Experiments with Kemeny Ranking: What Works When?

Alnur Ali[a], Marina Meila[b]

[a] *1 Microsoft Way, Microsoft, Redmond, WA, 98052, U.S.A.*
[b] *University of Washington, Department of Statistics, Seattle, WA, 98195, U.S.A.*

Abstract

This paper performs a comparison of several methods for Kemeny rank aggregation (104 algorithms and combinations thereof in total) originating in social choice theory, machine learning, and theoretical computer science, with the goal of establishing the best trade-offs between search time and performance. We find that, for this theoretically NP-hard task, in practice the problems span three regimes: strong consensus, weak consensus, and no consensus. We make specific recommendations for each, and propose a computationally fast test to distinguish between the regimes.

In spite of the great variety of algorithms, there are few classes that are consistently Pareto optimal. In the most interesting regime, the integer program exact formulation, local search algorithms and the approximate version of a theoretically exact branch and bound algorithm arise as strong contenders.

Keywords: Kemeny ranking, consensus ranking, branch and bound, sorting, experimental evaluation

1. Introduction

Preference aggregation has been extensively studied by economists under social choice theory. Arrow discussed certain desirable properties that a ranking rule must have and showed that no rule can simultaneously satisfy them all [2]. Thus, a variety of models of preference aggregation have been

Email addresses: `alnurali@microsoft.com` (Alnur Ali), `mmp@stat.washington.edu` (Marina Meila)

proposed, each of which satisfy subsets of properties deemed desirable. In theoretical computer science, too, many applications of preference aggregation exist, including merging the results of various search engines [8, 14], collaborative filtering [26], and multiagent planning [16].

The *Kemeny ranking rule* [18] is widely used in both of these areas. From Arrow's Axioms' point of view, the Kemeny ranking is the unique rule satisfying the independence of irrelevant alternatives and the reinforcement axiom [29]. More recently, it has been found to have a natural interpretation as the maximum likelihood ranking under a very simple noise model proposed by Condorcet [30]. The same noise model was proposed independently in statistics by [21] and refined by [17], under the name *Mallows' model*. Further extensions that go beyond the scope of this paper but that are relevant to modeling preferences have been proposed by [23, 25].

Finding the Kemeny ranking is unfortunately a computational challenge, since the problem is NP-hard even for only four votes [3, 8]. Since the problem is important across a variety of fields, many researchers across these fields have converged on finding good, practical algorithms for its solution. There are formulations of the problem that lead to exact algorithms, of course without polynomial running time guarantees, and we present two of these in Section 3.1. There are also a large number of heuristic and approximate algorithms, and we enumerate several classes of these algorithms in Section 3.2.

Surveying the various approaches is only the first step, since we are interested in finding which algorithms (or combinations of algorithms) are likely to perform well in practice. For this, we perform a systematic comparison, running all algorithms on a batch of real-world and synthetic Kemeny ranking tasks.

Since we are attacking an intractable task, what we can examine is the trade-off between computational effort and quality of the obtained solution. We will examine which algorithms, if any, are systematically achieving optimal trade-offs. We hope thereby to be able to recommend reliable methods to practitioners.

Such an analysis was recently undertaken by Schalekampf and van Zuylen [28], forthwith abbreviated as SvZ, with whose work several parallels exist. While from the point of view of the experimental setup our work has little to add, we differ fundamentally in the conclusions we draw. That is because we factor in the difficulty of the actual problem, something that SvZ do not consider. Since both the performance and running time of an algorithm can change with the difficulty of the problem, we expect that the "best" algorithm to use will differ depending on the operating regime.

Hence, we propose several natural measures of difficulty for the Kemeny ranking problem, and re-examine the algorithms' performance results from this perspective.

The rest of the paper is structured as follows. The next section formally defines the Kemeny ranking problem and introduces the relevant notation. Sections 3.1 and 3.2 review the algorithms, while Section 4 presents the datasets we used for experimental evaluation and how they were obtained. The experimental results follow in Section 5; this section already gives some insights into what algorithms are promising, and into the role of the problem difficulty in shaping the performance landscape. We further examine these findings in Section 6. Section 7 discusses the different regimes of difficulty of the Kemeny ranking problem and proposes a simple heuristic to distinguish between them in practice. The paper concludes with Section 8.

2. The Kemeny ranking problem

Given a profile of N rankings[1] $\pi_1, \pi_2, \ldots \pi_N$ over n alternatives, the *Kemeny ranking problem* is the problem of finding the ranking

$$\pi_0^* \;=\; \operatorname{argmin}_{\pi_0} \frac{1}{N} \sum_{i=1}^{N} d(\pi_i, \pi_0), \tag{1}$$

In the above, $d(\pi, \pi')$ represents the the *Kendall distance* between two permutations π, π' defined as the minimum number of adjacent transpositions needed to turn π into π'. The ranking π_0^* is the ranking that minimizes the total number of disagreements with the rankings contained in the profile, and is called the *Kemeny ranking* of this profile.

Our interest is in the algorithms proposed to solve the NP-hard minimization in Equation 1. But, before we review the algorithms, it will be useful to introduce an alternative representation of the profile $\pi_1, \pi_2, \ldots \pi_N$.

This is the *precedence matrix* $Q = [Q_{ab}]_{a,b=1:n}$ defined by

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

[1]We do not distinguish in this paper between rankings, permutations, and orderings.

where $I(\cdot)$ is the indicator function and $\prec_\pi$ means "precedes in the ranking π." That is, Q_{ab} represents the fraction of times item a is ranked higher than item b across all the input rankings. The diagonal of Q is 0 by definition. Note also that $Q_{ab} + Q_{ba} = 1$, a form of antisymmetry.

Since Q has $n(n-1)$ entries and the profile has $N \times n$ entries (here we have not counted the entries that can be inferred based on other entries), for the situation where there are more votes N than alternatives n, the matrix Q represents a lossy compression of the profile, in the sense that in general the profile cannot be reconstructed uniquely from Q. But the real importance of Q lies in the fact, observed many times independently [8, 10, 24], that the precedence matrix is sufficient for determining the Kemeny ranking of the profile.

Thus, it will not be surprising that many of the algorithms below can be described in terms of the precedence matrix.

3. Algorithms

We aimed for an ensemble of algorithms that would allow for various trade-offs between performance and accuracy. We first discuss the algorithms we experimented with that return an exact solution to the Kemeny ranking problem. Then, we discuss algorithms that return an approximate solution.

3.1. Exact algorithms

3.1.1. Integer linear program

We solve the following integer linear program (`ILP`) which returns an exact solution to the Kemeny ranking problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{a,b} Q_{ab} x_{ba} + Q_{ba} x_{ab} && (2) \\ \text{subject to} \quad & x_{ab} \in \{0, 1\} && (3) \\ & x_{ab} + x_{ba} = 1, \ \forall a, b && (4) \\ & x_{ab} + x_{bc} + x_{ca} \geq 1, \ \forall a, b, c && (5) \end{aligned}$$

The first constraint means that the x_{ab} are binary variables, set to 1 when $a \prec_{\pi_0} b$. The second constraint captures the fact that either $a \prec b$ or $b \prec a$ in the Kemeny ranking. The third constraint enforces the transitivity $a \prec b \wedge b \prec c \Rightarrow a \prec c$.

The above ILP formulation has been given before [10, 28]. This formulation can also be interpreted as solving the minimum weighted feedback arc set problem from computer science [10, 19].

Because this algorithm will terminate with an exact solution given enough time, we use it as a baseline for accuracy.

3.1.2. Branch and Bound algorithm

In [24] it is shown that another way to solve Equation 1 exactly is by a branch and bound (B&B) algorithm. Each node in the search tree corresponds to a prefix $\sigma = [a_1, a_2, \dots a_j]$ of π_0, so that level j in the tree contains all possible prefixes of length j; branching is on the item to be added in rank $j+1$ which is an element of the set $\bar{\sigma} = \{1:n\} \setminus \{a_1, a_2, \dots a_j\}$. The cost and cost-to-go at a node are computed from the submatrix $Q_{\bar{\sigma},\bar{\sigma}}$. If one uses admissible heuristics then the B&B algorithm is guaranteed to optimize exactly; however the search tree has $n!$ paths and in the worst case the search is intractable. On the other hand, it can be shown theoretically and empirically [24] that in favorable cases, which are equivalent with strong agreement between the rankings $\pi_1, \dots \pi_N$, the B&B algorithm will expand only a limited number of nodes (of order n^2). Moreover, as with any B&B algorithm, limiting the available memory (a technique called *beam search*) leads to a family of approximate algorithms in which memory/runtime can be traded off for accuracy. In our experiments we will examine this trade-off as well.

The admissible heuristic used by [24] is a restatement of one of the bounds proposed in [10] in a form that makes it efficiently implementable. The authors of [24] observe (see also [22]) that having a good heuristic has a strong impact on the algorithm's time and storage.

3.2. Approximate algorithms

We divide the approximate algorithms into various classes and discuss each class in turn.

3.2.1. Sort based approximate algorithms

As discussed in SvZ and [1], several classic sorting algorithms from the computer science literature can be adapted to use the precedence matrix; Condorcet's paradox allows different sorting algorithms to produce different rankings. This is done by using the predicate $Q_{ab} > Q_{ba}$ instead of the standard comparison operator. That is, if alternative a is preferred to b by a majority of the profile, then the "$a \prec b$" is considered true [28]. The advantage of doing so is that these algorithms an approximate solution that can be fast and easy to implement. Here we adapt 3 classic sorting algorithms for this purpose.

