An Empirical Study of Top-Lists Aggregation Methods

Ammar Eltigani and Tai Henrichs

Abstract

The Kemeny rank aggregation problem is to, given rankings over candidates submitted by voters, output the ranking of all candidates which minimizes the average Kendall Tau distance across all voters' rankings. When voters' are top-lists or full-lists, finding such a list is NP-Hard, motivating the search for good approximations and heuristics. We empirically evaluate a number of algorithms for solving *top-lists* aggregation. Approximation algorithms are sourced from [8], and (unproved) heuristics are sourced from [2], a paper with similar scope and methodology to ours, but that is exclusively concerned with full-lists aggregation. Algorithms are assessed on both synthetically generated and real-world data-sets.

Keywords: Top-Lists, Rank-Aggregation, Approximation Algorithms, Kendall-Tau, Kemeny-Young, Mallows Model, Empirical Study.

1 Introduction

Rank aggregation stems from a long line of research in the field of computational social choice. With its many variants and formulations (incomplete and partial lists, bucket orderings, p-ratings, etc. [1]), it has proved to be an important and useful problem to study because of its desirable properties due to the *Kemeny Rule*.

The Kemeny rule is a voting rule that aims to minimize the Kendall Tau distance[1], a distance function that measures pairwise disagreements between candidates' ranking over the input of preference lists given by voters. Rank aggregation is generally the problem of minimizing the Kendall Tau distance between all the voters' preference lists (input) and a single full ranking of the candidates (output). Solving this problem efficiently is important because it guarantees a Condorcet consistent[2] voting rule. And because of its nice properties (the output ranking has a natural interpretation as the maximum likelihood ranking under a very simple statistical noise model proposed by Condorcet called the *Mallows Model*), rank aggregation is also often used in machine learning, web content ranking, collaborative filtering, meta-searching, and similarity search [7, 2].

Unfortunately, rank aggregation has been proven to be NP-Hard even for a problem instance with only 4 candidates [3]. So it is not difficult to see that top-lists aggregation, a more general variant to rank aggregation[3] where voters are free to rank a subset of all candidates, is also NP-Hard. Because of these results we turn our focus in this project to evaluate a wide variety and class of approximation algorithms based on different use cases and data set parameters one would usually care about for the top-lists aggregation problem.

[1] Sometimes also referred to as 'bubble-sort distance' because it represents the number of bubble sort swaps needed to convert one preference list into the other in a pairwise comparison of two preference lists.

[2] Condorcet consitency: If there exists a candidate who would win a two-candidate election against each of the other candidates in a plurality vote, they are always ranked in the first (best) position.

[3] We do not distinguish between 'full-lists aggregation' and 'rank aggregation'

The rest of this report is structured as follows: in Section 1.1 we formalize the top-lists aggregation problem, in Section 1.2 we summarize the existent literature, in Section 2 we highlight our work and contributions, in Section 3 we provide a short description of each approximate algorithm we considered alongside some implementation notes when applicable, in Section 4 we introduce the data sets and relevant parameters we used in our experiments, in Section 5 we visualize and analyze the performance of our algorithms, and finally in Section 6 we summarize our findings.

1.1 Problem Definition

Let $A = \{1, \dots, n\}$ be the set of all the candidates and $\pi = [\pi_1, \dots, \pi_N]$ be the input.

Definition 1 (Full-List) *A Full-List or complete ranking σ_i is a strict ordering of all n candidates. We say that σ_i is the preference list[4] of voter i and that $\sigma_i(j)$ is the position of candidate j in σ_i where 1 is the most preferred position.*

Definition 2 (Top-List) *A Top-List π_i is a full-list that ranks **up to** n candidates. If U is the set of candidates not ranked by voter i, $U = A \setminus \pi_i$, then each pair of candidates $u, u' \in U$ are tied with each other $\pi_i(u) = \pi_i(u') = \infty$. Additionally, any candidate $u \in U$ is less preferred to any candidate $x \in \pi_i$, $\pi_i(u) > \pi_i(x)$.*

Definition 3 (Generalized Kendall Tau) *Let π_i and π_j be two top-lists. The Generalized Kendall Tau distance function is defined as*

$$\mathcal{K}(\pi_i, \pi_j) = \sum_{r=1}^{n} \sum_{s=r}^{n} \mathbf{1}\big(\pi_i(r) < \pi_i(s) \quad \& \quad \pi_j(s) < \pi_j(r)\big)$$

where the value of the inner term is either 1 or 0 depending on the conditional. Intuitively, this is just counting the regular Kendall Tau distance while disregarding ties since we have no information about their relative order. In other words, only a pair of disagreeing candidates that are ranked in both lists contribute to the cost function.

In practice, in order to compute the Generalized Kendall Tau distance between a given an output full ranking σ and a top-list π_i, we perform the transformation $f : \pi_i \to_\sigma \tau_i$ where f simply takes all the candidates that are not ranked in π_i and appends them to the back of π_i in the order that they appear in σ. Now we have reduced the problem to simply computing the regular Kendall Tau distance (over full lists).

