Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Heuristic Search for Rank Aggregation with Application to Label Ranking

Yangming Zhou
Sino-US Global Logistics Institute, Antai College of Economics and Management, Shanghai Jiao Tong University, Shanghai 200030, China
Data-Driven Management Decision Making Lab, Shanghai Jiao Tong University, Shanghai 200030, China
yangming.zhou@sjtu.edu.cn

Jin-Kao Hao*
Department of Computer Science, Université d'Angers, Angers 49045, France
jin-kao.hao@univ-angers.fr

Zhen Li
Tencent Technology (Shanghai) Company Limited, Shanghai 200233, China
acremanli@tencent.com

Fred Glover
Entanglement, Inc., Boulder, Colorado 80302, USA
fred@entanglement.ai

Rank aggregation combines the preference rankings of multiple alternatives from different voters into a single consensus ranking, providing a useful model for a variety of practical applications, but posing a computationally challenging problem. In this paper, we provide an effective hybrid evolutionary ranking algorithm to solve the rank aggregation problem with both complete and partial rankings. The algorithm features a semantic crossover based on concordant pairs and an enhanced late acceptance local search method reinforced by a relaxed acceptance and replacement strategy and a fast incremental evaluation mechanism. Experiments are conducted to assess the algorithm, indicating a highly competitive performance on both synthetic and real-world benchmark instances compared with state-of-the-art algorithms. To demonstrate its practical usefulness, the algorithm is applied to label ranking, a well-established machine learning task. We additionally analyze several key algorithmic components to gain insight into their operation.

Key words: Rank Aggregation, Label Ranking, Machine Learning, Evolutionary Computation, Metaheuristics.

1. Introduction

Rank aggregation is a classical problem in voting theory, where each voter provides a preference ranking on a set of alternatives, and the system aggregates these rankings into

a single consensus preference order to rank the alternatives. Rank aggregation plays a critical role in a variety of applications such as collaborative filtering (Huang and Zeng 2011, Li et al. 2021), multiagent planning (Gharaei and Jolai 2021), information retrieval (Tamine and Goeuriot 2022), and label ranking (Zhou et al. 2014, Destercke et al. 2015, Zhou and Qiu 2018, Alfaro et al. 2021). As a result, this problem has been widely studied, particularly in social choice theory and artificial intelligence.

Given a set of m labels $\mathcal{L}_m = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, a ranking with respect to $\mathcal{L}_m$ is an ordering of all (or some) labels that represent an agent's preference for these labels. Rankings can be either complete or partial. A complete ranking includes all the labels and can be identified with a permutation π of the set $\{1, 2, \ldots, m\}$ such that $\pi(\lambda_i)$ denotes the position of λ_i in the ranking π, that is, the rank of the label λ_i in the ranking π. For two labels λ_i and λ_j, $\pi(\lambda_i) < \pi(\lambda_j)$ indicates that λ_i is preferred to λ_j and this preference relation is represented by $\lambda_i \prec \lambda_j$. However, real-world problems usually include partial rankings, where only m' $(2 \leq m' < m)$ labels are ranked. For example, when customers' preference relations about a set of movies, books, and laptops, are collected, the preference information on some labels may not be available. In this case, partial rankings can be used to express these partial preference relations.

Rankings can also be classified as with or without ties. A tie means there is no preference relation among the ranked labels. The tied labels constitute a bucket. Therefore, an arbitrary ranking σ can be represented as a list of its disjointed buckets, ordered from the most to the least preferred, and separated by vertical bars. The labels between two consecutive vertical bars indicate a bucket. Formally, a ranking can be represented as follows:

$$\sigma = (\lambda_1^1, \lambda_2^1, \ldots, \lambda_{w_1}^1 | \lambda_1^2, \lambda_2^2, \ldots, \lambda_{w_2}^2 | \ldots | \lambda_1^k, \lambda_2^k, \ldots, \lambda_{w_k}^k)$$

where $1 \leq w_i \leq m$, $1 \leq k \leq m$, $2 \leq \sum_{i=1}^{k} w_i \leq m$, and the labels $\lambda_1^i, \lambda_2^i, \ldots, \lambda_{w_i}^i$ are in the i-th bucket. A pairwise preference $\lambda_i \succ \lambda_j$ is also denoted by $\lambda_i | \lambda_j$. Let $\mathcal{L}_4 = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ be the set of labels. Then $\sigma_1 = (1|4|3|2)$ represents a complete ranking without ties, $\sigma_2 = (1|3,4|2)$ represents a complete ranking with ties, $\sigma_3 = (1|2|4)$ denotes a partial ranking without ties, and $\sigma_4 = (1,2|4)$ denotes a partial ranking with ties.