InsertionSort `InsertionSort (IS)` starts from the front of an unordered list of the items that are to be ranked, and places them one-by-one into a sorted list that maintains an ordering over its items. Items are placed from the unordered list into the ordered list according to the following rule: if $Q_{ab} > Q_{ba}$ (that is, if item a is preferred to item b by a strict majority of the profile) then place a ahead of b in the sorted list [28].

MergeSort `MergeSort` recursively divides an unsorted list into two sublists. The sublists are then merged, according to the comparison rule described above, resulting in a fully-sorted list. There are several possible ways to divide lists into sublists. `MergeSort` places the items to the right of the median index into one list, and the rest of the items into another [28]. MS places even-indexed items in one list, and odd-indexed items in another [28].

QuickSort `QuickSort` recursively divides an unsorted list into two lists—one list comprising items that occur before a chosen index (called the *pivot*), and another comprising items that occur after—and then sorts each of the two lists. There are several ways to choose the pivot. `QuickSort` chooses the pivot uniformly at random [1]. `DetQuickSort (DetQS)` evaluates each item as a pivot and finally selects the item that has the fewest pairwise preference disagreements with the items that come before and after it according to Q given the current state of the list [31]. `LogQuickSort (LogQS)` proceeds just as `DetQuickSort` does, but only considers $\log n$ items (chosen at random) as the pivot, reducing its runtime [28]. `QS` chooses the item with the median index as the pivot [28]. [1] and SvZ, respectively, show that `QuickSort` and `DetQS` are 2-approximation algorithms for the Kemeny ranking problem.

3.2.2. Graph based approximate algorithms

There are several algorithms across the computer science, artificial intelligence, and machine learning communities that cast the Kemeny ranking problem as solving a problem on a graph [28, 13, 14, 8]. One can then resort to familiar methods for solving these graph based problems.

MC4 [14] introduce several Markov chain based methods for approximately solving the rank aggregation problem, of which `MC4` performs the best across their experiments, so we focus on it here. `MC4` constructs a Markov chain over items, where each entry $P_{ab},\ a \neq b$, of the row stochastic transition matrix P is set to

$$(1-\alpha)\frac{I(Q_{ab} > Q_{ba})}{n} + \frac{\alpha}{n}. \tag{6}$$

where α is a *teleport parameter* used to make sure the Markov chain is ergodic, and is often set to 1/7 in PageRank implementations. The indicator function in Equation 6 above returns 1 if item a is strictly preferred by the profile to item b and 0 otherwise. Diagonal entries P_{aa} are set to $1-\sum_{b\neq a} P_{ab}$. `MC4` then solves for the stationary distribution (by, e.g., the power method), and lastly ranks items in order of descending probability.

MC4Approx `MC4Approx (MC4a)` is intended to be a faster, approximate version of `MC4` [28]. Instead of solving for the stationary distribution, it computes vP^n, where v is a randomly initialized probability vector.

Footrule aggregation `Footrule` returns the ranking that minimizes the mean *Spearman footrule distance.* That is it minimizes

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{a=1}^{n} |\pi_i(a) - \pi_0(a)| \tag{7}$$

where $\pi(a)$ is the rank of item a in π. Thus, the objective represents the sum of the deviations in rank, for all items, between the candidate solution and each ranking in the profile, averaged over the profile [14, 13]. The solution to this problem is a 2-approximation to the Kemeny ranking. This problem can be solved by solving a minimum cost bipartite matching problem between items and their positions where the cost between item a and position p is set to $\sum_i |\pi_i(a) - p|$ [28, 13]. We use the Hungarian algorithm to solve this problem, which has a runtime of $\mathcal{O}(n^3)$, making `Footrule` one of the slowest algorithms that we experiment with.

CSS `CSS` is a greedy version of the `B&B` algorithm discussed previously introduced by [8].

3.2.3. Other approximate algorithms

These algorithms don't quite fit in any of the classes mentioned previously but have properties that make them interesting to investigate.

Pick-a-Perm `Pick-a-Perm (Pick)` randomly selects a ranking from the profile as the solution [28]. This algorithm is included here for completeness, as a default "pre-processing step" for other algorithms, and as a "strawman" algorithm. In this latter guise, comparing how other algorithms improve on the `Pick-a-Perm` gives a crude estimate of the potential gain of searching for a solution versus the lazy `Pick-a-Perm` solution.

Best-of-k `Best-of-k` returns the ranking in the profile with the smallest mean Kendall distance to the rest of the profile; it is a deterministic version of `Pick-a-Perm` [28].

Borda `Borda` sorts items in descending order according to their average position across all the input rankings [6]. This is equivalent to computing the columns sums of Q i.e. $q_a = \sum_b Q_{ab}$, then returning the permutation that sorts q_a in ascending order. In [17] it is shown that when the data are generated from the *Mallows' model* (which we define in the next section), this algorithm is asymptotically optimal.

Copeland's method `Copeland` sorts items in descending order according to the number of pairwise contests each item has won [11, 28]. In other words, it computes $q_a = \sum_b I(Q_{ab} > Q_{ba})$, and then returns the permutation that sorts q_a in descending order.

Chanas `Chanas` [7] iteratively swaps item positions as long as doing so reduces the number of pairwise disagreements with the profile. When no more moves can be made, the candidate solution is reversed, and the swapping phase begins once more, before returning the final solution. This algorithm demonstrated good empirical results in [28, 9].

Linear programming relaxation We also experiment with solving a linear programming relaxation (denoted `LP` or `LPRelaxation`) to the `ILP` mentioned previously in Equation 2.

3.2.4. Combinations of algorithms

Many of the algorithms mentioned previously start with an unordered list. Similar to SvZ, we experiment with combining these algorithms by initializing one algorithm with the output of another. This has the possibility of improving convergence to the optimal solution.

We experimented with all combinations of the algorithms mentioned above (denoted in subsequent sections with a "+" separating the combined algorithms).

3.2.5. Local search

We also experimented with running a local search procedure (denoted `LS` or `LocalSearch`) after all of the singleton and combination algorithms discussed previously—including following the family of exact and approximate `B&B` algorithms, which has not been attempted previously.

This procedure works by iteratively moving each item in the input list 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. Variations on this local search idea have been shown to give good theoretical approximation bounds as well as empirical results [28, 9, 19].

The complete list of algorithms we evaluated (comprising more than 104 different algorithms and combinations thereof) is given in the supplementary material[2].

3.3. Implementation remarks

All the algorithms were run on an 8 core, 1.86 Ghz machine with 32 GB of memory.

All the algorithms except for a few were implemented in Java. The `ILP` and `LP` were implemented in MATLAB calling into CPLEX. We note that the core `ILP` solver is written in highly optimized and parallelized code, biasing results towards the `ILP`. The family of `B&B` algorithms were implemented in Java by Bhushan Mandhani, as described in [22]. `MC4` and `MC4a` were implemented in MATLAB, again slightly biasing results in their favor.

The reader is of course aware that the implementation of an algorithm will affect its running time and our experiments are no exception. We cannot

[2]`http://www.stat.washington.edu/mmp/mss2011_complete_experimental_evaluation.pdf`.

remove all implementation and programming language effects on the running time. However, we made efforts to make the comparison as fair as possible, and in particular to make sure that no algorithm is specifically favored or disfavored. We took care to randomize the tie breaking, so that the algorithm solution is not affected by implementation. In addition we took special precautions (omitted since they are outside the scope of this paper) to control for systemic effects as garbage collection, initialization of large structures not belonging to the algorithm, etc. The code will be made public, and we believe the running times presented here accurately reflect what our implementation would do if another researcher used it.

4. Datasets

In this section, we discuss the datasets that we used for our experiments. In collecting these datasets, we aimed for a variety of lengths (n), number of rankings (N), and degrees of consensus. To this end, we used real-world and synthetic datasets. We first describe the real-world and then move on to the synthetic datasets.

4.1. Real world datasets

We procured several real world data sets, aiming for problems where a Kemeny ranking would be meaningful, but challenging to find.

4.1.1. Websearch dataset

In order to generate this dataset, we issued a set of 37 queries to $N = 4$ commercial search engines and captured the top 100 results returned. If a particular engine did not have the same results as the others, then we appended the set of results belonging to the other engines to the end of the top 100 results from the given engine (in random order). This procedure as well as the queries used replicate those of SvZ. This will facilitate a comparison between the results. The queries used and the total number of search results (i.e. alternatives in the Kemeny ranking) for each query are given in Table 1.

4.1.2. Skiers dataset

We obtained another dataset from the web site `www.ski-db.com` which records the results of World Cup (WC) and Olympic ski races from 1967 to the present. For our experiment, we used the rankings in the 8 WC

query	n
affirmative action	316
alcoholism	316
amusement parks	315
architecture	314
bicycling	318
blues	300
cheese	309
citrus groves	335
classical guitar	310
computer vision	296
cruises	290
Death Valley	322
field hockey	338
gardening	314
graphic design	329
Gulf war	318
HIV	282
java	313
Lipari	324
lyme disease	289
mutual funds	306
National parks	333
parallel architecture	341
Penelope Fitzgerald	329
recycling cans	336
rock climbing	348
San Francisco	319
Shakespeare	291
stamp collecting	300
sushi	326
table tennis	303
telecommuting	315
Thailand tourism	321
vintage cars	341
volcano	275
zen buddhism	298
Zener	320

Table 1: The 37 queries and their respective number of results, n, returned by $N = 4$ search engines used to curate the `websearch` dataset. $\overline{n} = 314.86$ and $\sigma_n = 17.35$ for the entire dataset.

races in Giant Slalom for women during the 2008-2009 season. Each race comprised two runs, so our dataset contains 16 rankings. A total of 59 skiers participated. However, not every skier ran in every race, and some were disqualified after the first run. We ranked all the non-participants after the participants, and broke ties alphabetically by last name. Consequently, we obtained a data set with $n = 59$ items and $N = 16$ rankings.

