Fast and Parallelizable Ranking with Outliers from Pairwise Comparisons

Sungjin Im ✉[1] and Mahshid Montazer Qaem[1]

EECS, University of California, 5200 N Lake Rd, Merced, CA 95343, USA
{sim3,mmontazerqaem}@ucmerced.edu

Abstract. In this paper, we initiate the study of the problem of ordering objects from their pairwise comparison results when allowed to discard up to a certain number of objects as outliers. More specifically, we seek to find an ordering under the popular Kendall tau distance measure, i.e., minimizing the number of pairwise comparison results that are inconsistent with the ordering, with some outliers removed. The presence of outliers challenges the assumption that a global consistent ordering exists and obscures the measure. This problem does not admit a polynomial time algorithm unless NP $\subseteq$ BPP, and therefore, we develop approximation algorithms with provable guarantees for all inputs. Our algorithms have running time and memory usage that are almost linear in the input size. Further, they are readily adaptable to run on massively parallel platforms such as MapReduce or Spark.

Keywords: Rank aggregation · outliers · approximation · distributed algorithms.

1 Introduction

Ranking is a fundamental problem arising in various contexts, including web pages ranking, machine learning, data analytics, and social choice. It is of particular importance to order n given objects by aggregating pairwise comparison information which could be inconsistent. For example, if we are given $A \prec B$ (meaning that B is superior to A) and $B \prec C$, it would be natural to deduce a complete ordering, $A \prec B \prec C$. However, there exists no complete ordering creating no inconsistencies, if another pairwise comparison result $C \prec A$ is considered together. As a complete ordering/ranking is sought from partial orderings, this type of problems is called rank aggregation and has been studied extensively, largely in two settings: (i) to find a true ordering (as accurately as possible) that is assumed to exist when some inconsistencies are generated according to a certain distribution; and (ii) to find a ranking that is the closest to an arbitrarily given set of the partial orderings for a certain objective, with no stochastic assumptions.

This paper revisits a central rank aggregation problem in the second domain, with a new angle.

The Minimum Feedback Arc Set Tournament Problem (FAST). The input is a tournament.[1] The goal is to delete the minimum number of edges in order to make the resulting graph acyclic. This is equivalent to finding a linear ordering of the vertices to incur the fewest 'backward' edges.

The FAST problem is well-studied: it does not admit a polynomial time algorithm unless NP $\subseteq$ BPP [3] and there are several constant approximations known for the problem [3,18,11]. We note that this is a special case of the more general problem (*Minimum Feedback Arc Set;* FAS *for short*) where the input graph is not necessarily complete.

1.1 Necessity of Another Measure for Inconsistencies

The FAST problem measures the quality of an ordering by the number of pairs that are inconsistent with the ordering; this measure was proposed by Kemeny [17] and is also known as the Kendall tau distance. Unfortunately, this measure fails to capture the 'locality' of inconsistencies, namely whether or not the inconsistencies are concentrated around a small number of objects. To see this, consider the two instances in Figures 1.

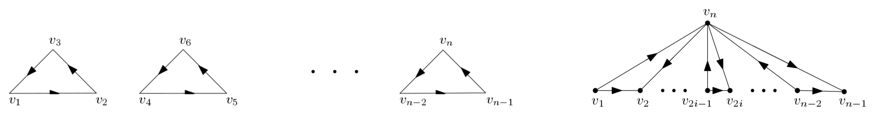

Fig. 1: The left graph consists of $\frac{n}{3}$ disjoint triangles (the rest of the arcs (v_i, v_j) for all $i < j$ are omitted). In the right graph, $v_1, v_2, \cdots, v_{n-1}$ has an ordering with no inconsistencies among them and v_n is a successor of other odd-indexed vertices while being a predecessor of even-indexed vertices.

It is easy to see no matter how we order the vertices in the left instance, we end up with having at least $n/3$ pairs that are inconsistent with the ordering, one from each triangle. Likewise, the number of inconsistent pairs in the right instance is at least $(n-1)/2$ in any ordering, one from each triangle $\{v_n, v_{2i-1}, v_{2i}\}$. Since the ordering, $v_1, v_2, \cdots, v_n$, creates $O(n)$ inconsistent pairs in both examples, the optimal objective is $\Theta(n)$ for both. However, the two instances have inconsistencies of very different natures. In the first graph, the inconsistent pairs are scattered all over the graph. In contrast, in the second graph, the inconsistent pairs are concentrated on the single vertex v_n – the second input graph becomes acyclic when v_n is removed.

The above two examples raise a question if the Kendall tau distance alone is an effective measure in the presence of outliers. We attempt to order or rank objects because we believe that they are comparable to one another, and therefore,

[1] A directed graph $G = (V, E)$ is called a tournament if it is complete and directed. In other words, for any pair $u \neq v \in V$, either $(u, v) \in E$ or $(v, u) \in E$.

there exists an ordering well summarizing the whole set of pairwise comparisons. However, this belief is no longer justified if there are some outliers that do not belong to the 'same' category of comparable objects. In the second input graph, perhaps, we should think of v_n as an outlier, as the input graph has no inconsistencies without it. Counting the number of inconsistent pairs could fail to capture the quality of the output ordering without removing outliers. This is the case particularly because we can only hope for approximate solutions and noises incurred by outliers could render multiplicative approximation guarantees not very interesting.

