Computing the Consensus Permutation in Mallows Distribution by Using Genetic Algorithms

Juan A. Aledo, Jose A. Gámez, and David Molina

Departamento de Matemáticas, Departamento de Sistemas Informáticos
Universidad de Castilla-La Mancha
{juanangel.aledo,jose.gamez}@uclm.es, d.molina@estudiante.uam.es

Abstract. We propose the use of a genetic algorithm in order to solve the rank aggregation problem, which consists in, given a dataset of rankings (or permutations) of n objects, finding the ranking which best *represents* such dataset. Though different probabilistic models have been proposed to tackle this problem (see e.g. [12]), the so called *Mallows model* is the one that has more attentions [1]. Exact computation of the parameters of this model is an NP-hard problem [19], justifies the use of metaheuristic algorithms for its resolution. In particular, we propose a genetic algorithm for solving this problem and show that, in most cases (specially in the most complex ones) we get statistically significant better results than the ones obtained by the state of the art algorithms.

Keywords: Mallows model, rank aggregation, genetic algorithms.

1 Introduction

In this paper we focus on a machine learning problem in which instances in the dataset are *permutations* or *rankings* of n objects. This is a problem which has gained popularity in the last years because it finds applications on several fields: preference lists [14], voting in elections [8], information retrieval [4], combinatorial optimization [3], etc.

Basically, the problem we face is the following one: given a dataset containing N rankings representing the preferences of N judges (or the output of several search engines, etc.), can we tackle these data in a compact way for making some future decisions? That is, can we identify the consensus ranking?

The answer to this question is the well known *rank aggregation problem*, where the goal is to obtain the ranking which best represents all the input ones.

Although different probabilistic models have been proposed to deal with this problem (see e.g. [12]), there is one whose use has gained popularity in the specific literature: the *Mallows model* [18]. This model has certain resemblance to the Gaussian distribution, as it is specified by two parameters: a permutation π_0 which can be seen as the consensus ranking, and a dispersion parameter θ (see Section 2.2 for details). Exact computation of these parameters can be done from a dataset of rankings, though worst-case exponential time is required because of the NP-hardness of the problem [19].

If π_0 is known, then θ can be easily estimated by using a binary search over a suitable real interval. For this reason, many approaches to estimate the Mallows model parameters focus on the problem of computing the consensus ranking. This is the case of the study carried out by Ali and Meila [1], where they analyzed a big deal of heuristics and exact search algorithms to solve this problem. The main conclusion is that for small values of n, integer linear programming (ILP) should be used because it returns the exact solution. However, for large values of n, in general, an approximate version of the A^* (branch and bound) algorithm presented in [19] is the best choice.

Nevertheless, in that study we missed the use of some metaheuristic approaches. Concretely, the segment of problems with low values of θ and large n are specially suitable for this kind of algorithms, as our work will show for the particular case of genetic algorithms. In fact, for this values of the parameters, the consensus among the input rankings is small, making the search particularly difficult. In these cases, ILP for attaining the exact solution is infeasible, and also the number of nodes for the branch and bound algorithm A^* grows so much that an approximate version of it must be used. More concretely, this approximate version *limits* the size of the queue of nodes to explore, so increasing the risk of loosing promising paths which eventually could drive to the best solution.

Thus, our aim is to study the applicability of metaheuristics algorithms to the problem of obtaining the consensus ranking. Concretely, as a first approximation we propose to use *genetic algorithms* [13] to guide the search process (see Section 4). In this sense, the authors have developed a comparison study (not included in this paper) among several genetic algorithms in order to determine the most suitable one for solving the rank aggregation problem. As a result of such study, we conclude that the algorithm which we will call GA presents the best behavior (in comparison to a wide family of generic algorithms included in [17]) to solve our particular problem. Then, we compare it with the best algorithms tested in [1], showing that in most of the cases, and specially in the most complex ones, the GA algorithm obtains statistically significant better results.