4.2. Synthetic datasets

We generated the following datasets in order to more thoroughly investigate algorithm performance on datasets of varying length, number of rankings, and consensus.

4.2.1. Mallows' model datasets

We drew samples from a *Mallows' model* (MM) with various parameter settings. The Mallows' model is an exponential model over rankings introduced by [21] and is given by:

$$P_{\pi_0,\theta}(\pi) \;=\; \frac{\exp\left(-\theta d(\pi,\pi_0)\right)}{Z} \qquad Z = \prod_{i=1}^{n-1} \frac{1-\exp\left((-n-i+1)\theta_i\right)}{1-\exp\left(-\theta_i\right)} \tag{8}$$

In the above, π_0 is the *central ranking* of the model and $\theta \geq 0$.

The parameter θ of the Mallows' model quantifies the concentration of the distribution around its peak π_0. For $\theta = 0$ the distribution is uniform (no peak, and no consensus), and for larger θ the distribution becomes increasingly peaked. Therefore, we expect that the larger the θ value, the stronger the consensus in the data, and therefore the easier the Kemeny ranking problem. We shall see that our experiments support this intuition. Also, from the point of view of an algorithm, increasing the number of items n will make the problem harder. The values for n, θ, N that we used are listed in Table 2. In selecting these parameters, we aimed to have a range of difficulties, with more emphasis on the challenging and hard problems.

4.2.2. Plackett-Luce model datasets

We also drew samples from a *Plackett-Luce (P-L) model.* The P-L model is a probability distribution over rankings introduced independently by [27] and [20] and is given by:

N	n	θ **(MM)**	$s_{\pi^{-1}(i)}$ **(P-L)**
100	10	0.001	10, 9, ..., 2, 1
100	50	0.001	50, 49, ..., 2, 1
100	10	0.01	-
100	50	0.01	-
100	10	0.1	-
100	50	0.1	-
5000	10	0.1	-
5000	50	0.1	-

Table 2: The parameters used to generate the Mallows' model and Plackett-Luce datasets.

$$P_s(\pi) = \prod_{i=1}^{n} \frac{s_{\pi^{-1}(i)}}{Z_i} \qquad Z_i = \sum_{i \le j}^{n} s_{\pi^{-1}(j)} \tag{9}$$

where $s_i > 0$ are parameters, and $\pi^{-1}(i)$ returns the alternative in rank i of the permutations.

Each alternative is assigned one s parameter, which can be considered as its importance. It is easy to see that the mode of the P-L distribution is represented by the ranking that sorts the alternatives decreasingly w.r.t. s_i. Beyond this, the s_i parameters of the P-L model are not as clearly interpretable as those of the Mallows' model, in the sense that it is harder to derive other more complex properties of the distribution by inspecting them.

However, from Equation 9 one sees that the consensus will be stronger when the s_i values decrease faster. The values we chose for our experiments decrease slowly, making our test cases relatively hard[3].

As with the Mallows' model, the problem becomes more difficult for larger n. Table 2 lists the parameters we used in our experiments.

We chose the P-L distribution, because it is a distribution peaked around the mode, but has a different shape and rate of decay than the Mallows' model. While the Kemeny ranking cost given by the r.h.s. of Equation 1 represents the log-likelihood for the Mallows' model, no analogous relation exists between the Kemeny cost and the log-likelihood of the P-L model.

[3]We ran pilot experiments with other values before settling on these.

4.2.3. "Random" dataset

Lastly, we generated a synthetic dataset consisting of $N = 100$ and $n = 100$ random permutations.

5. Experiments

5.1. Methodology

We ran each algorithm described above on each data set (i.e. 104 algorithms $\times$ 49 data sets) and recorded the running time, resulting ranking, and other variables.

As a measure of algorithm accuracy we use the number of pairwise disagreements between the algorithm's output ranking and all the data, i.e.

$$\text{cost}(\pi_0) \;=\; \sum_{i=1}^{N} d(\pi_i, \pi_0) \tag{10}$$

The above is nothing else than the rescaled r.h.s. of Equation 1, so it reflects how well the algorithm has optimized the desired objective. We call this objective the *cost* of π_0 and by extension the cost of the algorithm that produced π_0.

`B&B` is controlled by a *beam size* parameter, which limits the memory allocated by the algorithm. When this limit is attained, then `B&B` reverts to beam search. We run the algorithm with a range of beam sizes from a default 2000 down to 1. Thus, in our experiments, `B&B` is typically an approximate algorithm.

For each data set, we plot the runtime versus cost for all algorithms. Then we select the algorithms which are not dominated both in accuracy and in runtime by another algorithm. These form the *Pareto boundary*, displayed on all figures. For clarity, only selected algorithms have their names listed, the other algorithms are not named. Among the selected ones, we list `B&B`, `ILP` and selected algorithms on or near the Pareto boundary.

Another possible measure of accuracy would have been to subtract the optimal cost, associated to the true Kemeny ranking, from each algorithm's cost, showing only the regret incurred by a given algorithm w.r.t. to the ideal ranking. We do not do this when we first present the results, because we found it informative to consider the size of the regret in the context of the optimal cost value. Later, it will become apparent that for difficult Kemeny ranking problems, the regret incurred by the better algorithms is very small compared to the cost. To enhance this perspective, we present analyses where the regret is expressed as a fraction of the optimal cost.

Another possibility to measure algorithm accuracy is the inversion distance $d(\pi_0, \pi_0^*)$ between the algorithm's output π_0 and the true Kemeny ranking π_0^*. We have not pursued this route because, in many cases and especially for the `websearch` data, the Kemeny ranking is not unique, so this measure would have made little sense. By the triangle inequality, we know that the inversion distance to any Kemeny optimal permutation must be greater or equal to the regret divided by the number of samples N.

5.2. Mallows' model datasets

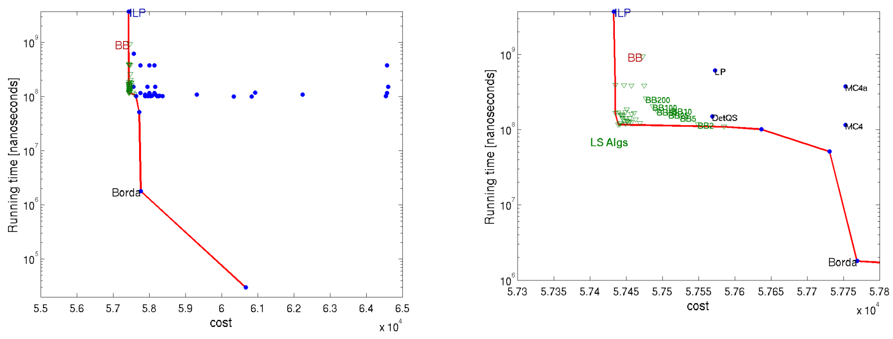


Figure 1: Mallows' model dataset results with $N = 100$, $n = 50$, and $\theta = 0.001$: all algorithms (left) and detail (right). `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles.

We present the experimental results starting with the most difficult cases and proceeding towards the easiest.

The case of $N = 100$, $n = 50$, and $\theta = 0.001$ (indicating no consensus) is plotted in Figure 1. The left panel shows the overall picture, while the right panel focuses on the left side of the plot, where the more competitive algorithms are located.

We discuss both graphs in some detail, since the other data sets produce qualitatively similar results.

Starting with the left plot, at one end of the Pareto curve is `Pick-a-Perm`, included as a "strawman" algorithm, while at the other end is the exact integer program `ILP`, which takes about 20 seconds to run (recall that the `ILP` is parallelized so the real run time should be multiplied by 8). `Borda` is the fastest algorithm that improves the cost considerably; all algorithms that outperform `Borda` in cost are more than 10 times slower. `Copeland` and `CSS` follow on the Pareto curve, after which the `LS` algorithms and the `B&B` algorithms with limited beam size cluster on and near the Pareto curve. One notices that a large number of algorithms have relatively large running time (comparable with `LS` and `B&B`) without the performance. This observation is replicated in all the experiments we performed. Therefore, in the right panel and in the subsequent experiments we focus on the subset of algorithms that are near the optimal cost.

In this area, one notices that the `LS` algorithms are near a knee in the Pareto curve, attaining the optimum or near optimum 100 times faster than `ILP`. In close pursuit but suboptimal are the `B&B` algorithms (all with limited memory) with comparable run time, but with costs approximately 0.2% away from optimum. This group of algorithms is about 100 times slower than Borda. Next come `CSS` and `Copeland` which are 2–5 times faster, but with worse cost. Also near the Pareto boundary are `DetQS` and `MC4`.

The case of $N = 100$, $n = 50$, and $\theta = 0.01$ is plotted in Figure 2. The results are similar to those in Figure 1, with `Borda`, `Copeland`, `B&B`1, `B&B`2, `LP`, `ILP` and some `LS` algorithms on the Pareto curve, and `DetQS, MC4` and the remaining `B&B` and `LS` algorithms near it.