Definition 4 (Top-Lists Aggregation) [5] *Given A and π, the Top-Lists aggregation problem is thus finding some σ_0 as the minimize the following cost function*

$$\mathcal{C}(\pi, \sigma_0) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{K}(\pi_i, \sigma_0)$$

[4] This also means that there are a total of N voters and π_i is not necessarily unique.

[5] There exists a variant to this problem called *top-k lists* aggregation in which all top-lists must be of equal length $k \leq n$.

A	The set of all candidates $\{1, \ldots, n\}$
π	Total input top-lists
π_i	Voter i's top-list
$\pi_i(j)$	The position of candidate j in voter i's preference list
$\mathcal{K}(\pi_i, \pi_j)$	Kendall Tau distance between π_j and π_i
$\mathcal{C}(\pi, \sigma_0)$	Cost of σ_0 over the input π

Table 1: Summary of Notation

Motivation We believe that top-lists aggregation is a more feasible and abundant problem in the real world when compared to full rank aggregation. Take for example the context of social choice. It is unlikely that a voter might have any reasonable preference over candidates that exceed his/her top 20 or so most preferred. Your average voter will have difficulty in deciding which to rank ahead of the over, candidate 57 or 58. This natural indifference is modelled better by simply saying that all non-top candidates are tied within each other but less preferred to any candidates that are ranked. A similar argument can be make to Web content aggregation as the probability that a user click on search results on the second or third page is pretty low. We suspect that such added freedom on the voter's side, although not reducing the actual computational complexity, results in better run times compared to full-rank aggregation[6] and give a better notion of accuracy since the disagreements incurred by the irrelevant[7] preferences are not counted towards the total cost in [8] and our formulation of top-lists.

1.2 Background and Related Work

As mentioned before, the existing literature on the subject of rank aggregation is very extensive. There are many papers that consider this general problem for specific applications, like aggregating web content [4] and others that introduce new variants allowing *ties*, *rating*, and different notions of *partial (incomplete) lists* with respectively different distance functions [6, 1]. However, we have found that there is only one recent paper in the theoretical realm that bounded a series of full-rank approximation algorithms and adopted them to top-lists.

How To Aggregate Top-Lists [8] The authors in this paper generalize the SpearmanFootrule algorithm and the RepeatChoice algorithm from [1]. They also analyze a Borda approach for rank aggregation using *average ranks* and *scores*. They handle a special case differently where the input lists have length at most k. And they prove that most of their algorithms are 2-approximations, despite having consistently stronger performance in our assessment.

Experiments with Kemeny Ranking [2] The authors here empirically evaluate a number of algorithms for outputting an approximate or exact Kemeny-ranking when provided with full rankings as inputs. They find that, in practice, the best algorithms to use can vary with the degree of *consensus* amongst the rankers. The greater the consensus amongst the rankers, the more tractable the problem. They also determine, for varying consensus-regimes, the relationship

[6] This is intuitive since there is less information in total and thus less disagreements between candidate pairs

[7] By irrelevant here we mean with marginal difference or indifference by the ranking agent in a full-list

between performance and accuracy across different algorithms. Their methods guide our own approach to empirically analyzing top-list aggregation.

2 Contributions

The main contribution of this experimental project is our implementation and comparison of some 15 total approximation algorithms for the top-lists aggregation problem. They are all outlined in the next section. To the best of our knowledge, these algorithms have not been implemented for the top-lists variant and some do not even have obvious counterparts[8]. We were naturally attracted to this project as we came to realize that we could replicate the empirical testing methodology found in [2] over full-lists, to top-lists, using the algorithms found in [8] as a foundation and adding some more.

Besides the approximation algorithms, we implemented a Integer Linear Program (ILP) optimal solution for computing a ground truth distance. Although we did find an open-source ILP implementation for general rank aggregation in python, it was either cumbersome to use for top-lists and did not perform as well because it used a slower solver. We also wrote a simulation/testing suite and visualization pipeline in which one would easily be able to replicate our results by cloning a repository and running one or two commands. Finally, we extended some existing code for the Mallows Model on top-k lists, the case where all top-lists must be of length k, to the case where each top-lists can be any size up to n.

3 Algorithms

In this section we provide a high-level overview of the algorithms we implemented as well as introduce some concepts and definitions that help us understand them better.

3.1 Optimal

We introduce an alternate representation to the input $\pi = [\pi_1, \pi_2, \ldots, \pi_N]$ called the *precedence matrix*, which is a requisite for implementing an LIP solution, but also turns out to be a crucial factor in ensuring that many of the subsequent algorithms are efficient. We follow the notation in [2]:

Definition 5 (Precedence Matrix) *The precedence matrix $Q = [Q_{ab}]_{a,b=1:n}$ is defined by:*

$$Q_{ab} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\pi_i(a) < \pi_i(b))$$

Q_{ab} intuitively represents the number of top-lists that rank a before b and Q_{ba} that rank b before a. For an incomplete top-list π_i, if a is not explicitly ranked, then any candidate b that is explicitly ranked is considered to be ranked before a. A consequence is that the diagonal of Q is zero (a candidate is tied with himself) and $Q_{ab} + Q_{ba} = N$ for all $a, b \in A$.

