Clustering Search applied to Rank Aggregation

Luiz H. N. Lorena
UNIFESP-ICT
São José dos Campos, Brazil
Email: luiz.lorena@unifesp.br

Ana C. Lorena
UNIFESP-ICT
São José dos Campos, Brazil
Email: aclorena@unifesp.br

Luiz A. N. Lorena
INPE-LAC
São José dos Campos, Brazil
Email: lorena@lac.inpe.br

André C. P. L. F. Carvalho
USP-ICMC
São Carlos, Brazil
Email: andre@icmc.usp.br

***Abstract*—Several practical applications require joining various rankings into a consensus ranking. These applications include gathering the results of multiple queries in information retrieval, deciding the result of a poll involving multiple judges and joining the outputs from ranking classification algorithms. Finding the ranking that best represents a set of rankings is a NP-hard problem, but a good solution can be found by using metaheuristics. In this paper, we investigate the use of Clustering Search (CS) algorithm allied to Simulated Annealing (SA) for solving the rank aggregation problem. CS will clusters the solutions found by SA in order to find promising regions in the search space, that can be further exploited by a local search. Experimental results on benchmark data sets show the potential of this approach to find a consensus ranking, achieving similar or better solutions than those found by other popular rank aggregation strategies.**

I. Introduction

The need for the aggregation of multiple rankings arise in several real applications. For instance, many search engines need to join the results from different queries into a consensus list, where web-pages are ranked from those most related to the query to those less relevant [1]. This problem is also frequent in Machine Learning and Data Mining, when one needs to combine the outputs from various ranking classifiers [2] or to produce a list of important features according to different feature ranking techniques [3]. In social sciences this problem has been also largely studied, when one needs to join the votes of multiple judges in an election [4].

Formally, the rank aggregation problem can be defined as finding a permutation of items that represents a consensus for the permutations created by m different judges. There are different criteria for evaluating whether a given permutation represents a consensus. In this paper we use the Kendall-tau distance [5] and define the optimal consensus permutation as the one which has minimum Kendall-tau distance to the others. This permutation is named the Kemeny optimal ranking [6].

Finding the Kemeny optimal ranking is a NP-hard problem [4] for which various works provide approximate solutions [7]. In [8] the authors employ a Genetic Algorithm (GA) for approximating the Kemeny optimal ranking. By using a permutation-based coding GA, those authors show the suitability of this GA in finding consensus permutations for a benchmark of problems of increasing difficulty.

In this work, we employ Clustering Search (CS) [9], [10], [11] algorithm using Simulated Annealing (SA) [12] as it base metaheuristic for solving the rank aggregation problem. CS will clusters the solutions found by SA, grouping them in promising regions to be exploited by a local search mechanism.

Our results reinforce the suitability of metaheuristics for finding Kenemy optimal ranking and strengthen the conclusions pointed out in [8], that metaheuristics are promising tools for solving the rank aggregation problem. These results are validated using the same benchmark data sets from [8] and [13], allowing the comparison against their results.

This paper is organized as follows. Section II defines the rank aggregation problem. Section III describes the technique employed in this paper. Section IV presents the experiments performed, while Section V discusses the main conclusions from this work.

II. Rank aggregation

Given a set of n items, labeled as $1, 2, ..., n$, a permutation π corresponds to a particular ordering of these n items. This ordering can also be regarded as a ranking of the elements, from the most to the least important, according to some criterion. In the rank aggregation problem we are given a set M with m distinct rankings π_i and we must find a permutation π^* that represents a consensus between them.

One way to express consensus is through similarity between the permutation π^* and other m permutations. Similarity between permutations can be estimated by applying a distance function. Permutations are similar when the distance between them is small. Therefore, for the rank aggregation problem, the permutation π^* must be chosen so as to minimize the sum of the distances between itself and each of the m input rankings.