Motivated by this, we propose a new measure that factors in outlier vertices as well as inconsistent pairs/edges:

The Minimum Feedback Arc Set Tournament with Outliers Problem (FASTO). This is a generalization of FAST. As in FAST, we are given as input a tournament $G = (V, E)$, along with a pair of target numbers, (x^*, y^*). A pair $(V' \subseteq V, E' \subseteq E)$ is a feasible solution if $(V \setminus V', E \setminus E')$ is a DAG – we refer to V' as the outlier set and E' as the backward edge set. Here, E' is a subset of edges between $V \setminus V'$. The solution quality is measured as $(\frac{|V'|}{x^*}, \frac{|E'|}{y^*})$, which are the number of outliers and backward edges incurred, respectively, relative to the target numbers, x^* and y^*. We can assume w.l.o.g. that $x^* > 0$ since otherwise FASTO becomes exactly FAST.

We will say that a solution is (α, β)-approximate if $|V'| \leq \alpha x^*$ and $|E'| \leq \beta y^*$. An algorithm is said to be a (α, β)-approximation if it always produces a (α, β)-approximate solution for all inputs. Here, it is implicitly assumed that there is a feasible solution (V', E') w.r.t. (x^*, y^*), i.e., $|V'| \leq x^*$ and $|E'| \leq y^*$ – if not, the algorithm is allowed to produce any outputs. Equivalently, the problem can be viewed as follows: Given that we can remove up to αx^* vertices as outliers, how many edges do we need to flip their directions so as to make the input graph acyclic. But we assume that we are given a target pair (x^*, y^*), as it makes our presentation cleaner.

1.2 Our Results and Contributions

Throughout this paper, we use n to denote the number of vertices in the input graph and $N = \Theta(n^2)$, which is the asymptotic input size. We use $\tilde{O}$ or $\tilde{\Theta}$ to suppress logarithmic factors. Recall that x^* is the target number of outliers. While we propose several algorithms that are scalable and parallelizable, the following is our main theoretical result with performance guarantees for all inputs; the first case is more interesting but we also study the second case for completeness.

Theorem 1. *There is an approximation algorithm for* FASTO *with* $\tilde{O}(N)$ *running time and* $\tilde{O}(N)$ *memory usage that outputs a solution, with probability at least* $1/2 - 1/n$, *that is*

- $(O(1), O(1))$-*approximate when* $x^* \leq \sqrt{n}$ *(Section 2); and*
- $(O(\log n), O(1))$-*approximate when* $x^* > \sqrt{n}$ *(Omitted from this paper due to space limit).*

Further, this algorithm can be adapted to massively parallel computing platforms so that they run in $O(\log n)$ rounds.

We note that the running time of our algorithm, which is almost linear in the input size, is essentially *optimal*. To see this, consider a simple instance that admits an ordering with one backward edge, with $(x^*, y^*) = (1, 0)$. Then, it is unavoidable to actually find the backward edge, which essentially requires to read all edges. While we do not know how to obtain a constant approximation for the case when $x^* > \sqrt{n}$ in the massively parallel computing setting, we can still get a relatively fast algorithm in the single machine setting. More precisely, we can obtain an $(O(1), O(1))$-approximate solution for all instances, with probability at least $1/2 - 1/n$, using $\tilde{O}(\sqrt{N} x^{*2})$ running time and $\tilde{O}(\sqrt{N} x^{*2})$ memory. For the formal model of massively parallel computing platforms, see [6].

Below, we outline our contributions.

Proposing a New Metric for Ranking: Ranking with Some Outliers Removed. Outliers have been extensively studied in various fields, such as statistics, machine learning, and databases. Outliers were considered before together with ranking, but they were mostly focused on the evaluation of outliers themselves, e.g., ranking outliers [22]. Our work is distinguished, as we seek to find a clean ordering which otherwise could be obscured by outliers. Various combinatorial optimization problems have been studied in a spirit similar to ours, particularly for clustering problems [9,10,21,15]. For example, the k-means clustering problem can be extended to minimize the sum of squares of distances to the k centers from all points, except a certain number of outliers [16,15]. We feel that it is a contribution of conceptual importance to study ranking problems in this direction for the first time. We believe this new direction is worth further investigation; see Section 4 for future research directions.

Fast and Memory-efficient Algorithms with Provable Guarantees. Our work is inspired by Aboud's work [1] on a closely related clustering problem – in the Correlational Clustering problem, there is an undirected complete graph where each edge is labeled as either '$-$' or '$+$'. The goal is to partition the vertices so as to minimize the number of inconsistent pairs where an edge (u, v) labeled '$+$' (resp., '$-$') incurs a unit cost if the two vertices u and v are placed into different groups (resp., the same group). This problem is closely related to FAST, and there exist simple and elegant 3-approximate randomized algorithms, called KWIK-SORT, for both problems, which can be seen as a generalization of quicksort: a randomly chosen pivot recursively divide an instance into two sub-instances. In the case of FAST, one subproblem contains the predecessors of the pivot and the other the successors of the pivot. Likewise, in the correlational clustering case, the vertices are divided based on their respective affinities to the pivot.

Aboud considered an outlier version of the Correlational Clustering problem and gave an $(O(1), O(1))$-approximation.[2] Not surprisingly, one can adapt his re-

[2] More precisely, he considered a slightly more general version where each vertex may have a different cost when removed as an outlier.

sult to obtain a $(O(1), O(1))$-approximation for FASTO. Unfortunately, Aboud's algorithm uses memory and running time that are at least linear in the number of 'bad' triangles, which can be as large as $\Omega(n^3) = \Omega(N^{1.5})$. In our problem, FASTO, a bad triangle v_1, v_2, v_3 is a triangle that does not admit a consistent ranking, i.e., $(v_1, v_2), (v_2, v_3), (v_3, v_1) \in E$ or $(v_2, v_1), (v_3, v_2), (v_1, v_3) \in E$.