The paper is organized as follows. In Section 2 we provide background knowledge on the problem under study and the Mallows model. In Section 3 we briefly review related proposals and concretely the outstanding algorithms used in the study carried out in [1]. Section 4 is devoted to describe the details of the POS-ISM genetic algorithm that we propose to approach the problem. In Section 5 we present an experimental study to test the proposed approach and discuss the obtained results. Finally in Section 6 we provide some conclusions.

2 Preliminaries

Next we provide the notation to be used and some background.

2.1 Ranks/Permutations

Suppose we have n items labeled $1, 2, \ldots, n$ that are to be ranked. Then, any permutation π of these items represents a ranking. The space of all the possible rankings agrees with the *Symmetric group* $\mathbb{S}_n$:

Definition 1. *(Symmetric Group)[9] The symmetric group* $\mathbb{S}_n$ *is the group whose elements are all the permutations of the n symbols* $\{1, 2, \ldots, n\}$*, and whose group operation is the composition of such permutations, which are treated as bijective functions from the set of symbols to itself. Since there exists* $n!$ *different permutations of size n, the order of the symmetric group is* $n!$*.*

We will use $n-$tuples to represent rankings. Thus, by $\sigma = (x_1,\ x_2, \cdots, x_n)$ we denote the ranking whose first position is x_1, the second is x_2, etc, and we use $\sigma(j)$ to denote the j-th element of σ.

To solve the rank aggregation problem and also to estimate the consensus ranking, we need to establish a way for measuring the difference between rankings. To do this we use a *distance* which allows us to know how similar they are. Although different distances are available in the literature, we describe here the *Kendall tau* distance, because it is usually considered for the definition of Mallows distribution.

Definition 2. *(Kendall Distance[16]) The Kendall distance* $d(\pi, \sigma)$ *between two rankings* π *and* σ *is defined as the total number of item pairs over which they disagree. There is disagreement over an item pair* (i, j) *if the relative ordering of* i *and* j *is different in* π *and* σ*.*

2.2 The Mallows Model

The Mallows model [18] is a distance-based probability distribution over permutation spaces which belongs to the exponential family. Given a distance over permutations, it can be defined by two parameters: the central permutation π_0, and the spread parameter θ.

Definition 3. *(Mallows model) [18] The Mallows Model is the probability distribution that satisfies, for all rankings* $\pi \in \mathbb{S}_n$*,*

$$P(\pi) = \frac{e^{-\theta \cdot d(\pi, \pi_0)}}{\psi(\theta)}, \tag{1}$$

where rankings π_0 *and* $\theta \geq 0$ *are the model parameters and* $\psi(\theta)$ *is a normalization constant (see [11]).*

The parameter θ of the Mallows model quantifies the concentration of the distribution around its peak π_0. For $\theta > 0$, the probability of π_0 is the one with the highest probability value and the probability of the other $n! - 1$ permutations decreases with the distance from the central permutation (and the spread parameter θ). For $\theta = 0$, we just get the uniform distribution. Because of these properties, the Mallows distribution on the space of permutations is considered analogous to the Gaussian distribution on the space of permutations.

2.3 The Kemeny Ranking Problem

As aforementioned, computing the consensus ranking is equivalent to the rank aggregation problem, also known as the Kemeny ranking problem [15].

Definition 4. *(Kemeny Ranking Problem) [1]*
Given a set of N rankings (permutations) $\pi_1, \pi_2, \cdots, \pi_N$ of size n, the Kemeny ranking problem consists in finding the ranking π_0 that satisfies

$$\pi_0 = argmin_\pi \frac{1}{N} \sum_{i=1}^{N} d(\pi_i, \pi), \tag{2}$$

where $d(\pi, \pi')$ stands for the Kendall distance between π and π'. π_0 in Equation (2) is the permutation that minimizes the total number of disagreements with the rankings contained in the set, and is called the Kemeny ranking of the set.

Finding the Kemeny ranking is an NP-hard problem for N≥4 (see e.g. [10]). The problem of computing this raking is nowadays an active field of research and several proposals for its exact computation have been presented, of course, without polynomial running time guarantees. Besides exact algorithms, many of heuristic nature have been also proposed (see e.g. [1]).

