89	0.0	4.7	62.17	0.0	4.7	31.49	0.0	0.2	34.78	0.0	0.7
EADES	MOVE	0.58	3.7	7.5	0.31	5.3	6.9	0.01	2.8	0.6	0.35	6.8	1.0
	CHAN	-0.25	8.3	9.7	0.00	7.3	7.3	-0.00	3.6	0.4	0.00	5.7	1.2
	—	8.65	0.0	1.9	52.23	0.0	1.9	19.45	0.0	0.1	37.65	0.0	0.3
EADES IMP.	MOVE	0.61	2.8	4.7	0.32	5.0	4.7	0.00	4.5	0.3	0.47	2.5	0.6
	CHAN	-0.17	6.6	6.8	0.00	7.3	4.7	0.00	3.8	0.3	0.04	10.0	0.8
	—	0.00	74.0	5.6	0.00	27.8	2.8	0.00	11.8	0.1	0.00	75.7	0.6
CHANAS	MOVE	0.00	32.8	7.5	0.00	12.1	5.3	0.00	5.0	0.3	0.00	35.3	0.8
	CHAN	0.00	6.3	7.8	0.00	7.3	5.5	0.00	3.7	0.3	0.00	6.0	0.9
	—	35.28	0.0	1.6	731.12	0.0	1.6	75.39	0.0	0.1	210.08	0.0	0.2
BUBBLE.	MOVE	0.76	3.9	4.6	0.21	6.7	3.4	0.00	2.9	0.3	0.25	11.8	0.6
	CHAN	0.02	4.2	7.2	0.00	7.2	3.6	-0.00	4.7	0.3	0.01	6.9	0.8
	—	23.25	0.0	1.6	130.91	0.0	1.6	0.87	0.7	0.1	65.73	0.0	0.2
MERGE.	MOVE	0.80	2.4	4.6	0.23	6.1	5.2	0.00	3.2	0.2	0.39	10.6	0.6
	CHAN	0.02	5.5	7.2	0.00	7.2	4.5	0.00	3.5	0.3	-0.01	14.4	0.9
	—	23.66	0.0	1.6	135.85	0.0	1.6	0.84	0.8	0.1	61.68	0.0	0.2
QUICK.	MOVE	0.79	3.4	4.6	0.22	6.6	4.4	0.00	3.3	0.2	0.39	7.2	0.6
	CHAN	0.03	4.8	7.1	0.01	7.2	4.5	0.00	3.5	0.2	-0.01	12.2	0.8
	—	2.12	0.2	345.4	0.06	21.5	208.9	0.01	9.6	2.7	1.13	7.2	31.2
TRI. BOTH	MOVE	0.24	11.6	347.9	0.05	9.9	211.1	0.00	4.7	2.8	0.05	17.9	31.5
	CHAN	-0.36	12.7	349.8	0.03	6.8	211.8	0.00	3.8	2.8	-0.16	19.5	31.7
	—	7.47	0.0	158.0	0.03	24.2	27.8	0.02	5.3	0.4	2.27	2.7	8.5
D. D.	MOVE	0.58	4.8	160.7	0.03	11.1	29.5	0.00	5.1	0.5	0.17	11.3	8.8
	CHAN	-0.17	5.0	162.8	0.00	7.3	29.8	-0.00	4.0	0.5	-0.10	17.6	9.0
	—	11.25	0.0	9.4	48.17	0.0	4.0	0.41	0.0	0.1	23.33	0.0	0.8
D. D. SAM. 1	MOVE	0.31	9.2	12.2	0.29	5.4	5.8	0.00	3.7	0.3	0.38	9.9	1.1
	CHAN	-0.36	11.1	14.0	0.00	7.2	6.1	-0.00	4.0	0.3	0.01	9.2	1.3
	—	10.75	0.0	15.6	100.18	0.0	6.0	0.84	0.0	0.1	27.05	0.0	1.2
D. D. SAM. 2	MOVE	0.35	7.6	18.3	0.28	5.7	8.4	0.00	3.3	0.3	0.36	3.9	1.6
	CHAN	-0.34	10.4	20.1	0.00	7.2	8.7	-0.00	3.9	0.3	-0.07	12.7	1.8
	—	0.85	11.8	17.6	0.28	12.0	11.5	0.00	12.5	0.7	0.45	29.4	1.9
MOVES	MOVE	0.87	1.9	19.4	0.28	5.5	14.3	0.00	4.7	0.8	0.47	8.4	2.2
	CHAN	-0.03	3.8	21.6	0.03	6.7	14.2	-0.00	3.9	0.8	0.08	10.1	2.3
	—	0.78	14.0	3.0	0.22	14.3	3.6	0.00	9.3	0.2	0.37	31.0	0.4
CHAN. BOTH	MOVE	0.79	3.3	4.8	0.22	6.5	5.4	0.00	3.5	0.3	0.38	10.6	0.7
	CHAN	-0.05	4.9	7.0	0.03	6.7	5.9	-0.00	4.0	0.3	0.04	11.7	0.8