There are two main distance functions considered when aggregating rankings [7]: Spearman's Footrule (d_s) [14] and Kendall-tau (d_k) [5]. This paper deals with minimizing the Kendall-tau distance, for which the best consensus permutation is named Kemeny optimal ranking [6]. Finding this ranking is an NP-hard problem for $m \geq 4$ and it is still an open issue whether the same condition holds for $m = 3$ [4]. Spearman's Footrule and Kendall-tau distances between two permutations π_i and π_j of size n are respectively given by equations 1 and 2.

978-1-4799-5618-0/14 $31.00 © 2014 IEEE
DOI 10.1109/BRACIS.2014.44

$$d_s(\pi_i, \pi_j) = \sum_{k=1}^{n} |\pi_{ik} - \pi_{jk}| \quad (1)$$

$$\begin{aligned} d_k(\pi_i, \pi_j) = \quad & |\{(a,b) : a < b, \\ & (\pi_i(a) < \pi_i(b) \wedge \pi_j(a) > \pi_j(b)) \\ & \vee \\ & (\pi_i(a) > \pi_i(b) \wedge \pi_j(a) < \pi_j(b))\}| \end{aligned} \quad (2)$$

Therefore, Spearman's Footrule distance measures the disarray of two ranks based on sum of the absolute differences between them, and can be computed in $O(n)$. On the other hand, the Kendall-tau distance counts the number of pairs of items (a, b) for which the permutations π_i and π_j disagree in their ordering, that is, where one of them gives a better ranking position to a and worse position to b while the other does the opposite. Our implementation of this distance is $O(n^2)$ but, with appropriate algorithms, it can be computed in $O(nlog(n))$ [15]. This modification may be necessary for permutations of size greater than 250 (larger permutation size of our data sets).

For the Rank Aggregation problem the best solution π^* (Kemeny optimal ranking) will be the one that minimizes:

$$v(\pi^*) = \sum_{i=1}^{m} d_k(\pi_i, \pi^*), \pi_i \in M \quad (3)$$

There are some algorithms to approximate the Kenemy optimal ranking in the literature [16], [17], [18]. The paper [13] presents a large set of techniques for searching the Kenemy optimal ranking, and recommend the use of a branch-and-bound $A*$ with beam-search, which reduces the number of nodes to be expanded in order to treat larger problems, since the basic algorithm runs out of memory and the problem becomes intractable. Finally, despite the large number of techniques considered in [13], the authors of [8] missed the use of some metaheuristic approach, and thus proposed a GA to be used in the same benchmark data sets from [13] and reported superior results.

There are a few previous papers addressing the rank permutation problems with metaheuristics. In [19] a Lamarckian GA which hybridizes metaheuristics based on the SA method or the noising method [20] is applied to find approximate solutions to this problem. The paper [21] uses a GA for optimizing Spearman's Footrule distance for partial lists (note that here we are dealing with total lists). They also discuss a parallel implementation of the GA, since this algorithm can be computationally costly for some practical applications, such as web-based search engines. In [22] a self-adaptive GA based on multiple genomic redundant representations, which encode different locality properties, was employed. Finally, [23] describes a ranking package named *RankAggreg* designed for combining ordered lists through different criteria and algorithms, which includes a GA. These works show that the use of metaheuristics is appropriate for approximating the Kenemy optimal ranking of multiple permutations.

This paper strengthen these results by employing a metaheuristic and shows the potential of employing a local search strategy for to better exploiting the search space, a theme not addressed in none of the previous papers.

III. Clustering Search

Clustering Search (CS) was earlier proposed by Oliveira and Lorena [24] as a hybrid method that combines metaheuristics and local search. The aim is to detect promising areas of the search space before applying local search procedures. Search is intensified only in areas of the search space that deserve special attention. These promising areas are discovered by dividing the search space into clusters, created by grouping the solutions provided by the base metaheuristic. Oliveira et al. [9] give a recent survey of CS characteristics and applications.