Given a dataset composed of n rankings $\sigma_1, \sigma_2, \ldots, \sigma_n$ provided by a set of n agents, the rank aggregation problem (RAP) aims to identify the consensus permutation that best represents this dataset (Dwork et al. 2001). The consensus permutation is a permutation

in which its difference to the rankings of the dataset is minimal. The difference between the two rankings is usually measured by the distance. Among the distance measures available in the literature, the Kendall tau distance (or the Kendall distance) (Kendall 1938) is the most widely used in several real-world applications centered on the analysis of ranked data (Aledo et al. 2013, Zhou and Qiu 2018, Alfaro et al. 2021, Rodrigo et al. 2021).

The Kendall distance between two permutations (i.e., complete rankings) counts the total number of pairs of labels that are assigned to different relative orders in these two rankings. Formally, given two permutations π_u and π_v, the Kendall distance $d(\pi_u, \pi_v)$ can be defined as follows:

$$d(\pi_u, \pi_v) = |\{(i,j) : i < j, (\pi_u(\lambda_i) > \pi_u(\lambda_j) \wedge \pi_v(\lambda_i) < \pi_v(\lambda_j)) \vee (\pi_u(\lambda_i) < \pi_u(\lambda_j) \wedge \pi_v(\lambda_i) > \pi_v(\lambda_j))\}| \tag{1}$$

This is an intuitive and easily interpretable measure. For two permutations, the time complexity of computing d is $O(m\log(m))$.

Rankings can be incomplete. To calculate the distance between two arbitrary rankings (e.g., partial rankings) σ_u and σ_v, the extended Kendall distance $d'(\sigma_u, \sigma_v)$ counts the total number of label pairs over which the rankings disagree, ignoring the label pairs that are not ranked in both σ_u and σ_v. The extended Kendall distance d' is non-negative and symmetric, but the triangle inequality does not hold. When rankings σ_u and σ_v are permutations, d' agrees with d. As indicated in (Aledo et al. 2016), d' is not a pseudometric but it is suitable to provide a similarity measure for evaluating the difference between two arbitrary rankings. Obviously, the distance between two equal rankings is zero (i.e., $d' = 0$), but $d' = 0$ does not indicate that these two rankings are the same. In the four rankings mentioned above, $d'(\sigma_3, \sigma_4) = 0$, $d'(\sigma_2, \sigma_3) = 1$, and $d'(\sigma_1, \sigma_3) = 1$.

Given a set of arbitrary rankings $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ over m labels $\mathcal{L}_m = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, RAP aims to find the permutation π_o such that

$$\pi_o \leftarrow \arg\min_{\pi \in \Omega} \frac{1}{n} \sum_{i=1}^{n} d'(\pi, \sigma_i) \tag{2}$$

where Ω denotes the permutation space of $\{1, 2, \ldots, m\}$ and $d'(\pi, \sigma_i)$ denotes the extended Kendall distance between the two rankings π and σ_i. π_o is the consensus permutation that minimizes the sum of the total number of pairwise disagreements with respect to the given rankings.

Solving RAP is computationally challenging because it is known to be NP-hard when aggregating more than three rankings (Bartholdi et al. 1989). As the review presented in Section 2 indicates, even if several solution methods have been proposed for the problem, there is still room for improvement. Certainly, the best existing methods for RAP with complete rankings are time consuming for solving large RAP instances. For RAP with partial rankings, only local search algorithms have been proposed. To the best of our knowledge, a powerful population-based memetic approach (Neri and Cotta 2012) has not yet been studied for RAP with arbitrary rankings.

In this study, we develop an effective hybrid evolutionary ranking (HER) algorithm for solving the RAP (see Section 3). The proposed algorithm features two original and complementary search components: a concordant pair-based semantic crossover (CPSX for short) to construct meaningful offspring solutions, and an enhanced late acceptance hill climbing (ELAHC) procedure to find high-quality local optima. The main contributions of this study are summarized as follows.