with local search. The Degree Difference algorithm is similar, though at a different point on the performance/speed tradeoff. The sampling methods for Degree Difference, Degree Difference Sampled 1 and Degree Difference Sampled 2, seem a better compromise.

5.1 The Time-Effectiveness Tradeoff Figure 4 highlights the relative performance and efficiency of selected algorithms. On the Biased ($p = 0.6$) data set, the two algorithms that cannot be said to be worse than others 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.

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. But for Chanas, the input is in fact a random ordering of the items. In Figures 5(a) and 5(b), we repeat each algorithm twice, four times, eight times, etc. to see whether spending greater computation time produces better solutions. Unfortunately for Iterated Kendall, there seems to be a relative stagnation in its effectiveness, especially on the **EachMovie** data. It is hard to differentiate between the hybrid Chanas and Degree Difference Sampled 1 algorithms. Naturally, one could run Wilcoxon-style non-parametric 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 similarity.

6 Conclusion

In this paper we outlined the operation of a number of algorithms which aim to solve the FAS problem, extending them where possible and developing a variety of new algorithms. We analyzed some of algorithms from the perspective of algorithmic approximability, proving that they cannot be good approximators. These results complement existing results about algorithms which are proven approximators.

Additionally we examined each algorithm from a practical perspective, testing their 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 with 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 Degree Difference Sampled 1 became more effective, though only slightly better than Chanas.

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 investigate local moves than both algorithms that will run quickly, 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 non-tournament 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 non-tournaments in much the same way as in this paper.

An issue that has not been addressed in great detail in this paper is how the algorithms scale for larger data sets. Further work investigating the large-scale time-performance of these algorithms would be of value.

7 Acknowledgements

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 for helpful advice and comments.

References

[1] Ailon, N., Charikar, M., and Newman, A. Aggregating inconsistent information: ranking and clustering. *Proceedings of the thirty-seventh annual ACM symposium on Theory of Computing* (2005), 684–693.

[2] Ali, I., Cook, W., and Kress, M. On the Minimum Violations Ranking of a Tournament. *Management Science 32*, 6 (1986), 660–672.

[3] Arora, S., Frieze, A., and Kaplan, H. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming 92*, 1 (2002), 1–36.

[4] Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., and Pandit, V. Local Search Heuristics for k-Median and Facility Location Problems. *SIAM Journal on Computing 33*, 3, 544–562.