To develop a fast and memory-efficient algorithm, we combine sampling with Aboud's algorithm. This combination is not trivial; for example, applying Aboud's algorithm to a reduced-sized input via sampling does not work. At a high-level, Aboud's algorithm uses a primal-dual approach. The approach sets up a linear programming (LP) and solves the LP by increasing the variables of the LP and its dual, where the constructed integer solution to the LP is used to identify outlier vertices. We have to adapt the LP, as we have to carefully handle the sampled points and argue their effects on potential backward edges. After all, we manage to reduce the memory usage and running time to $\widetilde{O}(N)$ preserving the approximation factors up to constant factors[3] under the assumption that the number of outliers is small, which we believe to be reasonable in practice.

Algorithms Adaptable to Massively Parallel Platforms. Finally, our algorithm can be easily adapted to run on massively parallel platforms such as MapReduce or Spark. On such platforms, each machine is assumed to have insufficient memory to store the whole input data, and therefore, multiple machines must be used. Aboud's algorithm is not suitable for such platforms, as it uses significantly super-linear memory. In contrast, our algorithm uses sampling appropriately to reduce the input size while minimally sacrificing the approximation guarantees. More precisely, our algorithm for FASTO can be adapted to run in $O(\log n)$ rounds on the parallel platforms – the number of rounds is often one of the most important measures due to the huge communication overhead incurred in each round.

As a byproduct, for the first time we show how to convert KWIK-SORT for FAST to the distributed setting in $O(1)$ rounds (see Sections 2.1 and 2.4), which is interesting on its own. The algorithm recursively finds a pivot and divides a subset of vertices into two groups, thus obtaining $O(\log n)$ rounds is straightforward. But we observe that as long as the pivot is sampled uniformly at random from each subgroup, the algorithm's performance guarantee continues to hold. Therefore, the algorithm still works even if we consider vertices in a random order as pivots – the sub-instance including the pivot is divided into two. Thus, we construct a decision tree from a prefix of the random vertex ordering, and using this decision tree, we partition the vertex set into multiple groups in a distributed way. This simple yet important observation also plays a key role in breaking bad triangles, thus reducing the memory usage of our algorithm for FASTO.

[3] We show that our algorithm is $(180, 180)$-approximate, which can be improved arbitrarily close to $(60, 60)$ if one is willing to accept a lower success probability. In contrast, Aboud's algorithm can be adapted to be $(18, 18)$-approximate for FASTO; however as mentioned above, it uses considerably more memory and run time than ours.

1.3 Other Related Work

The only problem that studies ranking with the possibility of removing certain outlier vertices, to our best knowledge, is the Feedback Vertex Set problem (FVS). The FVS problem asks to remove a minimum subset of vertices to make the remaining graph acyclic. It is an easy observation that FVS is a special case of our problem with $y^* = 0$ if the input graph were an arbitrary directed graph, not just a tournament. The FVS problem is known to be NP-hard and the current best approximation for the problem has an approximation factor of $O(\log n \log\log n)$ [13]. Thus, if we allow the input graph to be an arbitrary directed graph for our problem FASTO, then we cannot hope for a better than $(O(\log n \log\log n), c)$-approximation for any $c > 0$ unless we improve upon the best approximation for FVS. We are not aware of any other literature that considers rank aggregation with the possibility of removing outlier vertices, with the exception of the aforementioned Aboud's work on a closely related clustering problem [1]. Due to the vast literature on the topic, we can only provide an inherently incomplete list of work on ranking without outliers being considered. There exist several approximation algorithms for FAST. As mentioned, Ailon et al. [3] give the randomized KWIK-SORT that is 3-approximate for the problem, which can be derandomized [25]. Also, the algorithm that orders vertices according to their in-degrees is known to be a 5-approximation [11]. Kenyon-Mathieu and Schudy [18] give a PTAS; a PTAS is a $(1+\epsilon)$-approximate polynomial-time algorithm for any fixed $\epsilon > 0$. The complementary maximization version (maximizing the number of forward edges in the linear ordering) was also studied, and PTASes are known for the objective [14,7]. For partial rankings, see [2] and pointers in the paper. Extension to online and distributed settings are studied in [26] but the performance guarantee is not against the optimal solution, but against a random ordering, which incurs $\Theta(n^2)$ backward edges. For another extensive literature on stochastic inputs, see [4,19,5,23,12,24] and pointers therein.

1.4 Notation and Organization

We interchangeably use $(u, v) \in E$ and $u \prec v$ – we will say that u is v's predecessor, or equivalently, v is u's successor. We use $\tilde{O}$ or $\tilde{\Theta}$ to suppress logarithmic factors in the asymptotic quantities. We use n to denote the number of vertices in the input graph and $N = \Theta(n^2)$ to denote the asymptotic input size. The subgraph of G induced on V' is denoted $G[V']$. Let $[k] := \{1, 2, 3, \ldots, k\}$.

In Section 2, we present our algorithm for FASTO when the target number of outliers is small, i.e., $x^* \le \sqrt{n}$. We omit the other case in this paper due to space limit. In Section 3, we evaluate our algorithms and other heuristics we develop via experiments. In Section 4, we close with several interesting future research directions. Due to the space constraints, we will defer most analysis to the full version of this paper.

2 When the Target Number of Outliers x^* is Small

Our algorithm when $x^* \le \sqrt{n}$ consists of three main steps:

Step 1: Partitioning via KWIK-SORT-SEQ on Sample. Each vertex in V is sampled independently with probability $\frac{1}{4x^*}$ and placed into S. Randomly permute S and run KWIK-SORT-SEQ on the ordered set S to construct a decision tree $\tau(S)$. Let $k = |S|$. Partition the other vertices, $V \setminus S$, into $k+1$ groups, $V_1, V_2, \cdots, V_{k+1}$, using $\tau(S)$.
Step 2: Identifying Outliers via Primal-Dual. Formulate Linear Programming (LP) relaxation and derive its dual. Solve the primal and dual LPs using a primal-dual method, which outputs the set of outliers to be chosen.
Step 3: Final Ordering. Run KWIK-SORT on the non-outlier vertices in each group V_i, $i \in \{1, 2, \ldots k+1\}$.