[8] for which we made some assumptions that we explicitly mention in the next section

Integer Linear Program The optimal solution is found with the algorithm we call "opt," which solves the following integer program exactly:

- Minimize:

$$\mathcal{C}(\pi, \sigma_0) = \frac{1}{N} \sum_{\forall a,b} Q_{ab} x_{ba} + Q_{ba} + x_{ab}$$

- Constraints:

$$x_{ab} \in 0, 1, \forall a, b \tag{1}$$
$$x_{ab} + x_{ba} = 1, \forall a, b \tag{2}$$
$$x_{ab} + x_{bc} + x_{ca} \geq 1, \forall a, b, c \tag{3}$$

This is the same formulation used in [2]. The only difference is in the definition of the precedence-matrix Q. The given program minimizes averages Kendall-Tau distance as its goal. The first constraint enforces the fact that either a precedes b, or they do not. The second constraint enforces the fact that the output must be a complete, strict ranking, meaning either a precedes b, or b precedes a. The third constraint enforces transitivity over the ranking of candidates.

Relaxed Linear Program A linear-program relaxation of the above integer-program is also implemented. The sole difference is that the first constraint above is replaced with the following:

$$x_{ab} \in [0, 1], \forall a, b \tag{4}$$

3.2 Distance Heuristics

We consider three algorithms based on other distance rules such as Borda, Spearman FootRule, and Condorcet. First, however we define two important concepts that [8] uses to account not only for the relative positional ranking of candidates, but also each candidate's total frequence of being ranked.

Definition 6 (Scores) *The score of candidate j is defined as the probability that j is ranked in a randomly sampled top-list*

$$score_j = \frac{\#\ of\ top\text{-}lists\ that\ rank\ j}{N}$$

Definition 7 (Average Ranks) *The average rank of a candidate is simply their Borda points divided by their score*[9]*. This adjustment by score is an important concept for which we suspect much of the success of the Borda-based algorithms is due to*[10]*. It is defined as*

$$rank_i = \frac{1}{score_i} \sum_{r=1}^{n} \big(\#\ of\ top\text{-}lists\ that\ rank\ i\ in\ position\ r\big) \cdot r$$

[9]Conditioned on them appearing at list in one top-list (else their average rank is defined to be ∞).

[10]For two candidates with similar Borda points but different scores, the candidate with a greater score is placed ahead of the candidate with smaller score.

Footrule+ A 2-approximaton that computes a cost matrix C, where for each candidate i and each rank j, the Spearman's FootRule distance is defined as $C(i,j) = \sum_{r=1}^{j}(j-r)\cdot(\#$ of top-lists that rank i in j) And then performs the Linear Assignment optimization[11] to find a min-cost matching between candidates and ranks.

Borda+ A $O(n)$-approximation that sorts all candidates by increasing rank

Score-Then-Borda+ A $O(8e+4)$-approximation that first partitions candidates into groups with similar scores, performs the Borda+ procedure within each group, and finally outputs the concatenations of the candidates of each group (starting from the group with highest scores) in the Borda order.[12]

Copeland Sorts the candidates by the decreasing number of pairwise contests each candidates has won. Our implementation breaks ties at random (in lexicographic order) but it would be interesting to see if we can further improve performance by breaking ties based on scores.

3.3 Sorting Algorithms

Quick-Sort, Insertion-Sort Sort an input full ranking according to the precedence matrix Q such as for two candidates a and b, a appears before b in the final output ranking if $Q_{ab} > Q_{ba}$. We use different sorting algorithms because the order in which theses comparisons are made affects the final ranking - this is because pair-wise contest success is not transitive, that is, Condorcet's Paradox. Quick-Sort has its pivot selected by one of two methods. For the algorithm we call QS-det, the pivot is chosen by evaluating all pivots and picking the one that minimizes pairwise disagreements at each step. We also consider the variant QS-rand that chooses a pivot at random, which on average ran faster than DS-det. Insertion-Sort is the standard insertion sort algorithm, so we decline to describe it, though we call it IS later in the paper.

3.4 Top-k Algorithms

Score-Then-Adjust Is an Efficient Polynomial-Time Approximation Scheme [13] that, given some ϵ and k, first sorts the candidates in decreasing score then brute force permutes the first $\lceil(1+\frac{1}{\epsilon})(k-1)\rceil$ candidates to find the permutation that minimizes pairwise disagreements over the whole rank profile. We use the LIP code instead of performing brute force permutations in order to gain speed.