The CS algorithm is composed by three conceptually independent components: 1) Iterative Clustering Module (ICM), 2) Analyser Module (AM) and 3) Local Search Module (LSM). Figure 1 shows the conceptual design of the CS algorithm. In our case, the Simulated Annealing algorithm is used as a base metaheuristic. ICM is responsible to gather similar solutions into clusters. AM examines each cluster, at regular intervals, measuring its *activity level*. The *activity level* of a cluster is given by the number of solutions of the metaheuristic that are assigned to this cluster. This measure is called *cluster density*, and higher values indicate that the cluster is at a promising area. Finally, LSM module is responsible for applying a local search to the center of a cluster when its density is above a predefined threshold.

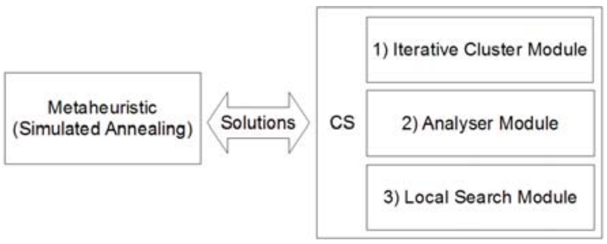

Figure 1: Conceptual design of CS algorithm.

Figure 2 shows a schematic representation of a search space. This figure illustrates how the ICM module behaves when a new solution provided by the metaheuristic is fed into the CS algorithm. Clusters are represented by their center c_i, namely the solution that best represents a group of solutions in a given area of the search space. Each cluster has a coverage radius R. The example contains four clusters, while the cross symbol represents a new solution provided by the metaheuristic. When this solution is presented, ICM checks for its similarity against all clusters centers, assigning it to the cluster with higher similarity. Similarity is determined by a *distance metric*, which is the Spearman's Footrule distance d_s in our case. In this example, the solution is assigned to c_3, because is closer to it and lies within its coverage radius.

To increase the chance of discovering better solutions a process called *assimilation* is conducted by ICM each time a new solution is assigned to a cluster and is within its coverage radius. It consists of using the path-relinking

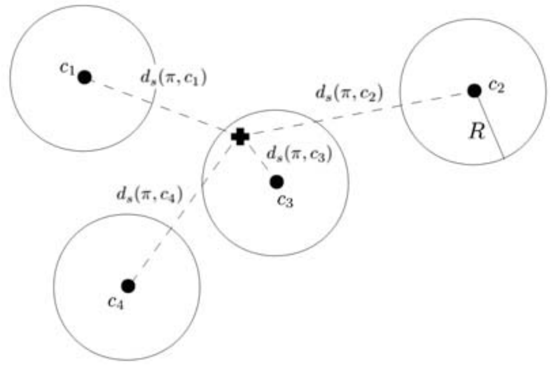

Figure 2: Clusters of solutions in the search space.

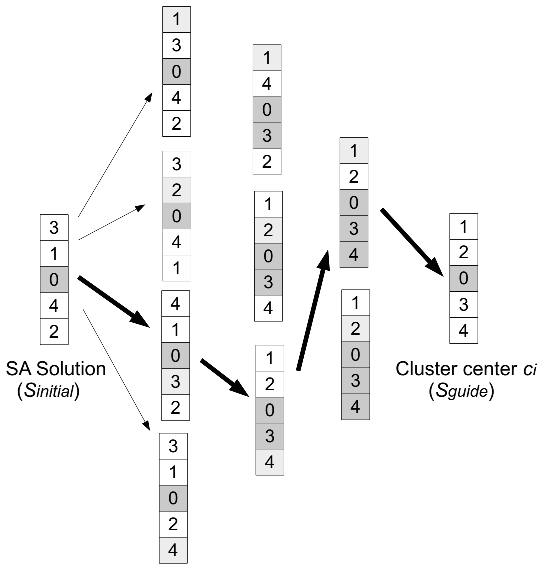

Figure 3: Path-relinking moves.

algorithm [25] to intensify and diversify the search within a group of solutions.