From the perspective of algorithm design, the proposed CPSX operator is the first backbone-based crossover for RAP that relies on the identification and transmission of concordant pairs (building blocks) shared by the parent solutions. By inheriting meaningful building blocks, CPSX aims to generate promising offspring solutions that serve as starting points for local search optimization. Moreover, ELAHC reinforces the well-known late acceptance hill climbing (LAHC) heuristic (Burke and Bykov 2017) by exploring different high-quality solutions around each new offspring solution through the combined use of a relaxed acceptance and replacement strategy and an incremental evaluation mechanism introduced for RAP.

From the perspective of computational results, we present extensive experimental studies to demonstrate the high competitiveness of the proposed algorithms compared to state-of-the-art algorithms on both synthetic and real-world benchmark instances. In addition, we show the practical usefulness of this study for an important machine learning task known as label ranking.

The remainder of this paper is organized as follows: Section 2 provides a review of existing rank aggregation methods. Section 3 presents the proposed algorithm, followed by the computational results and comparisons in Section 4. The practical usefulness of the proposed method is illustrated in label ranking in Section 5. Experimental studies on the

key issues of the proposed algorithm are presented in Section 6. Section 7 summarizes the study's contributions.

2. Related Work on Rank Aggregation

Owing to the theoretical and practical significance of RAP, considerable effort has been devoted to the design of solution methods for this problem. These methods can be classified into two categories: exact algorithms and heuristic algorithms. Ali and Meilă (2012) performed an experimental analysis of many heuristics and exact algorithms for solving the RAP problem using data obtained from Mallows distributions following different parameterizations. Because RAP is an NP-hard problem (Bartholdi et al. 1989), exact algorithms are only practical for problem instances with a limited size. To handle large and difficult instances, several heuristic algorithms have been proposed to find approximate solutions.

The standard Borda method (Borda 1781) is a well-established greedy heuristic for RAP, which is intuitive and simple to compute for complete rankings. This method has the advantage of being simple and fast, but the solutions obtained may be far from the true optima. Aledo et al. (2013) used the genetic algorithm (GA) to solve the RAP problem with complete rankings (i.e., Kemeny ranking problem (KRP) (Yoo and Escobedo 2021)). Even though this algorithm only relies on standard permutation crossovers (position-based crossover, order crossover, order-based crossover) and mutations (insertion, displacement, and inversion), it obtained significantly better results than the most representative algorithms studied in (Ali and Meilă 2012). Aledo et al. (2018) further applied $(1+\lambda)$ evolution strategies (ES) to solve the optimal bucket order problem (OBOP), whose objective is to obtain a complete consensus ranking (where ties are allowed) from a matrix of preferences (also known as the precedence matrix). The authors experimentally evaluated several configurations of their ES algorithm.

To address RAP in the general setting, Aledo et al. (2016) proposed an improved Borda method for RAP containing arbitrary kinds of rankings and outperformed the standard Borda method. In addition, Nápoles et al. (2017) applied ant colony optimization to solve an extension of KRP, that is, the weighted KRP for partial rankings. DAmbrosio et al. (2017) proposed a differential evolution algorithm for consensus-ranking detection within Kemeny's axiomatic framework. To reduce the computational cost of the evaluation in RAP, Aledo et al. (2017b) proposed a partial evaluation method and integrated it into a

mutation-based metaheuristic algorithm. Recently, Aledo et al. (2019) performed a comparative study of four local search-based algorithms: hill climbing (HC), iterated local search (ILS), variable neighborhood search (VNS) and the greedy randomized adaptive search procedure (GRASP). Both the interchange and insert neighborhood are used in these local search algorithms. Empirical results show ILS achieves the best performance when a large number of fitness evaluations is allowed, although GRASP and VNS do not differ from ILS in terms of statistical significance.

3. Hybrid Evolutionary Ranking for Rank Aggregation Problem

In this section, we present the first hybrid evolutionary ranking (HER) algorithm for the rank aggregation problem with complete rankings. We begin with the solution representation and evaluation function, and then introduce the main components of the proposed algorithm. In Section 4.4, we explain how the algorithm can be easily adapted to the case of partial rankings by simply replacing the Kendall distance with the extended Kendall distance.