This example shows that practically there is little difference between this data set and the one with $\theta = 0.001$. While in the limit of large N the case $\theta = 0.01$ would exhibit more consensus, and minimizing the cost in Q would become tractable, at this sample size ($N = 100$) both have almost no consensus and their distributions are indistinguishable.

The case of $N = 100$, $n = 10$, $\theta = 0.001$ is not plotted. All `LS`, all `B&B` (with the exception of `B&B`1), and the majority of the `QuickSort` based algorithms reach the optimum. The fastest to reach the optimum is `Copeland` (10 times faster than the rest). 22 algorithms do not attain the optimum, `Borda` being the first one on the Pareto curve. Among the others are the greedy `CSS`, `Foot` and `Best-of-k`, `Chanas`, and 14 sort based algorithms, including `IS`, `MS`, and `QuickSort`. `B&B` is exact from beam size 2000 up[4]. Note also that the "true model," the central permutation which generated the data, is not optimal. This is the case in all our artificial examples; it was

[4]The beam size is in units corresponding almost exactly to 2 nodes: for instance, `B&B`2000 will maintain a queue of at most 4000 nodes.

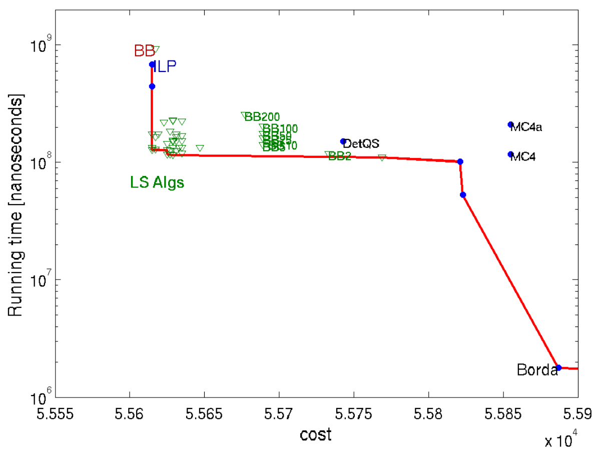


Figure 2: Mallows' model dataset results with $N = 100$, $n = 50$, and $\theta = 0.01$. Some algorithms are not shown because they were far from the Pareto boundary. `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles.

an experimental choice in order to prevent the particular shape of the data distribution, in this case the Mallows' model, to influence our results.

The case of $N = 100$, $n = 10$, and $\theta = 0.01$ (indicating weak consensus) is very similar to the above. All `LS` algorithms and all `B&B` algorithms except for `B&B1` and `B&B2` reach the optimal cost; `B&B` is exact from beam size 1000 upward. `MS, Best-of-k+IS` also reach the optimum. Barely faster than these, but suboptimal are `DetQS, B&B2,B&B1, CSS, Borda+MS, Copeland+QuickSort, MC4a+QuickSort, MC4a+MS, Copeland+MS, Pick+IS`. Finally, much faster and slightly worse in cost are `Copeland` and `Borda`.

5.3. P-L datasets

The case of $N = 100$, $n = 50$ is plotted in Figure 4. The results were mostly similar to Figure 1. The `B&B` and `LS` algorithms are clustered together on or near the upper part of the Pareto boundary, and its highest knee; the range of costs in this cluster is $< 0.03\%$. `LogQS` is at the inverted knee, and `Copeland` and `Borda` (not shown) are suboptimal and much faster. The range of costs between `Borda` and optimum is of 0.3% of the optimal cost.

In the case of $N = 100$, $n = 10$ (no figure), most algorithms reached the optimal cost, the fastest one being `Copeland`. `B&B` algorithms were exact

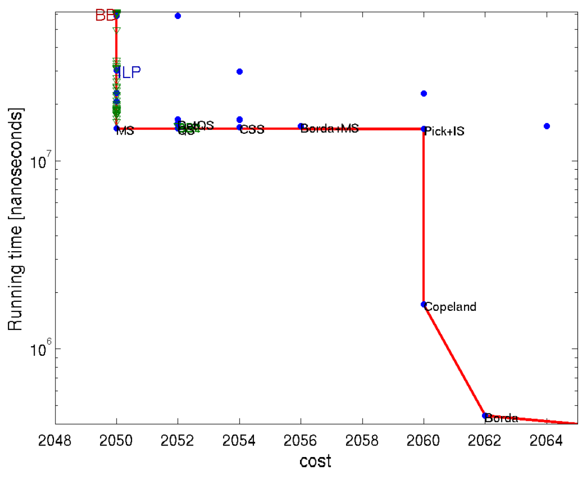


Figure 3: Mallows' model dataset results with $N = 1000$, $n = 10$, and $\theta = 0.01$. Some algorithms are not shown because they were far from the Pareto boundary. `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles.

from beam size 1000 upward.

5.4. Random dataset

The results on the `random` data set (Figure 5) are visually similar to the previous plots of artificial data with weak or no consensus.

5.5. Skiers dataset

This dataset distribution was between a peaked and a uniform distribution with consensus near the top but lack thereof in the other ranks. The results, in Figure 6, are strikingly similar to those on the artificial data with weak consensus. Therefore, we do not plot the lower part of the Pareto curve, spanned by `Copeland` and `Borda`. The `LS` algorithms have an advantage of $1-1.6\%$ over the `B&B` algorithms; a group of `QuickSort` algorithms is another 1.5% further in cost.

5.6. Websearch datasets

These tasks are more challenging for an algorithm as the consensus is weak, especially in the lower ranks, and they feature long rankings ($\overline{n} =$

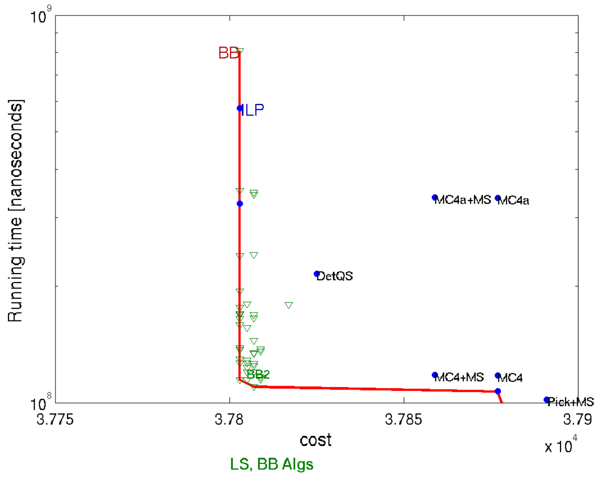


Figure 4: P-L dataset results with $N = 100$, $n = 50$. Detail of the upper part of the Pareto boundary. `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles. The algorithms on or near the undisplayed part of the Pareto curve are `LogQS, Copeland, Borda`, and `Pick-a-Perm`. Some algorithms are not shown because they were far from the Pareto boundary.

$314.86, \sigma_n = 17.35$). The results are plotted in Figure 7 and are an average over the results for each of the 37 individual queries comprising the dataset. The `LS` algorithms and the lower memory `B&B` algorithms are 2–3 orders of magnitude faster than `ILP` and `LP`. The `LS` algorithms are within 0.05% of the optimum, on average, while the `B&B` algorithms are approximately 0.06% higher in cost. `Copeland` remains the next fastest significant algorithm, but on these data its advantage in speed is only of a factor of 2 with respect to the more accurate low memory `B&B` algorithms. `Borda` stands out as much faster (10 times faster than `Copeland`) but at a significantly worse cost.

Because the `websearch` dataset is a real world dataset and also contains the longest permutations out of all the datasets we experiment with, we decided to further compare the group of `B&B` algorithms with the group of `LS` algorithms, with `ILP` as a reference point. For this, we normalized the costs by the optimal cost, and the running times by the running time of `ILP`. It appears (see Figure 8) that the optimal costs increase linearly with n while the run times increase approximately exponentially with n.

We then displayed the normalized running times and costs of all algo-

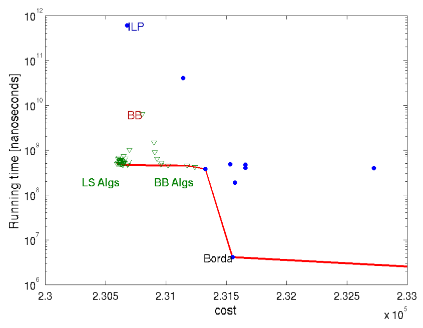


Figure 5: `random` dataset results ($N = 100$, $n = 100$). Some algorithms are not shown because they were far from the Pareto boundary. `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles. The algorithm on the “inverted knee” is `DetQS`.

rithms, but this time with the means and the standard deviations in both dimensions. These results are plotted in Figure 9. A number of algorithms had very large variations in cost; not surprisingly, these algorithms were the same algorithms that ran in time comparable with the `LS` algorithms, but did not achieve costs close to optimal, so we removed them from the plot.

The figure shows immediately that all `LS` algorithms always have very low cost, but high variance in running time. By contrast, the `B&B` algorithms have much higher variance in cost, attaining the optimal cost occasionally, with little variance in running time (which is dictated in the present implementation by the memory limit).

Thus, it seems that with a `B&B` the running time is predictable (either indirectly by the memory limit, or directly by setting a stopping time), but the cost may fluctuate away from the optimum. The opposite is true about the `LS` algorithms as a group. In addtion, the best cost and the best running times of `LS` algorithms outperform any of the `B&B` algorithms. Thus, is it worth running the `B&B` algorithms at all?