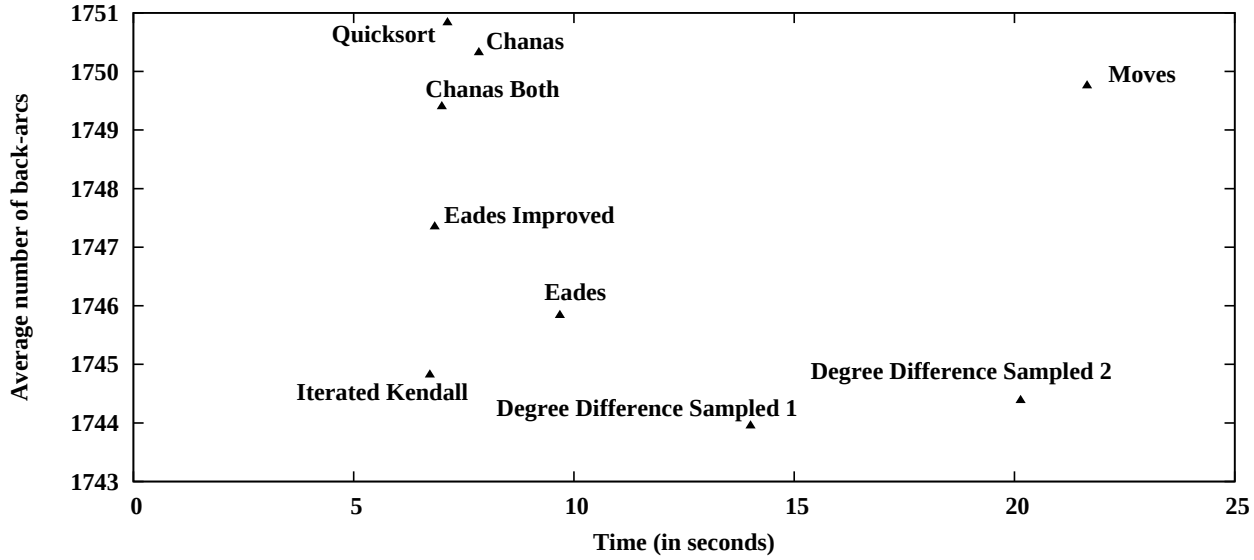
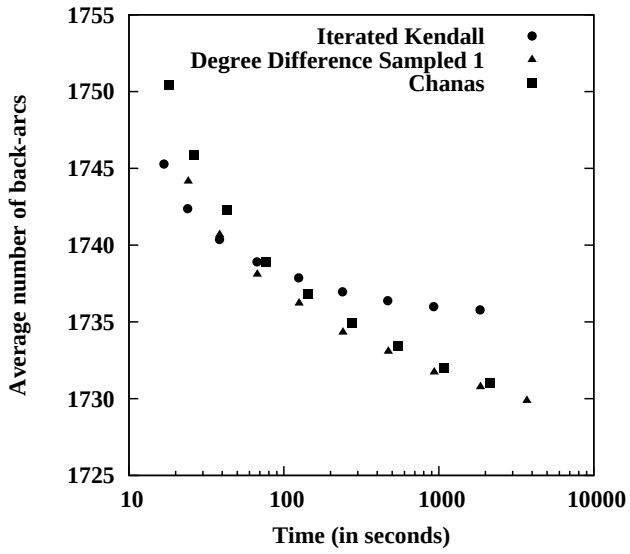
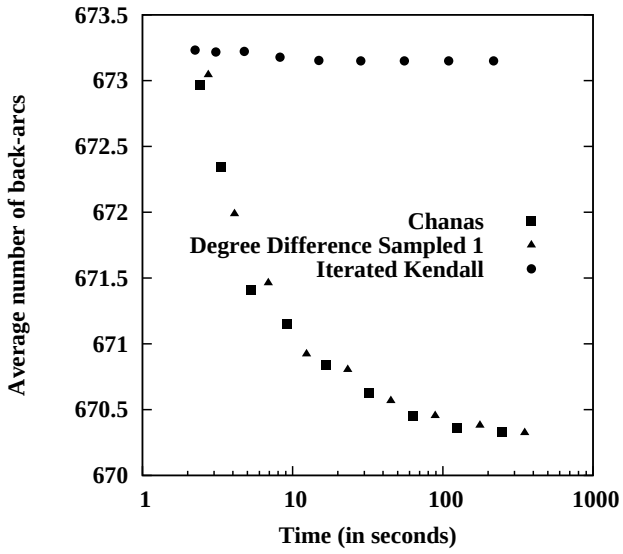


Figure 4: The amount of time taken compared to the effectiveness of the various algorithms as inputs to CHANAS. A point to the left indicates reduced running time; further downwards indicates less errors in the output ranking. The data shown is from the **Biased** dataset, $p = 0.6$, from a run of 1000 instances of size 100.