In this work we propose to test the use of genetic algorithms to guide the search. As we will see in Section 5, by means of this kind of algorithms we are able to get good solutions to complex problems; more specially to those with large dimension n and small degree of consensus θ.

3 Related Works

In this paper we rely on the excellent experimental comparison carried out by Ali and Meila [1]. In their study they compare a wide family of algorithms including exact Integer Linear Programming, Branch and Bound, specific heuristic and voting algorithms, and some others approximate algorithms. From their results/conclusions and taking into account our motivation, we have selected the best ones which we describe below. In order to do that, let us to introduce a data structure which make easier their description.

As in [1], given permutations π_1, $\pi_2, \ldots, \pi_N$, the *precedence matrix* $Q = [Q_{ab}]_{a,b=1:n}$ is defined as

$$Q_{ab} = \frac{1}{N} \sum_{i=1}^{N} 1(a \prec_{\pi_i} b), \tag{3}$$

where $1(\cdot)$ is the indicator function and $\prec_\pi$ means "precedes in the ranking π". In other words, Q_{ab} represents the fraction of times that item a is ranked before than item b across all the input rankings.

The following algorithms show the best behaviour according to [1]:

- **Integer Linear Programming (ILP).** It is an exact approach based on a mathematical formulation of the problem [6]. We do not include it in our experimentation because it runs out of memory for large values of n.
- **Branch and Bound (B&B) Search.** This algorithm relies on the use of the well known A^* algorithm jointly with the admissible heuristics proposed in [19], which guarantees to obtain the exact solution. However, in the worst case the search tree has $n!$ paths and the search becomes intractable. As was showed in [1], it works fine when there is a strong agreement between the rankings $\pi_1,\ \pi_2,\ldots,\pi_N$, that is, when θ is large, because in these cases the algorithm only expands a limited number of nodes.
- **B&B with Beam Search.** The same algorithm described above but using a limited size for the queue of nodes to be expanded. In this way, the algorithm does not run out of memory, but now there is a hight risk, specially in complex problems, of pruning good paths. Thus, the algorithm becomes an approximate one, looking for a good tradeoff between memory requirements and accuracy. Several beam sizes are experimented in [1] and according to it we have set the beam size as 1000.
- **CSS.** A graph-based approximate algorithm that implements a greedy version of the *Branch and Bound* method introduced in [5].
- **Borda.** This is an approximate algorithm which computes the sums of the columns of Q (3), i.e. $q_a = \sum_b Q_{ab}$, and then returns the permutation that sorts q_a in descending order [2].
- **DK** This algorithm is a *Branch and Bound* solver for the Integer Program described in [7], enhanced with the improved heuristics presented in [6].

4 Proposed Genetic Algorithm

In this work we study the competence of *genetic algorithms* (GAs) [13] to face the problem under study. Undoubtedly, there is room for their application, because due to the global search they carry out, we can expect to find better solutions in complex problems. Of course, we know that more CPU time will be required, but we think this is not a problem, as the estimation/learning of π_0 can be done *off-line*, being time important only for posterior *inference* over the learnt model.

GAs are the best representative of *evolutionary computation* and they work by maintaining a population of solutions which evolves according to natural selection principles. In practice they are stochastic optimization algorithms where natural selection is guided by three main operators: *selection*, *crossover* and *mutation*. Figure 1 shows the scheme of a canonical GA. Now, we present the main design decisions and parameter setting for the proposed GA:

- **Individual/Chromosome Representation.** As any ranking can be the consensus ranking π_0, our chromosomes or potential solutions are permutations of the n items. Therefore we search in $\mathbb{S}_n$, whose cardinality is $n!$.

```
BEGIN GA
  Make initial population at random
  WHILE NOT stop DO
BEGIN
   Select parents from the population.
   Produce children from the selected parents.
   Mutate the individuals.
   Extend the population adding the children to it.
   Reduce the extended population.
END
  Output the best individual found.
END GA.
```