In the following, we give a full description of all the steps of our algorithm; the last step is self-explanatory, and thus, is briefly discussed at the end of Section 2.2. Following the analysis of the algorithm, the extension to the distributed setting is discussed in the final subsection.

2.1 Step 1: Partitioning via KWIK-SORT-SEQ

We first present a sequential version of the original KWIK-SORT algorithm [3] which was described in a divide-and-conquer manner. As usual, there are multiple ways to serialize a parallel execution. So, as long as we ensure that a pivot is sampled from each subgraph *uniformly at random* for further partitioning, we can simulate the parallel execution keeping the approximation guarantee. Here, we introduce one specific simulation, KWIK-SORT-SEQ, which generates a random ordering of V, takes a pivot one by one from the random ordering, and gradually refines the partitioning. We show that this is indeed a valid way of simulating KWIK-SORT.

Algorithm 1 KWIK-SORT-SEQ ($G = (V, A)$)

```
 1: $\pi(V) \leftarrow$ a random permutation of $V$
 2: $\mathcal{V} = \{V\}$
 3: For $i = 1$ to $n = |V|$:
 4:     $\pi_i(V) \leftarrow i$th vertex in the ordering $\pi(V)$
 5:     Let $V' \in \mathcal{V}$ be such that $\pi_i(V) \in V'$
 6:     $V'_L, V'_R \leftarrow \emptyset$
 7:     For all vertices $j \in V' \setminus \{\pi_i(V)\}$:
 8:         If $(j, i) \in A$, then add $j$ to $V'_L$; else add $j$ to $V'_R$
 9:     Put $V'_L$, $\{\pi_i(V)\}$, $V'_R$ in place of $V'$, in this order
10: Return $\mathcal{V}$ (Order vertices in the same order they appear in $\mathcal{V}$).
```

Lemma 2. *KWIK-SORT-SEQ is a legitimate way of simulating KWIK-SORT, keeping the approximation guarantee. Therefore, KWIK-SORT-SEQ is a 3-approximation for* FAST.

Due to the space constraints, we defer the proof to the full version of this paper. Note that a fixed random permutation $\pi(V)$ completely determines the final ordering of vertices. Likewise, a fixed length-i prefix of $\pi(V)$ completely determines the intermediate partitioning $\mathcal{V}$ after $\pi_1(V)$, $\pi_2(V)$, …, $\pi_i(V)$ being applied, and the partitioning only refines as more pivots are applied. Thus, we can view this intermediate partitioning as the classification outcome of $V \setminus S$ via the decision tree $\tau(S)$ generated by KWIK-SORT-SEQ on the ordered set $S = \{\pi_1(V), \pi_2(V), \ldots, \pi_i(V)\}$. See Fig. 2 for illustration.

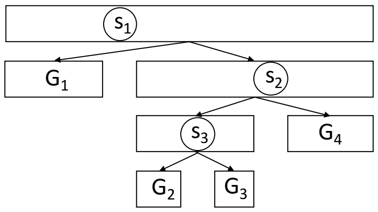

Fig. 2: Illustration of the construction of $\tau(S)$ and partitioning of $V \setminus S$ via $\tau(S)$. In this example, the decision tree $\tau(S)$ is induced by a(n ordered) sample $S = \{s_1, s_2, s_3\}$, where $s_1 \prec s_2$, $s_3 \prec s_2$, and $s_1 \prec s_3$. If a vertex $v \in V \setminus S$ is such that $s_1 \prec v$, $v \prec s_2$, $s_3 \prec v$, then v is placed into G_3.

The first step of our algorithm is essentially identical to what KWIK-SORT-SEQ does, except that our algorithm only needs a prefix of the random permutation, not the entire $\pi(V)$. It is an easy observation that sampling each vertex independently with the same probability and randomly permuting them is a valid way of getting a prefix of a random permutation. We note that we sample each vertex with probability $\frac{1}{4x^*}$, to avoid sampling any outlier (in the optimal solution) with a constant probability.

2.2 Step 2: Identifying Outliers

To set up our linear programming relaxation, we first need some definitions. We consider the ordered sample S and the induced groups $V_1, V_2, \cdots, V_{k+1}$ in the order they appear in the linear ordering produced by KWIK-SORT-SEQ performed in Step 1. For notational convenience, we reindex the sampled points so that they appear in the order of $s_1, s_2, \cdots, s_k$. We classify edges into three categories: Let E_{in} be the edges within the groups, S_{back} the backward edges with both end points in S, and E_{back} the backward edges $e = (u, v)$ such that $u \in V_i$, $v \in V_j$ for $i > j$; or $u = s_i$, $v \in V_j$ for $i \geq j$; or $v = s_i$, $u \in V_j$ for $i \leq j$. See Fig. 3.