3.5 Greedy

Local-Search Given some full ranking this greedy procedure works by iteratively moving each item in the full ranking to the position that yields a candidate solution with the fewest number of pairwise disagreements with the profile; if none of the items are moved from their positions, then the algorithm terminates.

[11] Also known as Hungarian algorithm. We found that a scipy library function was easier to work with and suspect that it is actually a network flow reduction instead of the Hungarian algorithm.

[12] This algorithm uses a random variable [0,1) to partition the candidates based on their scores.

[13] Efficient Polynomial-Time Approximation Scheme: algorithms that are $(1+\epsilon)$ approximations for $\epsilon > 0$, with time-complexity polynomial in the input size, independent of ϵ.

Chanas Given some full ranking this greedy procedure works by iteratively moving candidates at positions i to positions $j < i$ as long as doing so decreasing its pairwise disagreements. After one pass, the ranking is reversed and the same procedure is repeated before outputting the final solution.

Combinations In [2], the two greedy procedures above are shown to improve accuracy for little extra time when given as input the output full ranking σ_0 of some other algorithm. We investigate this for the top-lists case by executing Chanas and Local-Search upon on the outputs of all algorithms described above. This includes running Chanas on the output of Local-Search and vice versa. Including combinations, we consider a total of 28 algorithms.

4 Data

4.1 Synthetic Data

We generated synthetic data using the *Mallows Model*, which produces an exponential distribution of rankings. The original Mallows Model was defined for distributions of complete rankings, so we utilize the generalization of the Mallows Model to the setting with top-k-lists defined by [5]. For a Mallows Model, s_0 is the $ground - list$, which is the model's peak, if there is one. When $\theta = 0$, no peak exists because the data is completely uniform. As θ increases, rankings become increasingly centralized around s_0. Thus, for $\theta > 0$, the Mallows Model can represent situations where there is some *consensus* about which voters generally agree, the ordering provided by s_0.

The model as defined in [5] provides that all top-lists are top-k-lists for some k; however, this may not be the case in real situations.[14] We use a *Poisson distribution* to model the lengths of voters' top-lists. The choice is somewhat arbitrary, but the following properties motivate its selection:

- It is defined for $[0, \infty)$, that is, excludes negative values, which are invalid list lengths.
- It contains only integral values, which are the only valid list lengths.
- It models the fact that you would expect a distribution of top-lists to have a peak, which forms approximately at the rate parameter λ on which the distribution is defined.

For each of the n voters, we sample voter i's top-list length l_i from a Poisson distribution defined on λ, where λ is the average top-list length.[15] Then for each i, a top-l_i-list π_i is sampled from a Mallows Model where $k = l_i$.

4.2 Real-world Data

Algorithms were also evaluated on real-world data from PrefLib, specifically the "Ski Jumping" data-set from library-entry ED-00010: F1 and Skiing [9]. Each "voter" is one of four Ski Jumping races from the 2006-2009 Skiing World Championships, so each submitted top-list is the ranking induced by a races' results. Each "candidate" is one of 170 skiiers. Those that did not compete

[14] For example, in electoral settings, different voters may rank differing numbers of candidates, perhaps with more politically engaged voters explicitly ranking more candidates.

[15] We actually sample all l_i simultaneously, repeatedly sampling until $n \geq l_i > 0, \forall i$.

in a given race are implicitly ranked after those that competed since they failed to qualify for the World Championship, inducing a top-list. Because races had differing numbers of competitors, the resulting rankings are of varying lengths. Complete rankings of sports competitors are commonly made, so we believe this data represents a realistic use-case.

5 Experiments

We run a combinatorial experiments over all the algorithms listed in Section 3 except for the top-k EPTAS. Our parameter values chosen for those experiments are summarized in Table 2. For the EPTAS, we drop the Poisson distribution and generate top-k data only. The three top-k datasets, each of varying difficulty, are characterized in Table 3. For each of these datasets, we run Score-Then-Adjust over all epsilons $[.25, .5, .75, 1]$.

Our code was implemented in 64-bit Python and ran on a computer with a 6-Core, 12 thread, 3.3GHz Intel Core i7 5820K. The Integer-Program and its Linear-Programming Relaxation were solved using the Python API provided by Gurobi.

n	Total number of candidates $= \|A\|$	$10, 30, 50$
N	Size of input $= \|\pi\|$	$50, 500, 5000$
θ	Consensus parameter for Mallows Model $[0, \infty)$	$.001, .01, .1$
k-ratio	k equals k-ratio times n for a given experiment	$.1, .5, .9$

Table 2: Combinatorial Parameters

Easy	$n = 10$	$N = 50$	$\theta = 0.1$	$k = 1$
Medium	$n = 30$	$N = 500$	$\theta = 0.01$	$k = 15$
Hard	$n = 50$	$N = 5000$	$\theta = 0.001$	$k = 45$

Table 3: Top-k Parameters

Besides experiments on synthetic data, we evaluate algorithms on the ski competition data previously described. Note that because top-lists are of varying lengths, the top-k EPTAS cannot be evaluated on the ski data.