Figure 3 shows an example of the *assimilation* process. It makes exploratory movements in a neighborhood around an initial solution seeking to reach a guide solution. At each step the procedure examines all of the possible moves from the current solution and selects the one that has a better cost. The goal is to find better solutions in a path which interconnects them. The movements gradually introduce attribute information from the guide solution into the initial solution. Squares of darker color represent positions that are fixed (already have guide solution information), whereas lighter color squares represents positions where information from the guide solution is under analysis. The quality of the solution is evaluated every time an attribute information is introduced in the solution. Thick arrows show the path connecting the best solutions at each step. To reduce the computational burden of this process, the evaluation of all possible movements can be stopped when no further improvements to the solution are obtained, even if the path is not complete. The best solution found in path will replace the cluster center.

Algorithm 1 shows how CS interacts with the SA metaheuristic. Functions in lines 1 and 15 are described at Algorithm 2 and correspond to CS steps. Line 1 initializes the CS cluster centers and sets some initial parameters. Lines 2 to 4 initializes SA. Line 5 represents the main loop which controls the halt of the SA algorithm. Lines 7 to 13 perform the Metropolis algorithm [26]. At line 9 a neighborhood function is applied. In our case it consists of selecting one random position at the permutation and moving its corresponding value to another random position. Line 14 decreases the current temperature by α and checks if the solution provided by Metropolis is the best SA solution. At line 15 the CS Iterative Cluster Module (ICM) is called using as input argument the solution from the Metropolis algorithm. Finally, at line 16, the CS solution is checked against the best SA solution so far. The best value is then assigned to the global solution π^*.

Algorithm 1: Simulated Annealing (SA) with Clustering Search (CS)

```
input : The permutation matrix $M$ of size $m \times n$
        SA parameters: $T_0, T_c, SA_{MAX}, \alpha$
output: A consensus permutation $\pi^*$
1  CS.Initialization ()                         ▷ CS initialization
2  $\pi \leftarrow$ random initial permutation          ▷ SA start
3  $\pi_{SA} \leftarrow \pi$    $\pi^* \leftarrow \pi$
4  $T \leftarrow T_0$
5  while $T > T_c$ do
6      $iter \leftarrow 0$
7      while $iter < SA_{MAX}$ do
8          $iter \leftarrow iter + 1$
9          $\pi' \leftarrow$ Neighbor($\pi$)
10         if $v(\pi') < v(\pi)$ then
11             $\pi \leftarrow \pi'$
12         else
13             With probability $e^{-(v(\pi')-v(\pi))/T}$ set
               $\pi \leftarrow \pi'$
14     $T \leftarrow \alpha T$    $\pi_{SA} \leftarrow min(\pi, \pi_{SA})$
15     $\pi_{CS} \leftarrow$ CS.ICM($\pi$)                  ▷ CS ICM
16     $\pi^* \leftarrow min(\pi_{CS}, \pi_{SA})$
17 return $\pi^*$                                 ▷ SA end
```

Algorithm 2 presents the CS algorithm and its main components. The variable c_{TOTAL} is used to control the actual number of clusters. Line 3 in Procedure *Initialization* calculates the coverage radius R as the percentage of the maximum Spearman's Footrule distance between two permutations (ds_{MAX}) of size n. In lines 4 to 6 a random solution is assigned to each center and their density and inactivity levels are initialized.

Procedure ICM receives as argument a solution provided by the metaheuristic. Lines 2 to 9 are responsible to find the index of the best cluster i_{BEST}, which corresponds to that with lower Spearman's Footrule distance value. The remaining clusters have their inactivity increased (line 9). If the solution π is within the c_{BEST} coverage radius R (line 10) the solution is fed into the assimilation process (line 11). If the solution obtained is better than the current cluster center, it will be assigned as the new center and have its density and inactivity levels reset (lines 13 to 15); otherwise the inactivity of the center will increase and the algorithm checks for the possibility to create a new cluster using π as its center (lines 17 to 20). At line 21 the Analyser Module (AM) is called, which will return the best current solution within all clusters. This solution will be provided to the base metaheuristic (line 22).