3.1. Solution Representation and Evaluation Function

Let $\mathcal{D} = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be a given dataset. A feasible candidate solution for the problem is a permutation π of the set $\{1, 2, \ldots, m\}$. The search space Ω is composed of all possible permutations of size n. For a given candidate solution π in Ω, the objective function value (fitness) is calculated as follows:

$$f(\pi) = \frac{1}{n} \sum_{k=1}^{n} d(\pi, \pi_k) \tag{3}$$

where $d(\pi, \pi_k)$ denotes the Kendall distance between π and π_k. Because calculating the Kendall distance $d(\pi, \pi_k)$ requires $O(m \log(m))$ time, the evaluation of a candidate solution requires $O(n \cdot m \log(m))$ time. The purpose of the HER algorithm is to find a permutation $\pi^* \in \Omega$ with the smallest objective function value $f(\pi^*)$.

3.2. General Framework

The HER algorithm adopts the memetic algorithm framework in discrete optimization (Neri and Cotta 2012, Zhou et al. 2023b,c) and combines a population-based approach with local optimization. As shown in Algorithm 1, HER is composed of four main components: a population initialization procedure, a concordance pairs-based semantic crossover (CPSX

for short), an enhanced late acceptance hill climbing (ELAHC) method, and a population updating strategy. The algorithm starts with a population of high-quality solutions. At each subsequent generation, a promising offspring solution is first generated by CPSX operator, then improved by ELAHC procedure, and finally considered for acceptance by the population updating strategy. The process is repeated until a stopping condition (i.e., the time limit t_{max} or maximum idle generation count $\hat{\zeta}$) is satisfied. We present each key procedure in the following sections.

Algorithm 1 Hybrid Evolutionary Ranking for RAP

Input: Problem instance I, population size μ, length of history cost list ρ, maximum idle iteration count $\hat{\zeta}$, and maximal idle generation count $\hat{\xi}$

Output: The best found solution π^*

$P \leftarrow$ **PopulationInitialization**(μ); //build an initial population
$\pi^* \leftarrow \arg\min_{\pi_i \in P} f(\pi_i)$; //record the best solution
$G_{idle} \leftarrow 0$;
while Stopping condition is not met **do**
  $\pi \leftarrow$ **CPSX**(P); //construct an offspring ranking based on CPSX operator
  $\pi' \leftarrow$ **ELAHC**($\pi, \rho, \hat{\zeta}$); //improve it through the ELAHC procedure
  if $f(\pi') < f(\pi^*)$ **then**
    $\pi^* \leftarrow \pi'$;
    $G_{idle} \leftarrow 0$;
  else
    $G_{idle} \leftarrow G_{idle} + 1$;
  end if
  if $G_{idle} > \hat{\xi}$ **then**
    break;
  end if
  $P \leftarrow$ **PopulationUpdating**(P, π'); //update the population
end while
return The best found solution π^*

3.3. Population Initialization

HER starts its search with a population of μ high-quality solutions, where each solution is obtained in two steps. First, an initial solution is obtained using a traditional Borda procedure. Then, the initial solution is further improved by ELAHC (see Section 3.5) before being added to the population.

The Borda procedure uses a well-established voting rule in social choice theory. We assume that the preferences of n voters are expressed in terms of rankings $\pi_1, \pi_2, \ldots, \pi_n$ over m alternatives. For each ranking π_i, the highest-ranked alternative receives m votes, the second highest receives $m-1$ votes, ..., and the lowest-ranked alternative receives only one vote, where m is the number of alternatives. The total score of an alternative is the sum of the votes that it has received from all n voters. Finally, a representative ranking is obtained based on the scores of the alternatives, where all alternatives are sorted in decreasing order of their scores, and the ties are broken at random. The Borda method is simple and terminates in $O(n \cdot m)$ time. To introduce randomness into the approach, which is helpful for effective exploration of the search space, we adopt a randomized Borda method that aggregates only $(1-\beta) \cdot n$ ($\beta \in (0, 0.5)$ is a randomized factor) rankings randomly selected from n rankings.

3.4. Concordant Pairs-based Semantic Crossover

As a driving force of hybrid evolutionary algorithms, a meaningful crossover operator should be able to generate promising offspring solutions that not only inherit the good properties of the parents but also introduce new useful characteristics (Pavai and Geetha 2016). The backbone concept has been widely used to define the good properties of parents. A variety of backbone-based crossovers have been proposed for subset selection problems, such as the maximum diversity problem (Zhou et al. 2017), the Steiner tree problem (Fu and Hao 2015), the critical node problem (Zhou et al. 2021, 2023c), and grouping problems such as the graph coloring (Galinier and Hao 1999) and the generalized quadratic multiple knapsack problem (Chen and Hao 2016). For the RAP problem whose solutions are permutations, we propose the first backbone-based crossover, named concordant pair-based semantic crossover (CPSX for short), which relies on the identification and transmission of concordant pairs (building blocks) shared by the parent solutions. By inheriting meaningful building blocks, crossover favors the generation of promising offspring solutions.