5.1 Analysis & Results

We now present our experimental results. Results for Kendall Tau distances are normalized relative to the optimal solution, meaning an algorithm with cost c has an average Kendall Tau distance c times larger than the optimal cost. Note that for all graphics, we have excluded algorithms that were Pareto-dominated for the data being considered. In this context, an algorithm a_1 is Pareto-dominated by a_2 iff a_2 is at least as fast as a_1 with a lower cost, or a_2 has a cost no greater than a_1 and a lower time. If some algorithms are Pareto-dominated in a particular setting, there is no reason to choose it, making their analysis unnecessary. In addition, figures typically omit the integer-program opt and its linear relaxation because both were significantly slower than other algorithms, making it difficult to visually compare the other algorithms.

Figure 1 shows the average cost and cpu time across all parameters. This gives a general idea of which algorithms are fast and preferment across the board, without specifying any type of data set

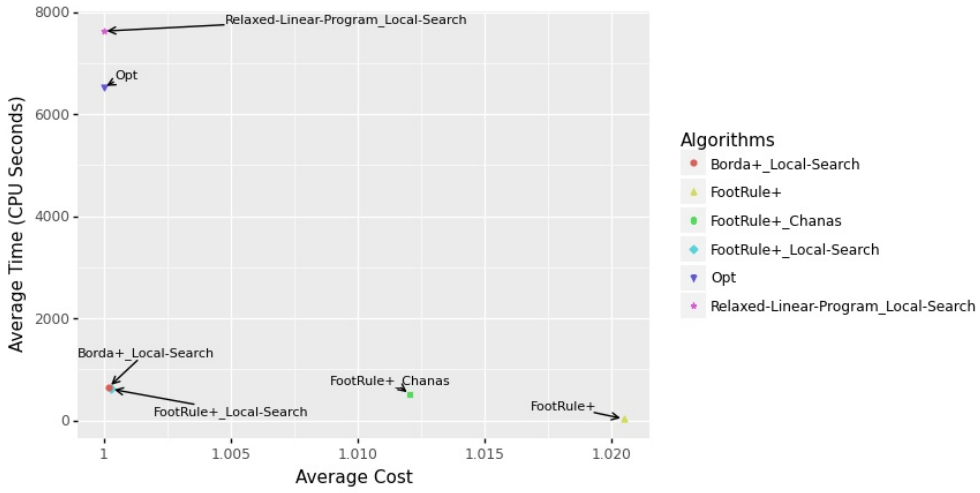

Figure 1: Average Costs and Times for Algorithms when Averaged Across All Combinatorial Synthetic Experiments

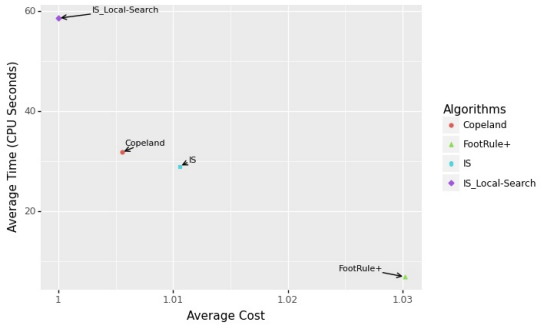

Figure 2: Average Cost and Time for Combinatorial Synthetic Data Sets where $n = 10$

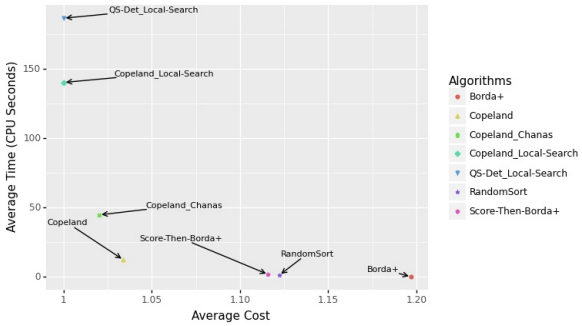

Figure 3: Average Cost and Time for Combinatorial Synthetic Data Sets where $n = 50$

in particular. Note how close to optimal Borda+,LocalSearch and FootRule+,LocalSearch come to Opt (Optimal LIP) while being about seven times faster on average. Footrule+ is the faster non-Pareto dominated algorithm but does not improve as much when combined with Chanas to when combined with LocalSearch. And Borda+,Chanas is even Pareto dominated, which makes us conclude that LocalSearch reduces costs more than Chanas as a post-processing algorithm, though at the expense of run-time. These results may not hold for arbitrary environments, but these overall, average results are solid rules of thumb in cases where less information is known about the data being processed, however rare such situations may be.