(a) For the **Biased** dataset, $p = 0.6$.



(b) For the **EachMovie** dataset (146 tournaments of size up to 100)

Figure 5: 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 and took the best output ordering. The effectiveness and time taken (in seconds) are displayed.

[5] Aslam, J., and Montague, M. Models for metasearch. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (2001), 276–284.

[6] Chanas, S., and Kobylański, P. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications 6*, 2 (1996), 191–205.

[7] Charbit, P., Thomassé, S., and Yeo, A. The Minimum Feedback Arc Set Problem is NP-Hard for Tournaments. *Combinatorics, Probability and Computing 16*, 01 (2006), 1–4.

[8] Cook, W., Golan, I., and Kress, M. Heuristics for ranking players in a round robin tournament. *Computers and Operations Research 15*, 2 (1988), 135–144.

[9] Coppersmith, D., Fleischer, L., and Rudra, A. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm* (2006), 776–782.

[10] Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. Rank aggregation revisited. *Proceedings of WWW10* (2001), 613–622.

[11] Eades, P., Lin, X., and Smyth, W. A Fast and Effective Heuristic for the Feedback Arc Set Problem. *Information Processing Letters 47*, 6 (1993), 319–323.

[12] Eades, P., and Wormald, N. Edge crossings in drawings of bipartite graphs. *Algorithmica 11*, 4 (1994), 379–403.

[13] Johnson, D. Finding All the Elementary Circuits of a Directed Graph. *SIAM J. Comput. 4*, 1 (1975), 77–84.

[14] Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., and Wu, A. A local search approximation algorithm for k-means clustering. *Computational Geometry: Theory and Applications 28*, 2-3 (2004), 89–112.

[15] Karp, R. M. Reducibility among combinatorial problems. *Complexity of Computer Computations* (1972), 85–103.

[16] Kemeny, J. Mathematics without numbers. *Daedalus 88* (1959), 571–591.

[17] Kendall, M. Further Contributions to the Theory of Paired Comparisons. *Biometrics 11*, 1 (1955), 43–62.

[18] Kenyon-Mathieu, C., and Schudy, W. How to rank with few errors. *Proceedings of the thirty-ninth annual ACM symposium on Theory of Computing* (2007).

[19] Klein, D., and Randić, M. Innate degree of freedom of a graph. *Journal of Computational Chemistry 8*, 4 (1987), 516–521.

[20] McJones, P. Eachmovie collaborative filtering data set. *DEC Systems Research Center 249* (1997).

[21] Pachter, L., and Kim, P. Forcing matchings on square grids. *Discrete Mathematics 190*, 1-3 (1998), 287–294.

[22] Park, L., and Ramamohanarao, K. Mining web multi-resolution community-based popularity for information retrieval. In *Proceedings of the 2007 ACM 16th Conference on Information and Knowledge Management* (2007), pp. 545–52.

[23] Saab, Y. A Fast and Effective Algorithm for the Feedback Arc Set Problem. *Journal of Heuristics 7*, 3 (2001), 235–250.

A Proof of Lemma 2.1

Let σ be an ordering on the vertices of T that induces a minimum Feedback Arc Set. 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 $\vec{\triangle}$s of the form v-w-x, where x is to the right of both v and w, and 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 $\vec{\triangle}$ x-v-w can be created where x is to the left of both vertices. So any $\vec{\triangle}$ created must involve an x between v and w.

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

1. $v \leftarrow x \leftarrow w$ (reversing a will create a triangle). Say there are A such x's.
2. $v \rightarrow x \rightarrow w$ (reversing a will delete a triangle). B of these.
3. $v \rightarrow x \leftarrow w$ (reversing a will have no effect). C of these.
4. $v \leftarrow x \rightarrow w$ (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 quickly gives

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

and therefore the number of $\vec{\triangle}$s will decrease.