Fig. 1. Pseudocode of a canonical GA

- **Fitness Function.** The objective of our GA is to solve the Kemeny ranking problem (2), therefore the fitness of a given individual π is:

$$f(\pi) = \frac{1}{N}\sum_{i=1}^{N} d\left(\pi_i, \pi\right),$$

- **Population.** Looking for a tradeoff between efficiency and population diversity, we have set the population proportional to the problem dimension (complexity). Thus, our GA will have a population of $k \cdot n$ individuals, n being the number of objects to rank, and $k > 1$ an appropriate integer. After preliminary experiments with different values, we decided to set $k = 20$. The initial population is randomly generated.
- **Selection.** In order to maintain diversity, we use a selection mechanism with low selective pressure, concretely a tournament selection [20] of size 2. Thus, at each iteration we randomly select $k \cdot n$ pairs of chromosomes and the individual of each pair with better (small) fitness is selected.
- **Crossover.** To select the crossover (and mutation) operators we have considered a great deal of choices successfully tested for other problems in the search space of permutations. Concretely, we have experimented with several combinations of crossover-mutation operators taken from the study carried out in [17] for the TSP problem. From our study[1] we finally have chosen the pair of operators POS and ISM for crossover and mutation respectively.
  Roughly speaking, the *Position based crossover operator (POS)* works as follows: it starts by selecting a random set of positions; the values for these positions are kept in both parents; the remaining positions are filled by using the relative ordering in the other parent. For example, consider the parents (1 2 3 4 5 6 7) and (4 3 7 2 6 5 1), and the positions {2, 3, 5} are selected.

[1] We have tested different combinations in the design of the GA: several schemes (standard and steady-step), selection mechanisms and crossover-mutation pairs of operators were studied. Because of the lack of space here we only show the results for our *winner* model, letting the full comparison for a long version of this paper.

Then, in the first step children are created as (* 2 3 * 5 * *) and (* 3 7 * 6 * *), while the unused items have the following relative ordering in the other parent: (4761) and (1245). In the second step the empty positions are filled: (4 2 3 7 5 6 1) and (1 3 7 2 6 4 5). The pairs for crossover application are randomly formed from the selected individuals.

– **Mutation.** ISM (*Insertion mutation*) operator is the one that best combines with POS crossover operator according to our study for this problem. It randomly chooses an element in the permutation, which is removed from its current position and inserted in a new one randomly selected.
– **Next Population Construction.** We use a *truncation* operator, that is, the population obtained after crossover and mutation and the previous population are put together in a common pool, and the best $k \cdot n$ adapted individuals are selected.
– **Stopping Criterion.** We stop the algorithm when we detect it has converged or is stagnated. Concretely, we stop if after p generations the best individual has not changed. After several experiments, we have set $p = 60$.

5 Experiments

In this section we describe the experiments carried out to test the goodness of our proposal with respect to those showing an outstanding behaviour in [1].

5.1 Datasets

All the datasets in the experiments have been generated using the Mallows model. For each case we choose a permutation π_0 of size n and a value for θ. Then, N permutations are generated from the resulting Mallows model by sampling according to the procedure described in [19]. In our case, we have set π_0 as the identity permutation $\pi_0 = (1\ 2 \dots n)$. Regarding θ, we have tested four different values: 0.2, 0.1, 0.01 and 0.001. Remember that the greater the value, the stronger the consensus in the data, and therefore the easier to solve the resulting Kemeny ranking problem. As our goal is to test the algorithm in complex problems, we have set n to the following four values: 50, 100, 150 and 200. Regarding the number of instances, in all the cases we have set $N = 100$ in order to focus on the Mallows model parameters. In a future work we plan to study the impact of the number of instances (ranks) in the dataset.

Finally, for each of the 16 combinations of the previous parameters (4 θ's and 4 n's) we generate 20 different datasets of N instances, in order to average the results and avoid sampling effects. That is, we experiment with 320 datasets.