Parameter Analysis The heart of our analysis lies in identifying algorithms that perform well over different kinds of data sets one could imagine being used. To do this, we provide trends for the average performance of Pareto Optimal algorithms over the parameters N, n, k, and θ, though, for the sake of space, we leave data for parameters besides n and θ to the appendix. Consider Figure 2,

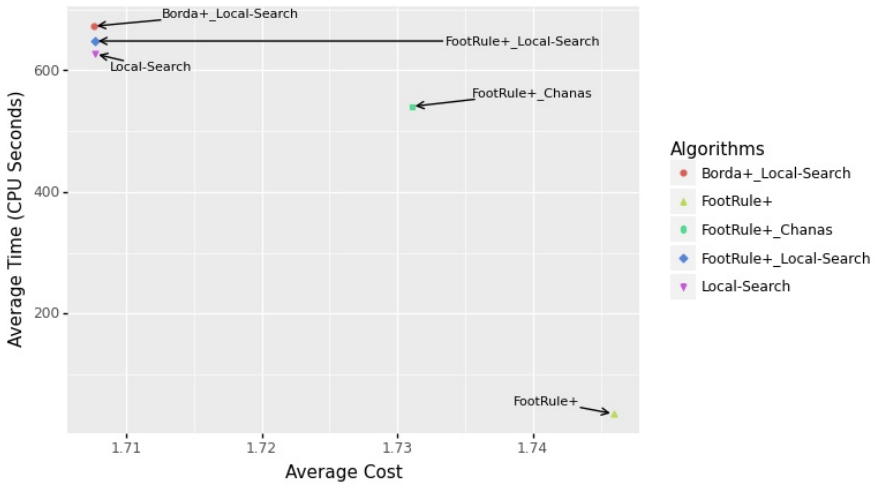

Figure 4: Average Cost and Time for Combinatorial Synthetic Data Sets where $\theta = .001$

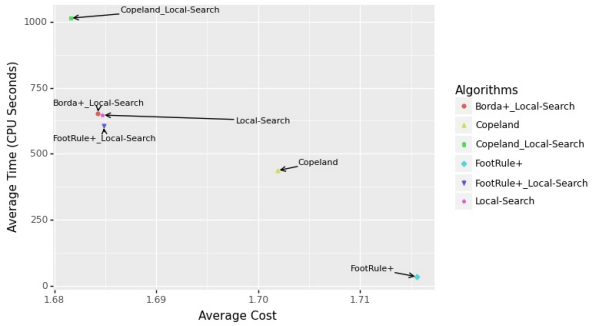

Figure 5: Average Cost and Time for Combinatorial Synthetic Data Sets where $\theta = .01$

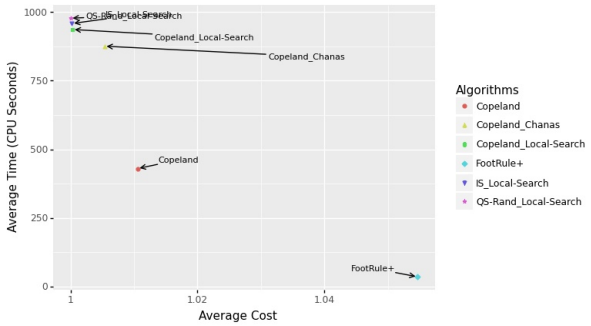

Figure 6: Average Cost and Time for Combinatorial Synthetic Data Sets where $\theta = .1$

and Figure 3. In the case where $n = 10$ the sorting algorithms Copeland and Insertion-Sort (IS) seem to provide for the best trade-off. As we scale the number of candidates and move on to $n = 50$, we observe the same trend when Copeland (and combinations with it) still perform generally well, however, other algorithms now emerge on the Pareto line.

We consider three values of θ to model varying levels of consensus. Recall that as θ grows, consensus grows, making small θ relatively harder cases. There are very few data-points, so drawing general trends may be misleading, but when $\theta = .001$, there is an approximately linear Pareto-boundary relating time and cost. All algorithms have relatively high costs, greater than seventy-percent from the optimal solution. So it appears that when voters have greater disagreement, scores and Borda ranks become less effective proxies for pair-wise disagreements across voters, perhaps owing to the fact that such values care how distantly ranked candidates are from each other, whereas Kendall Tau distance is only concerned that one candidate is ranked ahead of another. Indeed, this trend is borne out from considering the other values of theta: when $\theta = .01$, average costs are lower though still high, and when $\theta = .1$, all algorithms are within five-percent of the optimal solution.

The large improvement in costs observed when increasing from $\theta = .01$ to $\theta = .1$, as opposed to the improvements gained from changing from $\theta = .001$ to $\theta = .01$, is owing to the exponential relationship between consensus and θ.

As some additional observations, note that FootRule+ is consistently the fastest and least accurate (non-dominated) algorithm, whereas algorithms post-processed by Local-Search tend to be the slowest and most accurate. Finally, observe that for $\theta = .01$ and $\theta = .1$, Copeland provides a middling algorithm with respect to both time and cost, making it potentially useful when time and cost are weighed with approximately equal value. When $\theta = .1$, the results are dominated by algorithms not considered by [8], the exception being FootRule+. This may indicate that in easy, high-consensus situations, the guarantees provided by polynomial, constant-approximation algorithms become less important. Alternatively, these other algorithms may provide starting points for designing additional approximations for the top-list aggregation case, as some sorting-based methods provide such approximations when aggregating full ranks [1] [2].