Finally, in Procedure AM all cluster densities will be checked (line 3) and if they reach a threshold value den_{MAX} a local search will be performed by the LSM module (line 4). Local search consists of using the assimilation process between a center c_i and the remaining ones. This process is stopped whenever a better solution is found. This solution will replace the given cluster center (lines 5 to 7). If the inactivity level of the cluster reaches a threshold value $inact_{MAX}$, this cluster can be removed (lines 8 to 10). Lastly, line 11 search for the best solution within all clusters, and returns it.

IV. Computational results

The proposed solution was coded in Java 1.7.0_45 and the computational tests were performed in a i7-3517U @ 2.40 GHz processor with 8GB RAM running Windows 8 64-Bit. The data sets were previously used in [8] and gently provided by those authors. They model probability distributions over permutations by using the Mallows Model [27], which is a distance-based probability distribution over permutation spaces [8]. The data sets are generated by sampling from different instances of this distribution, which is given by Equation 4 [8]:

$$P(\pi) = \frac{e^{-\theta(d_k(\pi,\pi_0))}}{\phi(\theta)} \qquad (4)$$

Where π_0 is a central permutation and θ is a spread parameter that accounts for the concentration of the distribution around its peak. $\phi(\theta)$ is a normalization constant. Therefore, the permutation π_0 has a higher probability of being sampled, while the probabilities of the other $n! - 1$ permutations are reduced according to their distance to π_0.

For all data sets, [8] have employed $\pi_0 = 1, 2, 3, \ldots, n$. They varied the values of θ and n to produce different instances of the problem and generated $m = 100$ permutations. Therefore, the pair (θ,n) accounts for different complexities of the rank aggregation problem. While θ takes value in 0.2, 0.1, 0.01, 0.001, n takes value in 50, 100, 150, 200, 250. The most complex cases are those with smaller θ (bigger dispersion, less consensus) and larger permutation/rank size (n). For each of the 16 combinations of θ and n, 20 different data sets with $m = 100$ instances (permutations) were generated. Therefore, the results presented for each of the 16 combinations are averaged over 20 data sets.

To tune algorithms parameters, five instances were randomly chosen, one of each size (50, 100, 150, 200 and 250), and the following process was adopted: each

Algorithm 2: Clustering Search (CS)

```
input : A solution $\pi$ coming from metaheuristic
        CS parameters: $NC, DP, dens_{MAX}, inact_{MAX}$
output: A solution $\pi_{CS}$
1  Procedure Initialization()
2      $c_{TOTAL} \leftarrow NC$
3      $R \leftarrow DP * ds_{MAX}$
4      for $i \leftarrow 1$ to $c_{TOTAL}$ do
5          $c_i \leftarrow$ random solution
6          $dens_i \leftarrow 0$    $inact_i \leftarrow 0$

1  Procedure ICM($\pi$)
2      $i_{BEST} \leftarrow 0$    $dist_{BEST} \leftarrow \infty$
3      for $i \leftarrow 1$ to $c_{TOTAL}$ do
4          $dist_i \leftarrow ds(\pi, c_i)$
5          if $dist_i < dist_{BEST}$ then
6              $dist_{BEST} \leftarrow dist_i$
7              $i_{BEST} \leftarrow i$
8          else
9              $inact_i \leftarrow inact_i + 1$

10     if $dist_{BEST} \leq R$ then
11         $\pi_{NEW} \leftarrow$ PathRelinking($\pi, c_{i_{BEST}}$)
12         if $v(\pi_{NEW}) < v(c_{i_{BEST}})$ then
13             $c_{i_{BEST}} \leftarrow \pi_{NEW}$
14             $dens_{i_{BEST}} \leftarrow dens_{i_{BEST}} + 1$
15             $inact_{i_{BEST}} \leftarrow 0$
16     else
17         $inact_{i_{BEST}} \leftarrow inact_{i_{BEST}} + 1$
18         if $c_{TOTAL} < NC$) then
19             Create cluster with $\pi$ center
20             $c_{TOTAL} \leftarrow c_{TOTAL} + 1$

21     $\pi_{CS} \leftarrow$ AM()
22     return $\pi_{CS}$

1  Procedure AM()
2      for $i \leftarrow 1$ to $c_{TOTAL}$ do
3          if $dens_i \geq dens_{MAX}$ then
4              $\pi_{NEW} \leftarrow$ LocalSearch($c_i$)            ▷ LSM
5              if $v(\pi_{NEW}) < v(c_i)$ then
6                  $c_i \leftarrow \pi_{NEW}$
7                  $dens_i \leftarrow 0$    $inact_i \leftarrow 0$

8          if $inact_i \geq inact_{MAX}$ then
9              Remove cluster $i$
10             $c_{TOTAL} \leftarrow c_{TOTAL} - 1$

11     $i \leftarrow argmin\{v(c_i)\}\ i \in \{1, \ldots, NC\}$
12     $\pi_{CS} \leftarrow min(c_i, \pi_{CS})$
13     return $\pi_{CS}$
```