We investigated this question by first studying the `LS` algorithms separately. Figure 10 displays the normalized running times and costs of the `LS` algorithms only. The figure shows that the `LS` algorithms perform similarly, in the sense that the variations in both cost and run time depend much more on the data set than on the initialization method for the `LS`. Since for all `LS` algorithms the median cost is at the optimum, this means that a good rule of thumb is to run `LS` a small number of times with different initializations in order to have a high probability of getting the best possible result by LS[5].

The running time of B&B1000 (labeled B&B) and B&B200 are respectively about 15 times and 6 times the mean time of the LS, while the other B&B algorithms have shorter running times. Thus running a B&B algorithm once is comparable with running LS with multiple initializations. This also suggests the idea of running an approximate B&B followed by LS, which we discuss shortly.

Figure 11 revealed another surprising property of B&B: this figure plots the ranks of the B&B algorithms on the 37 data sets, where the data sets are sorted by increasing n. The figure shows a strong dichotomy in the ranks: each B&B algorithm is either optimal or worse than most LS algorithms[6] The effect of the beam size is negligible except for the extreme cases (B&B1000 is somewhat better, while B&B2 and B&B1 are worse).

A second striking dichotomy in this plot is that the data sets on which B&B algorithms are superior are without exception the queries with *large* n. Thus, as n grows larger, the B&B searches become more competitive.

These features also suggest that running B&B with a moderate beam size of 25–100, followed by a single LS could be a very promising strategy, especially for larger n. We tested this hypothesis on all our data sets by running B&Bwith beam size varying from 1 to 100000 followed by LS, and the results were in line with the other LS algorithms, for all the beam sizes. The interesting case is at lower beam sizes, where the added cost of initialization time is of the same order or smaller than that of LS. The performance of these algorithms was similar to the other LS algorithms. Figure 14 shows that the performance of B&B10+LS is identical with that of one of the faster LS algorithms, QuickSortLS. B&B10+LS's running time was faster than QuickSortLS's; moreover, B&B10+LS attained the best cost of any LS

[5]In more detail, we can reason as follows: there is a probability of approximately 1/2 of achieving the optimum in one run; therefore, therefore, every additional run halves the probability of not attaining the optimum. Hence, if we run LS, say, 5 times, and take the lowest cost, then the probability that that is not optimal is approximately $1/2^5 = 1/32$.

[6]When several algorithms reach the optimal cost, as for any other ties, we automatically rank a B&B algorithm before any other algorithm. This was not in order to "cheat" in favor of the B&B algorithms, but to better illustrate the phenomenon taking place. This is, we stress again, that while the ranking within the group of LS algorithm appear random and unstable, the ranking between any B&B algorithm and *all* the LS algorithms w.r.t. cost is practically always of the form: some B&B which attain the optimum, then all LS algorithms (some of which attain the optimum too), and finally the remaining B&B algorithms.

algorithms on all data sets but the `random` data set.

6. The Pareto curves and general observations

Across all experiments described here we observe the following. First, no algorithm reaches both the lowest cost and lowest runtime simultaneously over all experiments. This is why in our experimental results, we traced the Pareto boundary, i.e. the set of algorithms that cannot be bettered in both cost and running time by another algorithm.

The shape of the Pareto boundary tell us about the possible trade-offs between run time and accuracy. It is easy to observe that in practically all experiments the Pareto boundary has the same angular shape, with two "knees." One of these is represented by the `Borda` algorithm, which we recommend as the best trade-off if time is more important than accuracy. In our experiments, `Borda` is no more than 2.5% over the optimal cost, which agrees with the results of SvZ.

At the other knee, the boundary turns sharply towards high running times. In this regime, large increases in running time are needed for very small gains in cost. The `LS` algorithms cluster near the vertical part boundary cluster; they attain good, often optimal costs, but are relatively slow compared to `Borda`. Also in this area are the restricted beam size `B&B` algorithms, with comparable running times but slightly suboptimal costs. On the same boundary, than the `LS` and `B&B` groups of algorithms, sometimes by 1–2 orders of magnitude, are `LP` and `ILP`. In our experiments, `LP` found the optimal cost in all `websearch` data sets and in several of the other data sets.

In between these two extremes lie a set of other algorithms, among which the most frequent are `Copeland, DetQS, MC4, CSS, MC4a`—sometimes joined by `QuickSort` or `IS`. The Pareto curve is concave in this area, showing that with small losses in one criterion, one can gain much in the other criterion, by effectively "sliding" to one of the knees. The exception to this rule is `Copeland` which is always the fastest of them, and which always improves on `Borda`. Of the other algorithms, only `DetQS` is consistently near the boundary, but it is practically always overshadowed by the `B&B` algorithms with small memory. Therefore, the only algorithm we recommend as a reliable trade-off is `Copeland`.

The geometric shape of the boundary, and especially the very steep left side, is bad news, indicating that the availability of so many algorithms does not translate in a comparable number of trade-off points between computational effort and cost. The two knees, with their almost 90 degrees angles (no angle in the Pareto boundary can be less than or equal to 90 degrees), indicate that there are effectively only two good and robustly attainable trade-offs: the point at `Borda` and the small boundary region determined by the group of `LS`/`B&B` algorithms. A minor third point of interest is given by the `Copeland` algorithm.

If `Borda` or `Copeland` attain the optimum, then we are in luck to have an easy problem at hand. Otherwise, further gains in cost can be obtained only with very large (orders of magnitude) increases in run time.

Our conclusions agree with the previous experiments of SvZ, with respect to the `LS` algorithms, and to `Borda` and `Copeland`. In addition, we have shown that the `B&B` type of algorithms are also competitive, in the same high accuracy / high cost regime as `LS`, especially for large n.

An intriguing possibility is to accelerate the limited memory `B&B` algorithms by a more efficient implementation. This would have as an effect the filling up of the inverted knee region with a convex and almost continuous boundary controlled by a single parameter, the memory size of `B&B`. This is a real option, since our current implementation leaves room for improvement.

Besides underscoring the future potential of `B&B`, we want to draw attention to another aspect of practical importance in computing the Kemeny ranking. It is the fact that in evaluating Kemeny ranking methods, the line between theory and implementation detail is blurred. The care in implementation, the programming language and machine dependent optimization, can account for about an order of magnitude in running time and perhaps more in memory. Here we did not make special efforts for any particular algorithm, and were careful to remove as many of the implementation-dependent factors that we could. Never the less, the reader must be aware that in any empirical evaluation these factors will play a role.

7. Regimes of difficulty: strong, weak, and no consensus

In the tasks above, we presented the algorithms with difficulties ranging from hard to easy. The reader has probably noted that the success of the algorithms (and implicitly the difficulty of the problem) did not depend on n alone, but also on the data distribution. In particular, the concentration of the data around the Kemeny ranking had a strong influence on the algorithms' ability to optimize the cost, and for some algorithms (like `B&B`) it also influenced the running time. We now discuss specific lessons that depend on the difficulty of the data distribution distribution.

7.1. Strong consensus

There are easy distributions, like the Mallows' with $\theta = 0.1, n = 50$ or 10, and the P-L with $n = 10$, where the data are concentrated around a central permutation. In these cases, the Kemeny ranking is found by many algorithms, and for those algorithms whose running time varies the search is short. We refer to this situation as having *strong consensus*.

For strong consensus cases we recommend using `Borda` which emerges as a consistent winner. Many other algorithms, including `B&B` (which is exact), `LP`, the exact `ILP` and `LS` also solve these cases but are not worth the overhead.

7.2. No consensus

At the opposite end of the difficulty spectrum is the *no consensus* situation. In these cases, the data distribution is almost uniform, without a mode. This is typified by the `random` data set, but also by distributions like the Mallows' model data in Figure 1), which is almost indistinguishable from random data[7].

Another situation with no consensus is clustered data. That is, the data is the union of two or more groups, each of them having consensus, but around a different π_0. In this case, the cost may have local minima (corresponding to the Kemeny rankings of the groups) but not necessarily a minimum corresponding to a global consensus. An extreme example is the case where there are two equal groups, one with center π_0 and the other centered on the *reverse* of π_0. In this case, the Q matrix has all entries equal to 0.5, and every ranking is a Kemeny ranking.

In general, in this case the Kemeny ranking is too sensitive to small perturbations in the data (e.g. adding a single new ranking) to have any real meaning. In fact, especially for small data sets, like in the `websearch` data sets, there are multiple Kemeny rankings. Approximate solutions obtained from different algorithms can in fact be at large Kendall distances

[7]At a very large sample size, this Mallows' model and any non-uniform unimodal distribution will produce data for which `Borda` will find the true π_0 which will also be the Kemeny ranking. However, our sample size is $N = 100$, much too small for this effect, as shown next.

from each other. Algorithmically speaking, no consensus is an unstable situation. Intuitively, it is futile and misleading to give meaning to the computed (approximate) Kemeny rankings.

To better drive home this point, for the synthetic data, we tested the quality of the true π_0 that generated the data as a candidate for a Kemeny ranking. They are listed as `MallowsModel` and `PlackettLuce` in the supplementary material. For the difficult, no consensus distributions this cost is higher than that achieved by many other algorithms (e.g. by an $\approx 6\%$ increase over the algorithm with the lowest cost in the case of the MM model with $\theta = 0.001, N = 100, n = 50$). For the P-L model, the true permutation is not optimal even in the $n = 10$ case.