A ranking π of m labels can be equivalently transformed into a set of $m(m-1)/2$ pairwise preferences. For example, from $\pi = (4|2|1|3)$, we obtain a set of $4 \times (4-1)/2$ pairwise preferences $\{\lambda_4 \prec \lambda_2, \lambda_4 \prec \lambda_1, \lambda_4 \prec \lambda_3, \lambda_2 \prec \lambda_1, \lambda_2 \prec \lambda_3, \lambda_1 \prec \lambda_3\}$ where $\prec$ is the preference relation. Therefore, for any two or more rankings, their backbone can be defined as a set of concordant pairs (see Definition 1).

DEFINITION 1. (**Concordant pairs**). Given two rankings π_u and π_v of m labels, a pair of labels (λ_i, λ_j) is a concordant pair if labels λ_i and λ_j share the same preference relation $\lambda_i \prec \lambda_j$ or $\lambda_j \prec \lambda_i$ in the parent rankings.

Given two parent rankings π_u and π_v randomly selected from the population $\mathcal{P}$, the CPSX operator builds an offspring ranking π_o in four steps as follows. First, it decomposes each parent ranking $\pi_k, k \in \{u, v\}$ into a set of pairwise preference relations $\mathcal{R}_k$, and $|\mathcal{R}_k| = |\pi_k| \cdot (|\pi_k| - 1)/2$. Second, it identifies all concordant pairs (i.e., common preference relations between parent rankings), that is, $\mathcal{R}_o \leftarrow \mathcal{R}_u \bigcap \mathcal{R}_v$. Third, it combines the concordant pairs into a partial ranking according to a voting strategy. Specifically, each label λ_i receives $S(\lambda_i) = \sum_{\lambda_j \neq \lambda_i} \nabla_{ij}$ votes, where $\nabla_{ij} = 1$ if $\lambda_i \succ \lambda_j$ holds, otherwise $\nabla_{ij} = 0$. Then, a partial ranking is obtained by sorting all labels in descending order based on their votes. Finally, the partial ranking is repaired to form a permutation by determining all unknown preference relations in a random manner. The detailed pseudo code of the CPSX operator is provided in Algorithm 2.

Algorithm 2 Pseudo Code of the CPSX Operator

Input: Two parent rankings π_u and π_v

Output: The offspring ranking π_o

/* Step 1. decompose parent rankings into pairwise preference pairs */

decompose π_u into a set of pairwise preference relations $\mathcal{R}_u$;

decompose π_v into a set of pairwise preference relations $\mathcal{R}_v$;

/* Step 2. identify all concordant pairs */

$\mathcal{R}_o \leftarrow \mathcal{R}_u \cap \mathcal{R}_v$;

/* Step 3. combine concordant pairs into a partial ranking */

for $\forall i \in \{1, 2, \ldots, m\}$ **do**

    each alternative λ_i receives $S(\lambda_i) = \sum_{\lambda_j \neq \lambda_i} \nabla_{ij}$ votes, where $\nabla_{ij} = 1$ if $\lambda_i \succ \lambda_j$, otherwise $\nabla_{ij} = 0$;

end for

obtain a partial ranking π_o by ordering all alternatives in a descending order according to their votes;

/* Step 4. repair the partial ranking */

repair π_o by randomly determining all unknown preference relations;

return The offspring ranking π_o;

Figure 1 shows an illustrative example of the CPSX operator with two parent solutions: $\pi_1 = (1|3|4|5|2)$ and $\pi_2 = (1|5|3|4|2)$. **Step 1** decomposes the parent rankings into two sets of pairwise preference relation pairs: $\mathcal{R}_1 = \{\lambda_1 \prec \lambda_3, \lambda_1 \prec \lambda_4, \lambda_1 \prec \lambda_5, \lambda_1 \prec \lambda_2, \lambda_3 \prec \lambda_4, \lambda_3 \prec$