Finally, we let T_{in} be the set of bad triangles with all vertices in the same group; recall that a bad triangle is a cycle of length 3. We are now ready to define an integer programming (IP) for a penalty version of FASTO, where a backward edge incurs a unit penalty and each outlier incurs $c := y^*/x^*$ units of penalty:

$$LP^{primal}_{fasto}(G) := \min \sum_{e \in E_{in}} y_e + \sum_{x \in V \setminus S} p_x \cdot c + \sum_{e \in E_{back}} z_e + |S_{back}| \qquad \text{(PRIMAL)}$$

$$s.t. \quad \sum_{e \subset t} y_e + \sum_{x \in t} p_x \geq 1 \qquad \forall t \in T_{in} \tag{1}$$

$$p_u + p_v + z_e \geq 1 \qquad \forall e = (u,v) \in E_{back} : \{u,v\} \cap S = \emptyset \tag{2}$$

$$p_u + z_e \geq 1 \qquad \forall e = (u,v) \in E_{back} : v \in S \wedge u \in V \setminus S \tag{3}$$

$$p_v + z_e \geq 1 \qquad \forall e = (u,v) \in E_{back} : u \in S \wedge v \in V \setminus S, \tag{4}$$

over variables $y_e \geq 0$ for all $e \in E_{in}$, $p_x \geq 0$ for all $x \in V \setminus S$, and $z_e \geq 0$ for all $e \in E_{back}$.

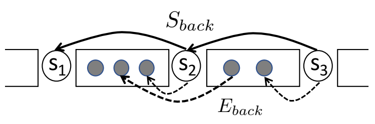

Fig. 3: Classification of edges. The rectangles shown represent groups V_1, V_2, V_3, V_4 from the left. Edges in S_{back} are shown as solid arcs and edges E_{back} as dotted arcs. Edges in E_{in} are those within groups and are omitted.

This IP has the following interpretation: an edge e in E_{in} (E_{back}, resp.) becomes backward if $y_e = 1$ ($z_e = 1$, resp.). Assuming that we will choose no sampled points as outliers, all edges in S_{back} will become backward. And each non-sampled point x incurs penalty c if it is chosen as an outlier when $p_x = 1$. Constraint (1) follows from the fact that for each bad triangle t, at least one edge e of t must become backward unless t is broken; a triangle gets broken when at least one of its vertices is chosen as outlier. The other constraints force each edge in E_{back} to become backward if its neither end point is chosen as outlier. A Linear Programming (LP) relaxation, which will be referred to as LP^{primal}_{fasto}, is obtained by allowing variables to have fractional values. Using the fact that KWIK-SORT is a 3-approximation, assuming that there is a feasible solution w.r.t. the target pair (x^*, y^*), conditioned on the sample being disjoint from the feasible solution's outlier, we can argue that the expected optimal objective of LP^{primal}_{fasto} is at most $4y^*$.

To obtain an approximate integer solution to LP^{primal}_{fasto}, we will use the primal-dual method. Primal-dual is a common technique for designing approximation algorithms [27]. The dual LP is shown below.

$$LP^{dual} = \max \sum_{t \in T_{in}} \alpha_t + \sum_{e \in E_{back}} \beta_e + |S_{back}| \tag{DUAL}$$

$$
\begin{aligned}
s.t. \qquad \sum_{e \subset t} \alpha_t &\leq 1 \quad \forall e \in E_{in} && (5)\\
\sum_{x \in t} \alpha_t + \sum_{x \in e} \beta_e &\leq c \quad \forall x \notin S && (6)\\
\beta_e &\leq 1 \quad \forall e \in E_{back} && (7)
\end{aligned}
$$

To develop an algorithm based on a primal-dual method, we replace Constraint (6) with two sufficient conditions (8).

$$\sum_{x \in t} \alpha_t \leq \frac{3}{5}c \qquad \forall x \in V \setminus S; \text{ and } \qquad \sum_{x \in e} \beta_e \leq \frac{2}{5}c \qquad \forall x \in V \setminus S \tag{8}$$

Below, we give our algorithm, **Algorithm 3**, that sets the variables of our primal with the help of the above dual. Although the algorithm updates all variables, the only information we need to run the final Step 3 is the outlier set U, as Step 3 runs KWIK-SORT on each group with vertices U removed as outliers. But the other outputs will be useful for the analysis of our algorithm. Note that although the dual variables can have fractional values, the primal variables will only have integer values. Note that E' and E^2 represent the backward edges within the groups and across the groups, respectively; since our algorithm samples no outlier vertices in the optimal solution, all edges in S_{back} become backward and they are counted separately.

Algorithm 2 Primal-Dual Algorithm

```
1: Initialization: $p \leftarrow 0, y \leftarrow 0, z \leftarrow 0, \alpha \leftarrow 0, \beta \leftarrow 0, U \leftarrow \emptyset, E' \leftarrow \emptyset, E^2 \leftarrow \emptyset$. Initially,
   all $\alpha_t$ and $\beta_e$ variables are active.
2: while $\exists$ active dual variables $\alpha_t$ or $\beta_e$ do
3:     Uniformly increase active dual variables until Constraints (5),
   (7), either of the two in (8) become tight.
4:     for each $e \in E_{in}$ s.t. $\sum_{e \subset t} \alpha_t = 1$, do add $e$ to $E'$; and
   inactivate $\alpha_t$ for all $t$ s.t. $e \subset t$.
5:     for each $x \in V \setminus S$ s.t. $\sum_{x \in t} \alpha_t = \frac{3}{5}c$, do add $x$ to $U$; and inactivate
   $\alpha_t$ for all $t$ s.t. $x \in t$
6:     for each $x \in V \setminus S$ s.t. $\sum_{x \in e} \beta_e = \frac{2}{5}c$, do add $x$ to $U$; and inactivate
   $\beta_e$ for all $e$ s.t. $x \in e$.
7:     for each $e \in E_{back}$ s.t. $\beta_e = 1$, do add $e$ to $E^2$; and inactivate $\beta_e$.
8: Remove from $E'$ and $E^2$ all the edges $e$ with $e \cap U \neq \emptyset$.
9: for $e \in E'$ do $y_e \leftarrow 1$;   for $\forall e \in E^2$ do $z_e \leftarrow 1$; and   for $\forall x \in U$ do $p_x \leftarrow 1$
10: return $p, y, z, U, E', E^2$
```