parameter was varied while the others were kept fixed. The algorithm was run five times for each parameter setting and for each instance, and the setting yielding the best average result was chosen. This methodology is used for tuning the SA and CS parameters described in Table I, which shows the best values found.

Simulated Annealing		
T_0	Initial temperature	5
T_c	Cooling temperature	0.0004
SA_{max}	Number of iterations of Metropolis	10000
α	Cooling rate	0.975
Clustering Search		
NC	Maximum number of clusters	10
DP	Percentage of maximum distance	0.4
$dens_{MAX}$	Maximum density for Local Search	2
$inact_{MAX}$	Maximum cluster inactivity allowed	7

Table I: Final parameters after tuning phase.

After parameter tuning, for each pair of (θ, n), five independent runs of the algorithm were carried out and an average performance was recorded. Table II shows the CS computational results (Kendal-tau distances) compared with those of the GA in [8] and of some algorithms described and detailed in [13]. B&B represents the branch and bound A* with beam search. CSS is a graph-based approximate algorithm [28]. DK is a solver for an integer program [29] with heuristics to reduce the search space. Borda is a method where each position in a rank is given a ponctuation [30]. The sum of the pontuactions achieved by the items in the different rankings is then used to devise a new ordering. Each cell in Table II accounts for the average of the 100 different experiments (20 data sets x 5 runs). The best overall results are highlighted in bold. CS found 13 (out of 20) new best solutions compared to the ones reported previously in the literature.

Comparing the results of the techniques in all data sets using the Friedman statistical test [31], there are significant differences of performance at 95% of confidence level. Applying the Bonferroni-Dunn post-test using the CS algorithm as baseline, we found that GA and CS performed closely, while CS outperformed all other techniques.

The larger and difficult problem (n = 250 and θ = 0.001) is approximately solved in 491.06 seconds by CS. This is also a case where CS reached better results than all other algorithms compared, as can be observed in Table IV. The smaller problems (with $n = 50$) were solved from 2.31 ($\theta = 0.2$) to 8.13 ($\theta = 0.001$) seconds. We judge this a very interesting result, since the time taken to obtain the solutions can be considered feasible for practical applications. We were not able to compare our processing time results against those from [8], [13], since they did not present this analysis.

The main parameters governing the benchmark instances are θ and n. Therefore, we analyze next the results of the algorithms CS, GA and B&B for the different values of these parameters. The GA results come from [8], while B&B is the best performing algorithm from [13].