$\lambda_5, \lambda_3 \prec \lambda_2, \lambda_4 \prec \lambda_5, \lambda_4 \prec \lambda_2, \lambda_5 \prec \lambda_2\}$ and $\mathcal{R}_2 = \{\lambda_1 \prec \lambda_5, \lambda_1 \prec \lambda_3, \lambda_1 \prec \lambda_4, \lambda_1 \prec \lambda_2, \lambda_5 \prec \lambda_3, \lambda_5 \prec \lambda_4, \lambda_5 \prec \lambda_2, \lambda_3 \prec \lambda_4, \lambda_3 \prec \lambda_2, \lambda_4 \prec \lambda_2\}$. **Step 2** identifies all concordant pairs $\mathcal{R}_o = \{\lambda_1 \prec \lambda_2, \lambda_1 \prec \lambda_3, \lambda_1 \prec \lambda_4, \lambda_1 \prec \lambda_5, \lambda_3 \prec \lambda_2, \lambda_3 \prec \lambda_4, \lambda_4 \prec \lambda_2, \lambda_5 \prec \lambda_2\}$ between $\mathcal{R}_1$ and $\mathcal{R}_2$, which form the backbone of the parent rankings. **Step 3** combines the concordant pairs $\mathcal{R}_o$ into a partial ranking $\pi_0 = (1|3|4,5|2)$ according to a voting strategy. **Step 4** repairs π_o to obtain a complete ranking (i.e., a permutation) $\pi_0 = (1|3|5|4|2)$. Specifically, we determine the unknown preference relation between 4 and 5 in a random manner (in our example, $(5|4)$ is considered).

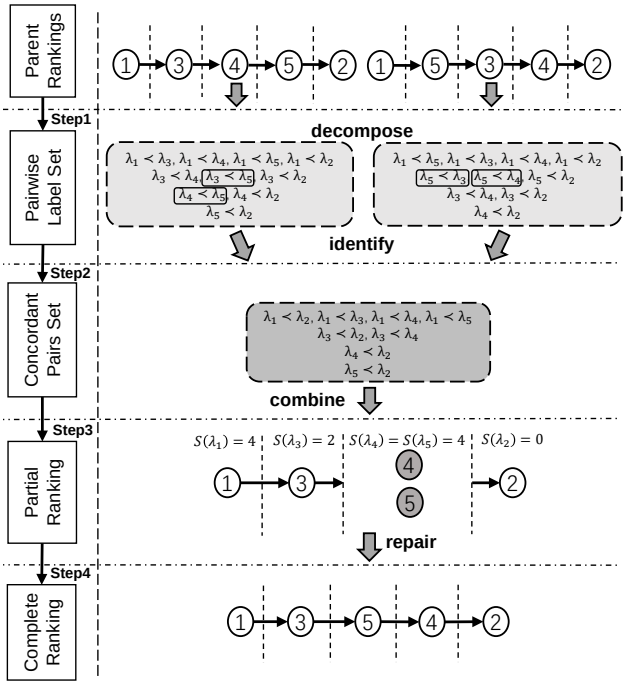

Figure 1 **Schematic Illustration of the CPSX Operator**

3.5. Enhanced Late Acceptance Hill Climbing

In addition to the CPSX operator, HER relies on a highly effective local optimization procedure named the enhanced late acceptance hill climbing (ELAHC) method, as shown in Algorithm 3. Our ELAHC procedure can be considered as an enhanced version of the well-known late acceptance hill climbing (LAHC) method (Burke and Bykov 2008, 2017). LAHC constitutes a hill climbing (HC) algorithm that employs a list of history costs of previously encountered solutions to decide whether to accept a new solution. The list of history costs is used as part of an acceptance criterion for any new solution encountered. If

a candidate solution has a better cost value than the least recent element of the list, then this solution is accepted. Correspondingly, the list is deterministically updated with cost values of new solutions. Since the cost values from previous iterations can be worse than those of the current solution, a candidate solution that is worse than the current solution can be accepted. This idea allows LAHC to avoid, to some extent, the local optimum problem of HC, which quickly converges to a bad locally optimal solution that is far from the global optimum. The use of the history cost list thus encourages search diversity. The larger the length of the history cost list ρ, the greater the diversity level. LAHC reduces to HC when the list only contains one cost value, i.e., $\rho = 1$. The pseudo code for our ELAHC procedure that replaces LAHC is as follows, with explanations of its components below.