2.3 Sketch of the Analysis: Approximation Guarantees, Memory Usage, and Running Time

In this subsection, we only give a sketch of the analysis due to space constraints. In Step 1, we can show that the sample S is disjoint from the optimal solution's outlier set, with a constant probability (at least 3/4). Conditioned on that, as mentioned earlier, the expected optimal objective of LP^{primal}_{fasto} can be shown to be at most $4y^*$. The primal-dual method in Step 2 obtains an integer solution to LP^{primal}_{fasto} that is a constant approximate against the optimal LP objective, which is established by LP duality. Therefore, the outlier set U's contribution to the LP^{primal}_{fasto}'s objective is $c|U| = (y^*/x^*) \cdot |U|$ and it is upper bounded by $O(1)y^*$. This shows our algorithm chooses at most $O(1)x^*$ outliers. We now turn our attention to upper bounding the number of backward edges. The primal solution to LP^{primal}_{fasto} gives the number of backward edges within groups 'covering' all the unbroken triangles, which upper bounds the minimum number of backward edges achievable within groups [3], and counts the number of other backward edges explicitly by z_e and $|S_{back}|$. Since the latter is determined by the partial ordering produced by Step 1 and U and we run a 3-approximate KWIK-SORT on each group, we can also establish that the final number of backward edges output is $O(1)y^*$.

Now we discuss our algorithm's memory usage and running time. We show that each group size is $\tilde{O}(n/k)$ if $|S| = \Omega(\log n)$. This requires us to prove that a randomly chosen pivot partitions a problem into two subproblems of similar sizes with a constant probability, meaning that there is a large fraction of vertices that have similar numbers of predecessors and successors. Then, the total number of bad triangles within groups is $(\tilde{O}(n/k))^3 \cdot k = \tilde{O}(n^2) = \tilde{O}(N)$ when $k \simeq \frac{n}{x^*} \geq \sqrt{n}$, as desired. Since the number of variables in our algorithm, particularly in Step 2 is dominated by the number of bad triangles in consideration and edges, it immediately follows that the memory usage is $\tilde{O}(N)$. Further, one can increase dual variables by a factor of $(1+\epsilon)$ in each iteration for an arbitrary constant precision parameter $\epsilon > 0$, starting from $1/n^2$. Using this one can show the number of iterations needed is $O(\log n)$. This immediately leads to the claim that the running time is $\tilde{O}(n^2) = \tilde{O}(N)$ when $x^* \leq \sqrt{n}$.

2.4 Extension to the Distributed Setting

Due to space constraints, in this subsection, we briefly discuss how we can adapt the algorithm to run in a distributed way. For formal theoretical models of the distributed setting we consider, see [6]. We assume that the input graph is stored across machines arbitrarily. Clearly, Step 1 of taking a sample S can be done in parallel. All edges between points in S are sent to a machine to construct the decision tree $\tau(S)$. The decision tree is broadcast to all machines to partition vertices in $k+1$ groups in a distributed way. If FAST were the problem considered, we would sample each vertex with probability $1/\sqrt{n}$ and move the subgraph induced on each group G_i to a machine and continue to run KWIK-SORT on the subgraph. Then, we can implement KWIK-SORT to run in $O(1)$ rounds assuming that each machine has $\tilde{\Omega}(n)$ memory. If FASTO is the problem considered, we can implement Step 2 in $O(\log n)$ rounds, as discussed in the previous subsection. Step 3, which is the execution of KWIK-SORT on each group, can be run in one round. Again, the only constriant is that each machine has $\tilde{\Omega}(n)$ memory for an arbitrary number of machines.

3 Experiments

In this section, we perform experiments to evaluate our algorithm against synthetic data sets. All experiments were performed on Ubuntu 14.04 LTS with RAM size 15294 MB and CPU Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz. We implemented the following four algorithms including ours for comparison. The last two (RSF and IOR) are new heuristics we developed in this paper, but with no approximation guarantees. We assume that all algorithms are given a 'budget' B on the number of outliers, which limits the number of vertices that can be chosen as outliers.

- Primal-Dual with Sampling (PDS): Our algorithm when x^* is small. We run the algorithm for all target pairs (x^*, y^*) of powers of two, where $x^* \in [0, B], y^* \in [0, B']$, and choose the best solution with at most $1.5B$ outliers. Here, B' is the number of back edges output by KWIK-SORT. Note that we allow PDS to choose slightly more outliers although it may end up with choosing less since the only guarantee is that PDS chooses up to $O(x^*)$ outliers. The running time is summed up over all pairs.
- Aboud's algorithm (ABD): The algorithm in [1] for Correlational Clustering is adapted to FASTO. As above, the best solution with at most $1.5B$ outliers is chosen over all the above target pairs. ABD is essentially PDS with $S = \emptyset$ in Step 1. The running time is again summed up over all pairs considered.
- Random Sample Filter (RSF): Take a random Sample S from V (for fair comparison, the same sample size is used as in PDS). Order S using KWIK-SORT and let $\pi(S)$ be the resulting order. For each $v \in V \setminus S$, let $b(v)$ be the minimum number of backward edges with v as one endpoint over all these $|S| + 1$ possible new permutations created by adding v to $\pi(S)$. Outputs B vertices $v \in V \setminus S$ with the highest $b(v)$ values as the outliers and order the remaining vertices by KWIK-SORT.
- Iterative Outlier Removal (IOR): We iteratively remove the vertex without which KWIK-SORT outputs the least back edges. Initially, $U = \emptyset$. In each iteration, for each vertex $v \in V$, run KWIK-SORT on $V \setminus v$, which yields $|V|$ permutations. Among all the achieved permutations, consider the one with minimum number of backward edges. Let $v \in V$ be the missing vertex in this permutation. Add v to U; and $V \leftarrow V \setminus \{v\}$. Stop when $|U| = B$. Then, output U as outliers and run KWIK-SORT on V/U.