Regarding the parameter n, we can observe that CS was the best performing technique in the majority of cases, although ties are observed for $n = 50$ and $n = 200$ with the GA. For $n = 150$, CS was always the best performing technique, despite the value of the parameter θ. B&B was able to achieve good results only for the smaller instances, and for a higher value of the spread

n=50	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
CS	18784.6	**32007.3**	**55873.2**	56855.7
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	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
CS	**41203.1**	78802.7	**215224.1**	**230196.9**
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	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
CS	**63687.75**	**125990.8**	**458678.9**	**519642.8**
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	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
CS	**86264.8**	173430.5	**769215.9**	**922854.6**
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
n=250	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
CS	108770.8	220658.7	**1130023.2**	**1442029.5**
GA	**108762.2**	**220656.4**	1130024.9	1442227.6
B&B	108765.4	220666.5	1130306.3	1443555.1
CSS	109079.6	221313.0	1131954.7	1445117.9
Borda	108771.9	220722.3	1131118.9	1444884.0
DK	108762.3	220663.1	1130708.5	1445374.6

Table II: Results.

parameter θ. This occurred because this technique can discard some of the good solutions, once it maintains only part of the search frontier of A* in order to remain feasible for larger problems. For the different θ values, the results of CS highlight again for more difficult instances, which are those for which the value of θ is lower.

Figure 4 summarizes which were the best performing algorithms for all data sets configurations, i.e., for all combinations of n and θ values. Higher values of n combined to lower values of θ correspond to more challenging problems, where the ranking length is higher and there is less consensus between the rankings. CS achieved the best results for 70% of the instances, specially for those which are harder to solve (with $n \geq 100$ and $\theta \leq 0.01$).

V. Conclusion

This paper applied the Clustering Search algorithm allied to the Simulated Annealing to solve the Rank Aggregation problem. Namely, the objective is to obtain a consensus ranking from a set of different rankings.

The experimental results shows that the proposed solution achieved similar or better solutions than those found by other popular rank aggregation strategies. These

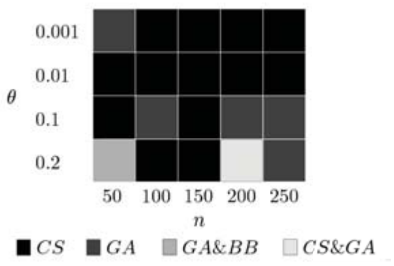

Figure 4: Best algorithm for each data set.

results reinforce the suitability of metaheuristics as an promising tools for solving the rank aggregation problem. Nonetheless, this work also supports that exploiting the search space through local search in promising regions of the search space, as done by Clustering Search, can lead to improvements.

As future work, we plan to employ the investigated technique in other rank aggregation data sets, including real data sets. Other metaheuristics can be employed with CS, as the GA from [8]. We believe that CS will be able to solve the problem in less iterations. Lower bounds can be found by linear programming techniques, and used to estimate the quality of CS solutions. A better parameter tuning for the algorithms can be employed, considering different parameter values for the distinct problem sizes.

VI. Acknowledgments

The authors would like to thank for the financial support of FAPESP and CNPq. We also would like to thank David Molina Garcia and Juan A. Aledo for providing the data sets.

References

[1] K. W. Lam and C. H. Leung, "Rank aggregation for meta-search engines," in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers*. ACM, 2004, pp. 384–385.

[2] M. Grbovic, N. Djuric, and S. Vucetic, "Multi-prototype label ranking with novel pairwise-to-total-rank aggregation," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2013, pp. 1358–1364.

[3] R. C. Prati, "Combining feature ranking algorithms through rank aggregation," in *Internatinoal Joint Conference on Neural Networks*, 2012, pp. 1151–1157.

[4] I. Bartholdi, J., C. Tovey, and M. Trick, "Voting schemes for which it can be difficult to tell who won the election," *Social Choice and Welfare*, vol. 6, no. 2, pp. 157–165, 1989.

[5] M. G. Kendall, "Rank correlation methods." 1948.

[6] J. Kemeny and J. Snell, *Mathematical Models in the Social Sciences*, ser. A Blaisdell book in the pure and applied sciences. Blaisdell Publishing Company, 1962.