In such cases it is important to carefully examine the need for a Kemeny ranking and to detect and signal the lack of consensus.

If the situation is a decision task, for instance voting, then the decision maker (or decision algorithm) must always output a ranking. We recommend in this case[8] picking a fast algorithm (e.g. `Borda, Copeland`), as there is not a substantial difference in costs achieved by the best algorithm, nor can one attach more significance to a lower cost ranking.

In such a case, the important trade-offs will not be of a computational nature, but of a social nature (like having an simple, understandable rule) and are beyond the scope of our investigation. We do recommend that, whenever possible, the lack of consensus be detected. The `B&B` algorithm has a natural way of detecting the lack of consensus, in the guise of very large number of optimal or almost optimal partial solutions in the algorithm's queue. It is not necessary to run the algorithm to completion in order to diagnose lack of consensus—it is sufficient to monitor if the queue extends in breadth rather than in depth. If N is large enough, then the *bootstrap* [15], i.e. repeating the algorithm (e.g `Borda`) many times, with slight variations in the sample, will also indicate lack of consensus.

There are other situations, for instance when our aim is descriptive, and the Kemeny ranking is to be used as a summary of the voters' preferences. In this case, we recommend that one declares that no ranking can summarize the expressed preferences in an acceptable way, and changes the model. For

[8]If the exact solution cannot be found in the available time. When the exact solution can be found, and in particular a certificate of optimality, this has additional advantages from a social choice stanpoint.

instance, one could investigate the existence of consensus within groups, i.e. the existence of "voting blocks."

7.3. Weak consensus

The more interesting case from a computational point of view is the case of *weak consensus.* In such cases, the distribution has a mode, but this is not pronounced, or appears only in some of the ranks (typically the higher ranks). Here, the Pareto boundary presented in Section 6 is relevant: having a high performance algorithm can make a difference in cost.

In these cases the user can choose between speed for suboptimal cost (with `Borda, Copeland`) and more effort for almost optimal cost (with `LS`, `B&B` , and, whenever possible, `ILP`).

The important question is how to establish whether we are in a weak consensus regime or not, preferably before running one of the time-consuming algorithms? We investigate this question in the next section.

7.4. Heuristic measures of hardness and consensus

Intuitively, we have argued that in the event that the data has consensus, the problem of finding the Kemeny ranking becomes tractable. Formally, one would like a function of the data, which we denote symbolically $h(n, \{\pi_{1:N}\})$, that accurately upper bounds the time to solve the current Kemeny ranking instance.

A remarkable theoretical approach in this direction is [5] which finds such a function h which is exponential in the Kemeny score. Other approximation bounds and worst case bounds exist, see e.g. SvZ and [9], but they are not of practical utility for the Kemeny ranking problem.

Therefore, here we have taken the approach of experimentally searching for good surrogate functions to measure the amount of consensus and the hardness of the problem.

We formulated the following possible functions.

The lower bound to the optimal cost Defined as: $h_{low} = \frac{2}{Nn(n-1)} \sum_{a,b} \min(Q_{ab}, Q_{ba})$.

Related measures one can consider are: the upper bound

$$h_{uplow} = \frac{2}{Nn(n-1)} \sum_{a,b} [\max(Q_{ab}, Q_{ba}) - \min(Q_{ab}, Q_{ba})] \qquad (11)$$

$$= 1 - 2h_{low}; \tag{12}$$

the improved, but computationally more demanding bounds introduced in [12] and [10]; and lower bounds produced by running a few steps of `B&B` (a constant number of node expansions).

The uniformity $h_{unif} = \frac{1}{n(n-1)} \sum_{a,b} I(0.5 - \delta \leq Q_{ab} \leq 0.5)$, which is the proportion of weights Q_{ab} which are within δ from 0.5, for e.g. $\delta = 0.1$.

The intransitivity $h_{intrans}$, which is the proportion of *intransitive triplets* a, b, c in Q. An unordered triplet of alternatives (a, b, c) is called intransitive if all three of Q_{ab}, Q_{bc}, Q_{ca} are either simultaneously greater than 0.5 or simultaneously smaller than 0.5.

The average rank entropy $h_{enti} = \frac{1}{n} \sum_{a=1}^{n} H(a)$, where $H(a)$ is the discrete entropy of the rank distribution of alternative a.

The average rank variance $h_{rankvar} = \frac{1}{n} \sum_{a=1}^{n} \mathrm{var}(\pi(a))$, with $\pi(a)$ being the rank of a in π.

The Borda cost h_{borda}, which is the cost of the ranking given by the `Borda` algorithm, normalized by N. We will also use the *normalized Borda cost* $\tilde{h}_{borda} = \frac{2}{n(n-1)} h_{borda}$.

Since we also have available the exact cost, and the running time of, the exact `ILP` algorithm, we use these two as surrogates for the amount of consensus, respectively for the problem hardness (in fact we use the *normalized optimal cost* $\frac{cost(\texttt{ILP})}{Nn(n-1)/2}$).

In the list above, we aimed to include measures of “consensus” which are varied, natural, and tractable to compute. If we take the running time of the `Borda` algorithm as a working definition for “fast,” then all measures are fast, with the exception of $h_{intrans}$ which is cubic in n.

We computed these consensus measures for all data sets we used, and investigated extensively the dependencies of these measures on each other and on the optimal cost and `ILP` running time. Here we include only the findings most significant for our purpose.

Figure 12 shows that the Borda cost, h_{borda}, is a good predictor of the `ILP` running time. We have already seen in Figure 8 that the running time of algorithms on the `websearch` datasets grow roughly exponentially with n. We prefer the Borda cost to the Borda running time, not just because the former relationship appears to hold better on the ensemble of all our data sets, but also because the running time of `Borda` can be subject to much more variation and simply harder to measure. This is useful, since as we said before, the running time of the exact `ILP` whether we can afford to run the algorithm or not, is a de facto good upper bound on the tractability of the instance at hand. The linear regression obtained from the 53 data sets is

$$\log_{10} \text{ILP run time} \;=\; 1.46 \log_{10} h_{borda} + 5.5335$$

with an R^2 value of 0.95. Removing the logarithms we get the following relationship

$$\text{ILP run time} \;=\; 34,160 ns \times h_{borda}.$$

We have also found a weaker dependence of the `ILP` running time on h_{enti}, the rank entropy, and another, non-linear dependence on $\sqrt{h_{rankvar}}$, the rank standard deviation. These are not shown.

While h_{borda}, together with the experimental results of Section 5, can indicate how tractable are the more expensive algorithms like `ILP` or `B&B` on the problem instance, it does not tell us whether to expect a significant improvement in cost from actually running them. Figure 13 shows that our proxy measures, the tractability as expressed by the `ILP` run time and the amount of consensus as expressed by the normalized optimal cost, are not predictive of each other.

7.5. A simple rule

Below we propose a simple way to determine the regime, strong, weak or no consensus. In fact, more practically, we introduce a simple rule to predict whether running a high performance Kemeny ranking heuristic, like `B&B` or `LS`, will result in significant improvements to the cost.

We know that h_{low} and $\tilde{h}_{borda}$ represent respectively a lower and an upper bound to the normalized optimal cost. Hence, one expects that when these are close, the Borda heuristic is practically optimal, and there is little to gain from another, more sophisticated algorithm. Figure 14 shows, for two LS algorithms, QuickSort+LS and BB10+LS, the relative gain in cost $cost/cost(\texttt{Borda})$ versus the ratio $\tilde{h}_{borda}/h_{low}$. This dependence is remarkably close to linear, with slope 1. Qualitatively similar plots were obtained for other LS, B&B+LS and pure B&B algorithms. Thus, the abscissa is a good predictor of what we stand to gain from running a more expensive algorithm.

Moreover, the data sets separate cleanly into the websearch data sets, the group for which largest gains are expected, the skiers data sets (the original data set and two variations where we randomly perturbed the ranks by a small amount), and the group of synthetic data sets, all with $\tilde{h}_{borda}/h_{low} < 1.02$. This last group includes both the strong consensus data sets (like Mallows' with $\theta = 0.1$), and the no consensus data sets (like random, the red dot in the left graph). In other words, the values of $\tilde{h}_{borda}/h_{low}$ distinguish between the weak consensus regime on one side and the no consensus or strong consensus regimes on the other side, with $\tilde{h}_{borda}/h_{low}$ away from 1 indicating a likelihood that with more running time one can improve the cost.

We caution that there can be cases when the optimal cost is near or even identical to the Borda cost, even though $\tilde{h}_{borda}/h_{low}$ is high. This can happen, for instance, with a cyclic data set with 3 alternatives $a \prec b \prec c, c \prec a \prec b, b \prec c \prec a$. An interesting theoretical question, which is to our knowledge unsolved, is finding lower and upper bounds on the optimal cost, given n and $\tilde{h}_{borda}/h_{low}$.

8. Conclusions

We have performed an extensive comparison of algorithms for the Kemeny rank aggregation problem, originating in social choice theory, machine learning, and theoretical computer science. The problem being NP-hard, the focus has been on establishing the best trade-offs between search time and performance.

8.1. Conclusions on Kemeny ranking algorithms

Our first conclusion is with respect to the available Kemeny ranking algorithms. The first choice, whenever available, is to search for the exact solution. In our experiments, the commercial ILP solver CPLEX found the optimum in every instance. If the user is in one of these fortunate situations, then having a provably optimal solution is of additional importance in a social choice situation (as it removes disagreements or suspicions related to which approximation to be used).