We mainly use two natural models to generate synthetic data sets. The first model, which we call the *uniform* model, assumes inconsistencies uniformly scattered over edges, in addition to randomly chosen outlier vertices. More precisely, the uniform model is described by a quadruple $\langle p, q, r, n \rangle$: For a tournament $G = (V, E)$ over n vertices with no inconsistencies, flip each edge with probability p and perturb each vertex v with probability q by flipping each edge incident on v with probability r – all independently. The second model, which essentially follows the *Bradley-Terry-Luce* model [8,20] and therefore we refer to as BTL, assumes that each vertex i has a certain (hidden) score $w_i > 0$. Then, each pair of vertices i and j has edge (i, j) with probability $\frac{w_j}{w_i+w_j}$; or edge (j, i) with the remaining probability. Intuitively, if edge (i, j) is present for every i, j such that $w_i < w_j$, we will have a tournament that is a DAG. However, some edges are flipped stochastically – in particular, edges between vertices with similar scores are more likely to be flipped. Assuming that the underlying vertex scores are a geometric sequence, we can compactly describe a BTL instance by a quadruple $\langle b, q, r, n\rangle$, where the score of the n vertices forms a geometric sequence of ratio b. Then, edges are first generated as described above, and vertices are perturbed using the parameters q and r as are done for the uniform model.

We first confirm that ABD is not very scalable.

Table 1: PDS withsample size $1/(2q) = 50$ vs. ABD for the uniform model $\langle p = 0.001, q = 0.01, r = 0.50, n\rangle$ and $B = nq$. Bad $\triangle$s denotes the number of bad triangles in 1000 units.

pts (n)	outliers		back edges		bad $\triangle$s		time (sec)	
	PDS	ABD	PDS	ABD	PDS	ABD	PDS	ABD
250	0	2	269	33	0	18	0.2	3.8
500	6	7	377	152	365	228	4.8	100.0
1000	9	N/A	1597	N/A	441	1032	20.0	600+

Even when $n = 1000$, ABD does not terminate in 600 seconds while our algorithm PDS does in 20 seconds. Our algorithm's speed-up is explained by the significantly smaller number of bad triangles. For $n = 500$, PDS outputs factor 2 or 3 more back edges than ABD. But we were not able to compare the two algorithms for larger inputs because of ABD's large run time.

Next, we compare PDS to RSF and IOR for inputs generated under the uniform and BTL models. Note that RSF and IOR choose exactly the same number of outliers, B.

Table 2: PDS with sample size $1/(2q) = 50$ vs RSF vs IOR for the uniform input $\langle p = 0.001, q = 0.01, r = 0.50, n\rangle$ and $B = nq$.

pts (n)	outliers		back edges (10^3)			time (sec)		
	RSF	PDS	RSF	IOR	PDS	RSF	IOR	PDS
500	5	6	1.1	0.98	0.4	0.1	0.1	4.7
1000	10	9	2.6	1.8	1.6	0.4	0.5	20.3
2000	20	16	7.5	6.1	2.5	1.1	3.8	145.4
4000	40	55	26.2	71.5	11.8	3.8	41.3	1132

Table 3: PDS with sample size $1/(2q) = 50$ vs RSF vs IOR for the BTL input with $q = 0.01, r = 0.50$, $(b, n) \in \{(4.2, 500), (2.3, 1000), (1.57, 2000), (1.266, 4000)\}$; here $B = nq$.

pts (n)	outliers		back edges (10^3)			time (sec)		
	RSF	PDS	RSF	IOR	PDS	RSF	IOR	PDS
500	5	1	0.3	0.0	0.3	0.1	0.1	1
1000	10	9	0.8	0.3	0.8	0.3	0.5	8.3
2000	20	26	7.9	7.6	1.6	0.9	2.3	73
4000	40	33	8.1	8.0	7.9	3.4	15.9	381.3

Our algorithm PDS outperforms the other two in terms of the number of back edges although it occasionally chooses slightly more outliers. However, PDS is considerably slower than RSF and IOR.

Finally, to showcase the major advantage of our algorithm PDS that it has performance guarantees for all inputs, in contrast to the two heuristics RSF and IOR, we consider certain specific instances. First, we observe that RSF significantly underperforms compared to PDS and IOR when the instance is constructed by choosing $\sqrt{n}$ points at random and flipping edges among them. As before, note that RSF and IOR choose the same number of outliers, B, thus we only display RSF for the outliers column.

Table 4: PDS with sample size $\sqrt{n}/2$ vs RSF with sample size $\sqrt{n}/2$ vs IOR. Each vertex is perturbed with probability $1/\sqrt{n}$ – each edge between perturbed vertices is flipped with probability 1/2. $B = \sqrt{n}$.

pts (n)	outliers		back edges			time (sec)		
	RSF	PDS	RSF	IOR	PDS	RSF	IOR	PDS
1000	31	31	226	0	0	0.2	1.2	6.4
2000	44	44	541	0	0	0.8	7.3	27.2
4000	63	63	1570	0	0	3.3	62.8	130.2
8000	89	89	2887	0	0	13.4	442.0	702.7

As shown in Table 4, when $n = 4000$, all algorithms choose exactly 63 outliers; but RSF produces 1570 back edges while the other two produce no back edges. For all cases when $n = 1000, 2000, 4000$ and 8000, PDS and IOR create no back edges while RSF does a considerable number of back edges. Interestingly, IOR appears not to be very scalable. For $n = 8000$, IOR is only twice faster than PDS.