5.2 Methodology

The GA and the last four algorithms described in Section 3 are run for each one of 320 generated dataset. In the case of B&B only the approximate version (beam search) is used, because the exact one runs out of space in most of the cases.

Table 1. Mean Kendall distance for each algorithm over different parameter values

n=50				
	θ=0.2	**θ=0.1**	**θ=0.01**	**θ=0.001**
GA	18781.5*	32010.4*	55876.9*	56846.9*
B&B	18781.5*	32010.4*	55892.8$^-$	56866.2$^-$
CSS	18834.2$^-$	32088.3$^-$	56072.0$^-$	57049.9$^-$
Borda	18783.7$^-$	32019.4$^-$	55991.5$^-$	56970.1$^-$
DK	18781.6	32012.8	55958.2$^-$	56954.6$^-$
n=100				
	θ=0.2	**θ=0.1**	**θ=0.01**	**θ=0.001**
GA	41215.4*	78802.6*	215224.7*	230323.1*
B&B	41255.4*	78810.2$^-$	215298.6$^-$	230498.3$^-$
CSS	41320.1$^-$	79012.6$^-$	215745.0$^-$	231026.6$^-$
Borda	41257.1$^-$	78827.9$^-$	215530.1$^-$	230827.7$^-$
DK	41255.4*	78805.8$^-$	215429.4$^-$	230803.8$^-$
n=150				
	θ=0.2	**θ=0.1**	**θ=0.01**	**θ=0.001**
GA	63717.6*	126058.3*	458967.2*	519673.1*
B&B	63717.7	126061.0$^-$	459091.7$^-$	520123.3$^-$
CSS	63890.3$^-$	126417.1$^-$	459943.1$^-$	521001.5$^-$
Borda	63724.5$^-$	126096.4$^-$	459513.7$^-$	520699.8$^-$
DK	63717.7	126064.5$^-$	459349.8$^-$	520834.0$^-$
n=200				
	θ=0.2	**θ=0.1**	**θ=0.01**	**θ=0.001**
GA	86264.8*	173430.3*	769227.1*	923284.0*
B&B	86268.5$^-$	173439.5$^-$	769463.9$^-$	924155.7$^-$
CSS	86515.4$^-$	173942.9$^-$	770632.3$^-$	925365.5$^-$
Borda	86270.7$^-$	173481.0$^-$	769999.5$^-$	925021.0$^-$
DK	86265.0	173433.6$^-$	769713.6$^-$	925602.1$^-$

Regarding the GA, because of its stochastic nature, we carry out 5 independent runs for each dataset and the average of the five runs is used for comparison.

The code for B&B, Borda, CSS and DK is the one provided by Ali and Meila [1] and is written in Java. Starting from that package we have also coded our GA in Java. Experiments have been carried out in the clusters of the supercomputing service of the University of Castilla-La Mancha (Spain). Concretely, they run under Linux operating system and we have been allowed to book a maximum of 20 GB of RAM memory.

5.3 Results

Table 1 shows the obtained results. We have organized them from the easiest to the most difficult case. Thus, results for the smallest n are in the first rows, and results for largest θ are on the left. The content of each cell in the table accounts for the average of the 20 different datasets sampled for the corresponding (n, θ) pair. These *mean values* are expressed in terms of the Kendall distance between the output (permutation) provided by the algorithm and all the permutations in the generated dataset. So, the smaller the value, the better the permutation.

To make easier the interpretation of the results, the algorithm (some times more than one) obtaining the best result for each (n,θ) combination is marked with a star symbol. Furthermore, for obtaining sound conclusions a statistical analysis has been carried out. Thus, Wilcoxon test has been used to ascertain whether the behavior of the tested algorithms is statistically significant. The algorithm with the best average is used as reference and compared with the remaining ones using a significance level $\alpha = 0.05$. Algorithms showing a statistically significant performance worse than the best one are marked with a minus symbol.