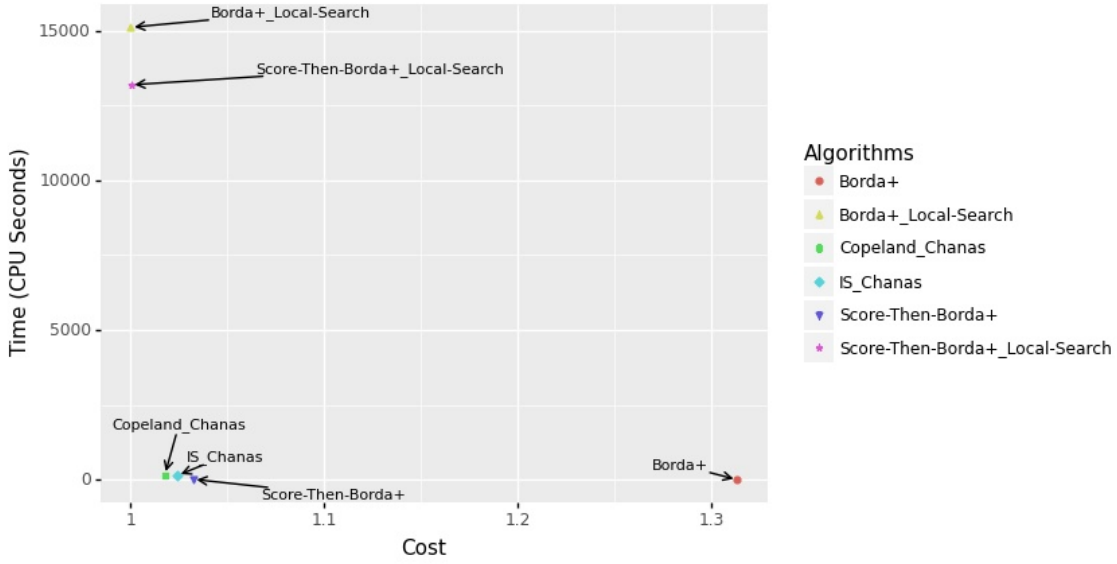

Figure 7: Costs and Times of Algorithms in when Evaluated On Skiers Data

We now consider the real-world skiers dataset. For context, recall that the skiers data-set had 170 candidates but a meager four voters, which is somewhat unusual. However, the number of voters is of relatively less importance than the out-sized quantity of candidates; this is because all algorithms considered have time-complexity linear in the number of voters. Generally, there may be a positive relationship between the number of voters and dis-consensus when voters are human beings of heterogeneous backgrounds, but one should expect relatively high consensus in situations such as this, where rankings are objective performance metrics of skiers. This mixture of a large number of candidates paired with a relatively high degree of consensus helps explain why Borda+ followed by Local-Search is able to find an optimal result faster than opt. With 170 candidates, the integer-program (and its linear relaxation) would have over 14,000 variables and an even greater quantity of constraints. We also observe an L-shaped Pareto-boundary, similar to [2]. Borda+,Local-Search and Score-Then-Borda+,Local-Search have better costs than other non-dominated algorithms, but at the expense of much greater run-times. For only slightly greater costs,

less than five-percent from optimal, one can find nearly instant results using Copeland,Chanas or IS,Chanas or Score-Then-Borda+. This also indicates that Local-Search produces more accurate results as a post-processing algorithm than does Chanas at the expense of additional run-time. One should expect such a result, given that Chanas must terminate after two iterations of candidate greedy re-ordering, whereas Local-Search will execute longer so long as more greedy improvement may exist.

Top-k Remark Unfortunately, our experiments show that Score-Then-Adjust, despite being efficient for top-k aggregation, were Pareto dominated by at least one algorithm for every single value of ϵ that we tested. Therefore, we chose to omit its visuals as they closely correspond to the combinatorial average for synthetic data in Figure 1.

6 Conclusion

Our results indicate that, just as in the full-ranking case, much potential exists for the application of post-processing algorithms such as Chanas and Local-Search to improve upon the results of existing algorithms. Like in the full rank aggregation case, there are not any proven results indicating that Local-Search, either on its own or paired with another algorithm, must provide a guaranteed constant-approximation. Thus, just as in the full-list case, this provides an avenue for conducting further investigation. We also confirm, perhaps unsurprisingly, that the degree of consensus amongst voters should affect the selection of algorithms, assuming such information is available. [2] proposed a number of heuristic algorithms for evaluating the level of consensus present in a given data-set - evaluating such heuristics for top-list aggregation would be valuable, especially in contexts where it would be preferred that algorithms be selected automatically based on parameters such as consensus level. Finally, we found that many algorithms not proposed in [8] provide useful heuristics, indicating such algorithms may be preferred for some settings and that some may provide as-yet-unproven approximation guarantees, which merits further investigation.