We continue to observe that IOR also quite underperforms compared to PDS for a certain class of instances. The instance we create is parameterized by t. The instance is constructed by combining $4t$ sub-tournaments, G_1, G_2, G_3, $\cdots$, G_{4t}, which are identical. Each G_i has t vertices and admits a perfect ordering with one vertex removed therein – each edge in G_i incident on the vertex is flipped with probability 1/2. We connect the sub-tournaments, so that for any $i < j$, all vertices in G_i are predecessors of all vertices in G_j. As shown in Table 5, when $n = 4096$, PDS creates no back edges while IOR creates 313 back edges; both chooses the same number of outliers, 128. Further, PDS is slower than IOR only by a factor of at most 3.

Table 5: PDS with sample size $t/2$ vs IOR

pts (n)	outliers		back edges		time (sec)	
	IOR	PDS	IOR	PDS	IOR	PDS
1024	64	62	87	7	2.4	21.4
2116	92	89	142	79	16.7	203.2
4096	128	128	313	0	129.5	305.5

4 Conclusions

In this paper, we studied how to order objects in the presence of outliers. In particular, we developed approximation algorithms that are nearly optimal in terms of running time and can be adapted to the distributed setting, along with potentially useful heuristics. There are many interesting future research directions. First, our algorithm may choose more than x^* outliers. It would be interesting if one can get an approximation algorithm that finds an ordering resulting in $O(1)y^*$ backward edges with strictly no more than x^* outliers. Second, we currently do not know if it is possible to obtain a $(O(1), O(1))$-approximation algorithm whose running time is almost linear in the input size when $x^* > \sqrt{n}$. Finally, it would be of significant interest to consider arbitrary directed graphs as input.

Acknowledgements

This work was supported in part by NSF grants CCF-1409130 and CCF-1617653.

References

1. Aboud, A.: Correlation clustering with penalties and approximating the reordering buffer management problem. Master's thesis. The Technion Israel Institute of Technology (2008)
2. Ailon, N.: Aggregation of partial rankings, p-ratings and top-m lists. Algorithmica **57**(2), 284–300 (2010)
3. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: ranking and clustering. Journal of the ACM **55**(5), 23 (2008)
4. Altman, A., Tennenholtz, M.: Ranking systems: the pagerank axioms. In: ACM EC (2005)
5. Ammar, A., Shah, D.: Ranking: Compare, don't score. In: IEEE Allerton (2011)

6. Andoni, A., Nikolov, A., Onak, K., Yaroslavtsev, G.: Parallel algorithms for geometric graph problems. In: ACM STOC. pp. 574–583 (2014)
7. Arora, S., Frieze, A., Kaplan, H.: A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. Math programming **92**(1), 1–36 (2002)
8. Bradley, R.A., Terry, M.E.: Rank analysis of incomplete block designs: I. the method of paired comparisons. Biometrika **39**(3/4), 324–345 (1952)
9. Charikar, M., Khuller, S., Mount, D.M., Narasimhan, G.: Algorithms for facility location problems with outliers. In: ACM-SIAM SODA (2001)
10. Chen, K.: A constant factor approximation algorithm for k-median clustering with outliers. In: ACM-SIAM SODA (2008)
11. Coppersmith, D., Fleischer, L.K., Rurda, A.: Ordering by weighted number of wins gives a good ranking for weighted tournaments. ACM Transactions on Algorithms **6**(3), 55 (2010)
12. Duchi, J.C., Mackey, L.W., Jordan, M.I.: On the consistency of ranking algorithms. In: ICML. pp. 327–334 (2010)
13. Even, G., (Seffi) Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multicuts in directed graphs. Algorithmica **20**(2), 151–174 (Feb 1998)
14. Frieze, A., Kannan, R.: Quick approximation to matrices and applications. Combinatorica **19**(2), 175–220 (1999)
15. Guha, S., Li, Y., Zhang, Q.: Distributed partial clustering. In: ACM SPAA (2017)
16. Gupta, S., Kumar, R., Lu, K., Moseley, B., Vassilvitskii, S.: Local search methods for k-means with outliers. PVLDB **10**(7), 757–768 (2017)
17. Kemeny, J.G.: Mathematics without numbers. Daedalus **88**(4), 577–591 (1959)
18. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: ACM STOC (2007)
19. Lu, T., Boutilier, C.: Learning mallows models with pairwise preferences. In: ICML (2011)
20. Luce, R.D.: Individual Choice Behavior a Theoretical Analysis. John Wiley and sons (1959)
21. Malkomes, G., Kusner, M.J., Chen, W., Weinberger, K.Q., Moseley, B.: Fast distributed k-center clustering with outliers on massive data. In: NIPS (2015)
22. Muller, E., Sánchez, P.I., Mulle, Y., Bohm, K.: Ranking outlier nodes in subspaces of attributed graphs. In: IEEE ICDEW (2013)
23. Negahban, S., Oh, S., Shah, D.: Rank centrality: Ranking from pairwise comparisons. Operations Research **65**(1), 266–287 (2016)
24. Rajkumar, A., Agarwal, S.: A statistical convergence perspective of algorithms for rank aggregation from pairwise data. In: ICML (2014)
25. Van Zuylen, A., Williamson, D.P.: Deterministic algorithms for rank aggregation and other ranking and clustering problems. In: WAOA. pp. 260–273. Springer (2007)
26. Wauthier, F., Jordan, M., Jojic, N.: Efficient ranking from pairwise comparisons. In: ICML (2013)
27. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press (2011)