On the other hand, the focus of this paper is on possible replacements of the exact `ILP` algorithm when using it is not practical, and we assume this to be the case from now on. We find that, in spite of the many algorithms formulated, a user has only a few reliable choices.

There are the fast and cheap algorithms like `Borda` and perhaps `Copeland`. However, if one can afford up to two orders of magnitude more time, one can hope improve the cost significantly by running an `LS` algorithm. The dominance of `LS` algorithms for good cost requirements with large but tractable running time has already been remarked on by SvZ. We have shown additionally that the theoretically exact `B&B` algorithm introduced by [24] can form the basis for a second family of approximate algorithms, which also offer good cost with run times comparable with `LS`. Note that the improvements are potentially significant (up to 10-12%) only if the problem has consensus, but is not too easy; this is a second aspect we study in our paper.

In our experiments, `LS` dominate `B&B` type algorithms slightly, while a combination of `B&B` followed by `LS` is the most competitive algorithm we found. We note that in practice, one will likely find that the running times for the algorithms in these categories are implementation and machine dependent. This is why we do not rule out either of these two categories of algorithms for the time being.

In particular, `LS` has the following advantages: it uses a fixed amount memory, is easy to program, and presently achieves better cost for running time than `B&B`. The running time is unpredictable[9], however.

The `B&B` approximate algorithm family offers some advantages, too. It offers control of the memory and time resources with one parameter (in our experiments, memory), which tracks extremely well the cost vs time trade-off. This is particularly attractive, because with a 10 times faster implementation, these algorithms can smoothly fill in the concave region of the Pareto boundaries where no algorithms are available. In addition, a `B&B` method returns a lower bound on the Kemeny cost, detects when the optimum is

[9]However, a `LS` algorithm, just like a `B&B` one, can be stopped any time. We did not experiment with this class of methods in our paper.

found[10], and in general, because of its global approach, can produce a wealth of diagnostic information.

There is a category of algorithms not studied here, that appear as very promising complements to any high performance algorithm. These are the set of exact preprocessing methods that can reduce the size of the problem without affecting the solution. In this category we see the Branch and Bound solver for the Integer Program described in [12] with improved heuristics in [10], the heuristics in [8], and the exact transformations of [4].

8.2. Conclusions on Kemeny ranking problems

First, our paper demonstrates empirically that Kemeny ranking problems are not uniformly hard. The truly hard cases are the ones where no consensus exists, while the cases when the data exhibits consensus become gradually tractable. Moreover, tractability depends both on consensus and on the problem size n. Such ideas are foreshadowed in other experimental trials, like [12] and [10] which show a connection between problem difficulty and consensus.

In the extremely hard and extremely easy cases, namely the *no consensus* or *strong consensus* cases, one has limited possibility to trade off computing power for improvements in the (approximate) Kemeny cost or Kemeny ranking. In either case, the fast `Borda` algorithm will be close to optimal.

We found however evidence of the existence of a middle ground of *weak consensus* problems. On such problems, the tradeoffs described in Section 8.1 above are possible, and this is the regime for which developing new algorithms can bring the most gains.

Second, we quantify the hardness and the consensus, in order to provide quantitative criteria for the choice of algorithm. We introduce a simple and practical criterion, the ratio between an upper and a lower bound to the optimal cost, to detect the possibility of being in the weak consensus regime, and thus to help a user decide whether to run a computationally expensive algorithm or not.

The work of [5] has conceptual similarities to ours: they introduce a variety of consensus measures, all but the optimal cost being tractable, and prove (exponential) upper bounds on the running time as a function of these.

[10]This may not be a real option when an exponential number of Kemeny rankings exist, since `B&B` will have to explore them all before declaring optimality.

Acknowledgements

We thank the anoymous reviewers for the constructive feedback that significantly improved this paper. MM acknowledges partial support from NSF award IIS-0535100.

References

[1] Ailon, N., Charikar, M., & Newman, A. (2005). Aggregating inconsistent information: ranking and clustering. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)* (pp. 684–693). Association for Computing Machinery (ACM).

[2] Arrow, K. J. (1963). *Social choice and individual values*. Yale University Press.

[3] Bartholdi, J., Tovey, C. A., & Trick, M. (1989). Voting schemes for which it can be difficult to tell who won. *Social Choice and Welfare*, *6*, 157–165.

[4] Betzler, N., Bredereck, R., & Niedermeier, R. (2010). Partial kernelization for rank aggregation: Theory and experiments. In *Proceedings of the International Symposium on Parameterized and Exact Computation (IPEC'10)* Lecture Notes in Computer Science. Springer-Verlag.

[5] Betzler, N., Fellows, M. R., Guo, J., Niedermeier, R., & Rosamond, F. A. (2009). Fixed-parameter algorithms for kemeny rankings. *Theoretical Computer Science*, *45*, 455–4570.

[6] Borda, J. (1781). *Memoire sur les elections au scrutin*. Histoire de l'Academie Royal des Sciences.

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

[8] Cohen, W. W., Schapire, R. E., & Singer, Y. (1998). Learning to order things. *Journal of Artificial Intelligence Research*, *10*, 243–270.

[9] Coleman, T., & Wirth, A. (2008). Ranking tournaments: local search and a new algorithm. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 133–141).

[10] Conitzer, V., Davenport, A., & Kalagnanam, J. (2006). Improved bounds for computing kemeny rankings. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)* (pp. 620–627). Boston, MA.

[11] Copeland, A. (1951). *A 'reasonable' social welfare function*. Technical Report Seminar on Applications of Mathematics to Social Sciences, University of Michigan.

[12] Davenport, A., & Kalagnanam, J. (2004). A computational study of the kemeny rule for preference aggregation. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)* (pp. 697–702). San Jose, CA.

[13] Diaconis, P., & Graham, R. L. (1977). Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B*, *39*, 262–268.

[14] Dwork, C., Kumar, R., Naor, M., & Sivakumar, D. (2001). Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference (WWW)* (pp. 613–622).

[15] Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Chapman and Hall.

[16] Ephrati, E., & Rosenschein, J. S. (1993). Multi-agent planning as a dynamic search for social consensus. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 423–429). Morgan Kaufmann.

[17] Fligner, M. A., & Verducci, J. S. (1986). Distance based ranking models. *Journal of the Royal Statistical Society: Series B*, *48*, 359–369.

[18] Kemeny, J. L., J. G.and Snell (1962). *Mathematical models in the social sciences*. New York: Blaisdell.

[19] Kenyon-Mathieu, C., & Schudy, W. (2007). How to rank with few errors. In *Proceedings of the Thirty Ninth Annual ACM Symposium on Theory of Computing (STOC)* (pp. 95–103).

[20] Luce, R. D. (2005). *Individual choice and behavior: a theoretical analysis*. Dover.

[21] Mallows, C. L. (1957). Non-null ranking models. *Biometrika*, *44*, 114–130.

[22] Mandhani, B., & Meila, M. (2009). Tractable search for learning exponential models of rankings. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)* (pp. 392–399). volume 5.

[23] Mao, Y., & Lebanon, G. (2008). Non-parametric modelling of partially ranked data. *Journal of Machine Learning Research*, *9*, 2401–2429.

[24] Meila, M., Phadnis, K., Patterson, A., & Bilmes, J. (2007). Consensus ranking under the exponential model. In *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 285–294).

[25] Meilă, M., & Bao, L. (2010). An exponential family model over infinite rankings. *Journal of Machine Learning Research*, *10*, 3481–3518.

[26] Pennock, D. M., Horvitz, E., & Giles, C. L. (2000). Social choice theory and recommender systems: analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)* (pp. 729–734). AAAI Press.

[27] Plackett, R. L. (1975). The analysis of permutations. *Applied Statistics*, *24*, 193–202.

[28] Schalekampf, F., & van Zuylen, A. (2009). Rank aggregation: together we're strong. In *Proceedings of the 11th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 38–51).

[29] Young, H. P., & Levenglick, A. (1978). A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics, C*, *35*, 285–300.

[30] Young, P. (1995). Optimal voting rules. *Journal of Economic Perspectives*, *9*, 51–64.

[31] van Zuylen, A., & Williamson, D. P. (2007). Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *Proceedings of the 5th Workshop on Approximation and Online Algorithms (WAOA)* (pp. 260–273).

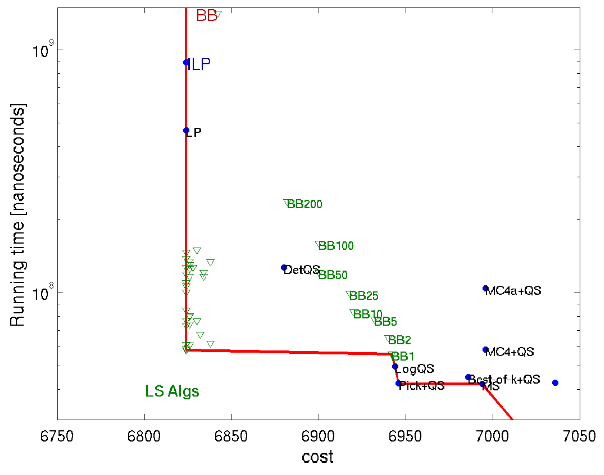


Figure 6: `skiers` dataset results. Detail of the upper part of the Pareto boundary. `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles.