7 Acknowledgement

In this section we want to acknowledge help we received from Simon Mauras, co-author of [8], in understanding some specifics about his paper and getting insight into implementing some of his algorithms efficiently. For our real-world data set, we give credit to [9], the PrefLib library of preference data.[16] Finally, we want to also thank Fabien Collas and Ekhine Irurozki [5] for their implementation of the Mallows model on top-k lists.[17]

[16] https://www.preflib.org/

[17] https://github.com/ekhiru/top-k-mallows

8 Appendix

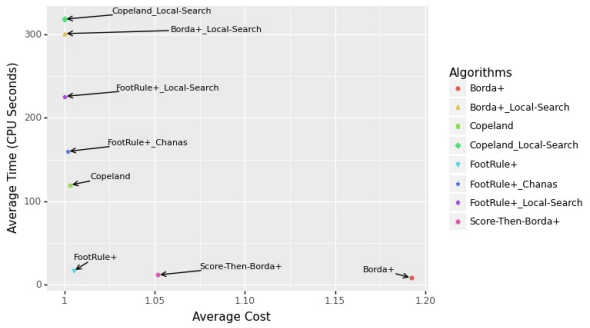

Figure 8: Average Cost and Time for Combinatorial Synthetic Data Sets where k-ratio= .1

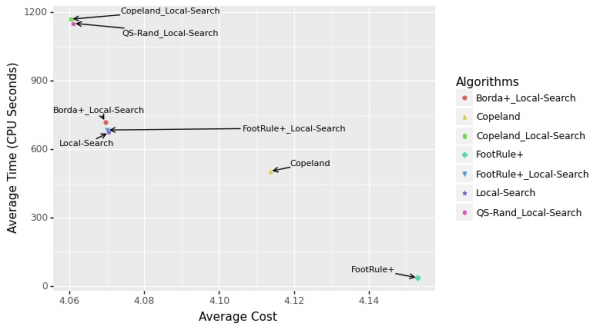

Figure 9: Average Cost and Time for Combinatorial Synthetic Data Sets where k-ratio= .5

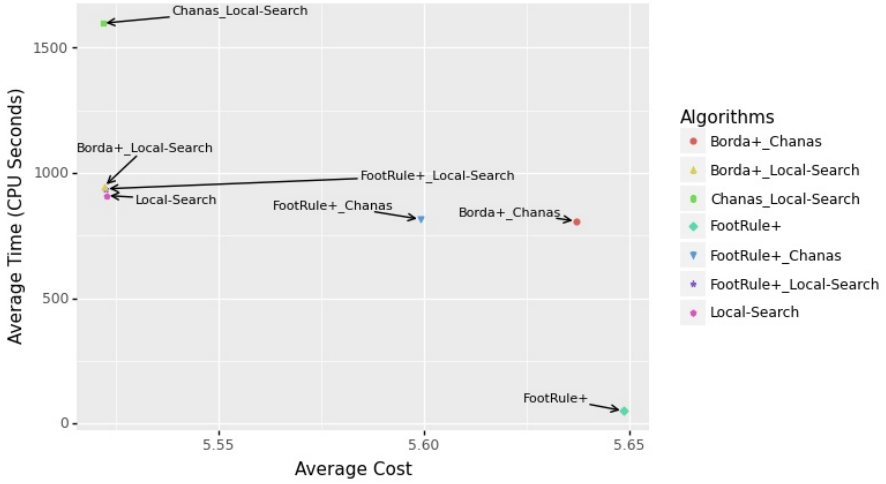

Figure 10: Average Cost and Time for Combinatorial Synthetic Data Sets where k-ratio= .9

References

[1] Nir Ailon. Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica*, 57(2):284–300, 2010.

[2] Alnur Ali and Marina Meilă. Experiments with kemeny ranking: What works when? *Mathematical Social Sciences*, 64(1):28–40, 2012.

[3] William W Cohen, Robert E Schapire, and Yoram Singer. Learning to order things. *Journal of artificial intelligence research*, 10:243–270, 1999.

[4] Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, pages 613–622, 2001.

[5] Collas Fabien and Irurozki Ekhine. Concentric mixtures of mallows models for top-k rankings: sampling and identifiability. *arXiv preprint arXiv:2010.14260*, 2020.

[6] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006.

[7] Gattaca Lv. An analysis of rank aggregation algorithms. *arXiv preprint arXiv:1402.5259*, 2014.

[8] Claire Mathieu and Simon Mauras. How to aggregate top-lists: Approximation algorithms via scores and average ranks. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2810–2822. SIAM, 2020.

[9] Nicholas Mattei and Toby Walsh. Preflib: A library of preference data HTTP://PREFLIB.ORG. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT 2013)*, Lecture Notes in Artificial Intelligence. Springer, 2013.