5.4 Results Discussion

The first conclusion is clear: the GA always obtains the best result, beating in all the cases to CS and Borda algorithms. Regarding B&B and DK, as we can expect they are competitive with respect to the GA only in the *less complex* cases, that is those having large θ and/or small n. However, they perform significantly worse in the remaining ones.

6 Conclusions

A study about the applicability of GAs to the problem of consensus permutation estimation in Mallows parameter estimation has been carried out. The proposal obtains very good results in all the cases, being statistically significantly better than competing approaches in most cases, specially in the harder ones, that is, large number of items to be ranked and/or few consensus among the instances (permutations) in the dataset.

In the near future we plan to go on with this research by following several lines: (1) increasing the experimental study by testing larger values for n and considering the impact of the data set number of instances; (2) studying the behaviour of the GA and competing approaches when the data does not come from a pure Mallows distribution, but from a *mixture* of them; and (3) extending the approach to the *generalized* Mallows model, while consensus permutation and θ's values should be estimated simultaneously.

Acknowledgements. The authors want to thank Alnur Ali and Marina Meila for facilitating us the code of their algorithms used in the experimental comparison. This work has been partially funded by FEDER funds and the Spanish Government (MICINN) through project TIN2010-20900-C04-03.

References

1. Ali, A., Meila, M.: Experiments with kemeny ranking: What works when? Mathematical Social Sciences 64(1), 28–40 (2012)
2. Borda, J.: Memoire sur les elections au scrutin. Histoire de l'Academie Royal des Sciences
3. Ceberio, J., Mendiburu, A., Lozano, J.A.: Introducing the mallows model on estimation of distribution algorithms. In: Lu, B.-L., Zhang, L., Kwok, J. (eds.) ICONIP 2011, Part II. LNCS, vol. 7063, pp. 461–470. Springer, Heidelberg (2011)
4. Chen, H., Branavan, S.R.K., Barzilay, R., Karger, D.R.: Global models of document structure using latent permutations. In: Proceedings of Human Language Technologies, NAACL 2009, pp. 371–379 (2009)
5. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. J. Artif. Int. Res. 10(1), 243–270 (1999)
6. Conitzer, V., Davenport, A.J., Kalagnanam, J.: Improved bounds for computing kemeny rankings. In: AAAI, pp. 620–626 (2006)
7. Davenport, A.J., Kalagnanam, J.: A computational study of the kemeny rule for preference aggregation. In: AAAI, pp. 697–702 (2004)
8. Diaconis, P.: A generalization of spectral analysis with application to ranked data. The Annals of Statistics, 949–979 (1989)
9. Diaconis, P.: Group representations in probability and statistics. Lecture Notes - Monograph Series, vol. 11. Institute of Mathematical Statistics, Harvard (1988)
10. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW, pp. 613–622 (2001)
11. Fligner, M.A., Verducci, J.S.: Distance based ranking models. Journal of the Royal Statistical Society 48(3), 359–369 (1986)
12. Fligner, M.A., Verducci, J.: Probability models and statistical analyses for ranking data. Springer (1993)
13. Goldberg, D.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley (1989)
14. Kamishima, T.: Nantonac collaborative filtering: Recommendation based on order responses. In: The 9th International Conference on Knowledge Discovery and Data Mining (KDD), pp. 583–588 (2003)
15. Kemeny, J.L., Snell, J.G.: Mathematical models in the social sciences. Blaisdell, New York
16. Kendall, M.G.: A new measure of rank correlation. Biometrika 30, 81–93 (1938)
17. Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artif. Intell. Rev. 13(2), 129–170 (1999)
18. Mallows, C.L.: Non-null ranking models. I. Biometrika 44(1-2), 114–130 (1957)
19. Meila, M., Phadnis, K., Patterson, A., Bilmes, J.: Consensus ranking under the exponential model. In: 22nd Conf. on Uncertainty in Artificial Intelligence (2007)
20. Miller, B.L., Goldberg, D.E.: Genetic algorithms, tournament selection, and the effects of noise. Complex Systems 9, 193–212 (1995)