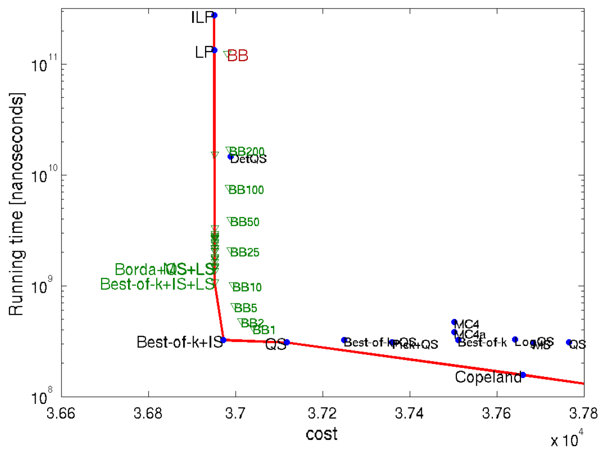


Figure 7: `websearch` dataset results (averaged). Detail of the upper part of the Pareto boundary. `LS` algorithms and `B&B` when it reverts to beam search are shown in green triangles.

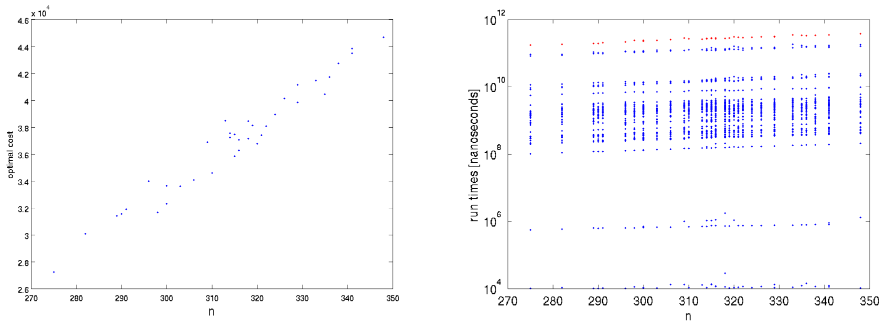


Figure 8: `websearch` dataset: normalized optimal cost vs. n (left) and normalized running times of all algorithms vs. n (right). The runing time of `ILP` is shown in red.

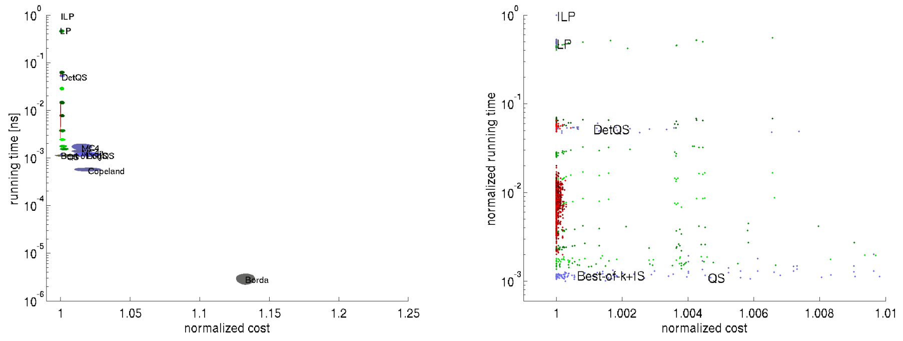


Figure 9: `websearch` dataset: normalized running time vs. normalized cost. **Left:** mean and standard deviations by algorithm. Some algorithms with high cost and high cost variance were excluded. **Right:** detail of the top-left part of the graph. Here a dot represents each individual run. The `B&B` algorithms are in shades of green, the `LS` algorithms are in shades of red (clustered near the running time axis), and the other algorithms are in shades of blue. Note that the `B&B` and the other algorithms cluster by running times, while the `LS` algorithms runs do not appear differentiated.

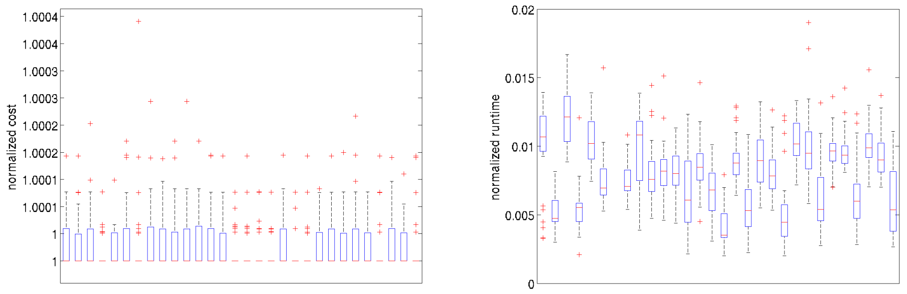


Figure 10: `websearch` dataset: normalized optimal cost (left) and normalized running times (right) for the `LS` algorithms. Boxplots are over the data sets. In both graphs, the algorithms were sorted by their *mean* running time.

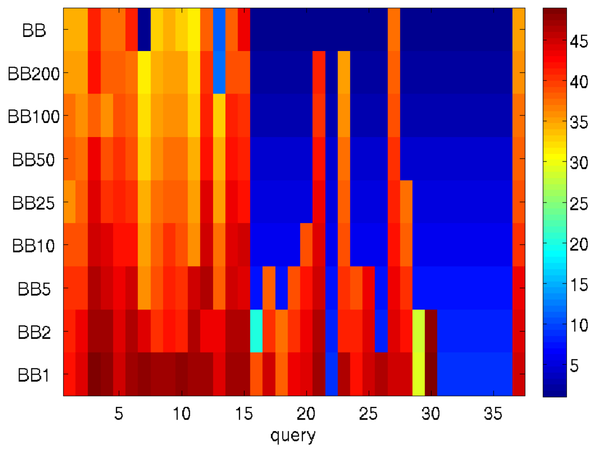


Figure 11: `websearch` dataset: ranks of the `B&B` algorithms w.r.t. cost, for each of the 37 queries / data sets. Note that most ranks are either below 10 or above 30. There are 9 `B&B` beam sizes, and 30 `LS` algorithms. In computing the ranks, ties were broken alphabetically, meaning that if a `B&B` algorithm attains the optimal cost it will be in ranks 1–9. Conversely, if all the `LS` algorithms have better cost than a `B&B` algorithm, then its rank will be > 30 (because `ILP` will also be ranked better). The `B&B` algorithms are sorted by decreasing beam size, `B&B` is `B&B1000`; the queries are sorted by increasing n.

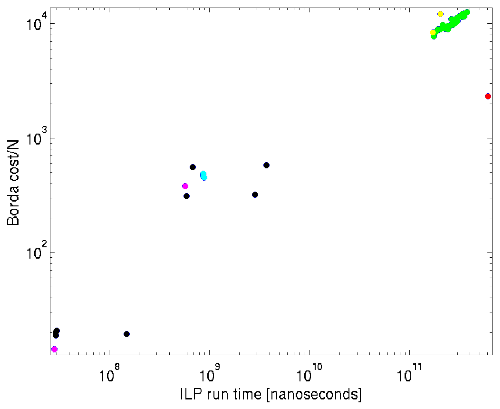


Figure 12: The Borda algorithm cost versus the running time of the `ILP` algorithm on the 53 data sets: Mallows' model (black), Plackett-Luce (magenta), ski (cyan), `websearch` (green), `websearch` with the results of several queries merged together to form a single dataset (yellow), `random` (red).

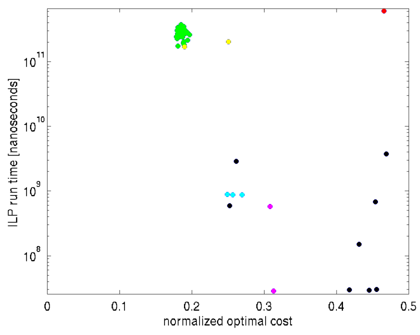
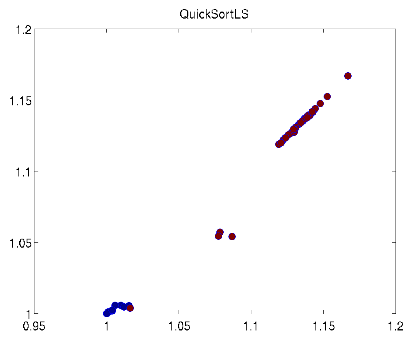


Figure 13: The optimal cost normalized by $Nn(n-1)/2$ versus the running time of the `ILP` algorithm on the 53 data sets: Mallows' model (black), Plackett-Luce (magenta), ski (cyan), `websearch` (green), `websearch` with the results for several queries merged to form a single dataset (yellow), `random` (red).



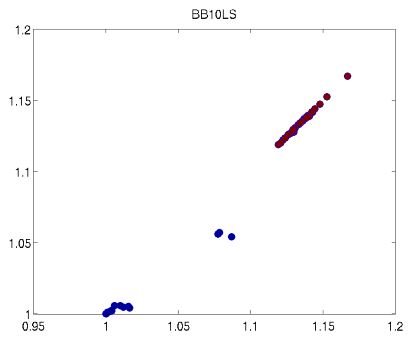


Figure 14: Tradeoffs for two high performance algorithms. The horizontal axis represents $\tilde{h}_{borda}/h_{low}$; the vertical axis represents `Borda` cost/algorithm cost. Both quantities are always greater or equal to 1. The color (red is higher) measures the $\log_{10}$ or the relative running time of the high performance algorithm w.r.t the `Borda` algorithm; its maximum is 3.7 for `QuickSortLS` and 3.5 for `BB10+LS`.