[7] S. Vembu and T. Gärtner, "Label ranking algorithms: A survey," in *Preference Learning*, J. Fürnkranz and E. Hüllermeier, Eds. Springer Berlin Heidelberg, 2011, pp. 45–64.

[8] J. A. Aledo, J. A. Gámez, and D. Molina, "Tackling the rank aggregation problem with evolutionary algorithms," *Applied Mathematics and Computation*, vol. 222, pp. 632 – 644, 2013.

[9] A. C. M. Oliveira, A. A. Chaves, and L. A. N. Lorena, "Clustering search," *Pesquisa Operacional*, vol. 33, no. 1, pp. 105–121, 2013.

[10] R. L. Rabello, G. R. Mauri, G. M. Ribeiro, and L. A. N. Lorena, "A clustering search metaheuristic for the point-feature cartographic label placement problem," *European Journal of Operational Research*, vol. 234, no. 3, pp. 802–808, 2014.

[11] A. A. Chaves and L. A. N. Lorena, "Clustering search algorithm for the capacitated centered clustering problem," *Computers & Operations Research*, vol. 37, no. 3, pp. 552–558, 2010.

[12] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[13] A. Ali and M. Meilă, "Experiments with kemeny ranking: What works when?" *Mathematical Social Sciences*, vol. 64, no. 1, pp. 28 – 40, 2012.

[14] P. Diaconis and R. L. Graham, "Spearman's footrule as a measure of disarray," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 262–268, 1977.

[15] D. Christensen, "Fast algorithms for the calculation of kendall's τ," *Computational Statistics*, vol. 20, no. 1, pp. 51–62, 2005.

[16] A. Van Zuylen and D. P. Williamson, "Deterministic algorithms for rank aggregation and other ranking and clustering problems," in *Approximation and Online Algorithms*. Springer, 2008, pp. 260–273.

[17] C. Kenyon-Mathieu and W. Schudy, "How to rank with few errors," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 2007, pp. 95–103.

[18] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: ranking and clustering," *Journal of the ACM (JACM)*, vol. 55, no. 5, p. 23, 2008.

[19] I. Charon and O. Hudry, "Lamarckian genetic algorithms applied to the aggregation of preferences," *Annals of Operations Research*, vol. 80, pp. 281–297, 1998.

[20] ——, "The noising method: a new method for combinatorial optimization," *Operations Research Letters*, vol. 14, no. 3, pp. 133–137, 1993.

[21] M. Sufyan Beg, "Parallel rank aggregation for the world wide web," in *Proceedings of International Conference on Intelligent Sensing and Information Processing*. IEEE, 2004, pp. 385–390.

[22] M. L. Gargano and M. P. Kasinadhuni, "Rank aggregation for metasearch engines using a self-adapting genetic algorithm with multiple genomic representations," *Generations*, vol. 1, no. 2, p. 3, 2005.

[23] V. Pihur, S. Datta, and S. Datta, "Rankaggreg, an r package for weighted rank aggregation," *BMC bioinformatics*, vol. 10, no. 1, p. 62, 2009.

[24] A. C. Oliveira and L. A. Lorena, "Detecting promising areas by evolutionary clustering search," in *Advances in Artificial Intelligence–SBIA 2004*. Springer Berlin Heidelberg, 2004, pp. 385–394.

[25] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking," *Control and cybernetics*, vol. 39, no. 3, pp. 653–684, 2000.

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, p. 1087, 1953.

[27] C. L. Mallows, "Non-null ranking models. i," *Biometrika*, vol. 44, no. 1/2, pp. 114–130, 1957.

[28] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *J. Artif. Int. Res.*, vol. 10, no. 1, pp. 243–270, 1999.

[29] A. Davenport and J. Kalagnanam, "A computational study of the kemeny rule for preference aggregation," in *Proceedings of the 19th National Conference on Artifical Intelligence*, 2004, pp. 697–702.

[30] J. Borda, "Memoire sur les elections au scrutin," *Histoire de l'Academie Royale des Sciences*, 1781.

[31] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.