Algorithm 3 Pseudo Code of the ELAHC Procedure

Input: Initial ranking π, length of history cost list ρ and maximal idle iteration count $\hat{\zeta}$
Output: The best ranking π^* found

$\pi^* \leftarrow \pi$, $f(\pi^*) \leftarrow f(\pi)$; //record the best ranking
$\forall i \in \{0, \ldots, \rho - 1\}$, $\phi[i] \leftarrow f(\pi)$;
$\phi_{max} \leftarrow f(\pi)$, $count \leftarrow \rho$;
initialize $I \leftarrow 0$, $I_{idle} \leftarrow 0$;
while $I_{idle} < \hat{\zeta}$ **do**
    $f_{prev} \leftarrow f(\pi)$;
    /* incremental evaluation */
    generate two different random integers $i, j \in \{1, 2, \ldots, n\}$;
    $\pi' \leftarrow \pi \oplus \mathbf{SWAP}(i, j)$; //generate a ranking based on SWAP operator
    $f(\pi') \leftarrow f(\pi) + \sum_{k=1}^{n} \Delta d(\pi, \pi', \pi_k, i, j)$;
    /* acceptance phase */
    if $f(\pi') < \phi_{max}$ or $f(\pi') = f(\pi)$ **then**
        $\pi \leftarrow \pi'$, $f(\pi) \leftarrow f(\pi')$; // accept the new ranking
    end if
    if $f(\pi) < f(\pi^*)$ **then**
        $\pi^* \leftarrow \pi$, $f(\pi^*) \leftarrow f(\pi)$; //update the best ranking
        $I_{idle} \leftarrow 0$;
    else
        $I_{idle} \leftarrow I_{idle} + 1$;
    end if
    /* replacement phase */
    $v \leftarrow I \mod \rho$; //calculate the virtual beginning
    if $f(\pi) < \phi[v]$ and $f(\pi) < f_{prev}$ **then**
        if $\phi[v] = \phi_{max}$ **then**
            $count \leftarrow count - 1$;
        end if
        $\phi[v] \leftarrow f(\pi)$; //update the history cost list
        if $count = 0$ **then**
            recompute $\phi_{max}, count$;
        end if
    else
        if $f(\pi) > \phi[v]$ **then**
            $\phi[v] \leftarrow f(\pi)$; //update the history cost list
        end if
    end if
    $I \leftarrow I + 1$;
end while
return The best ranking found π^*

Inspired by LAHC, ELAHC distinguishes itself from LAHC in two aspects: a relaxed acceptance and replacement strategy is applied to increase the diversity of the search, and an incremental evaluation mechanism is introduced to speed up the solution evaluation. We describe these two key components of ELAHC as follows.

3.5.1. Relaxed Acceptance and Replacement Strategy. The study by Franzin and Stützle (2018) shows a competitive performance of LAHC compared to eight other acceptance criteria, including simulated annealing, the great deluge algorithm and threshold acceptance. In addition, Burke and Bykov (2017) suggest that alternative methods can be used to update the history cost list of LAHC. Following this direction, we propose a relaxed acceptance and replacement strategy to increase the overall diversity of the search, which shares a similar idea with (Namazi et al. 2018). At the acceptance phase (lines 10-18), the diversity of the search is increased by employing a more relaxed acceptance strategy than LAHC. In particular, our acceptance strategy compares the cost function value $f(\pi')$ of the candidate ranking π' in each iteration I with the maximum cost value ϕ_{max} in the list ϕ instead of comparing it just with $\phi[v]$, where $v = I \mod \rho$. Our acceptance strategy accepts more new rankings during the search because a larger threshold is used in the comparison, i.e., $\phi_{max} \geq \phi[v]$. The replacement phase (lines 19-32) increases the diversity of cost values stored in the list by using a more relaxed replacement strategy than LAHC. A replacement occurs in ϕ if $f(\pi)$ is better than both $\phi(v)$ and the previous cost value f_{prev}, or if the cost value of the new current solution is worse than $\phi(v)$. Since ELAHC employs a relaxed acceptance and replacement strategy, more non-improving solutions are accepted during the search compared to LAHC, and ELAHC requires a smaller ρ value than LAHC.

3.5.2. Incremental Evaluation Mechanism. The complete evaluation of a neighboring ranking according to Equation (3) has a time complexity of $O(n \cdot m \log(m))$, which is extremely time-consuming. It is worth noting that existing algorithms for RAP suffer from the high computational complexity of calculating the Kendall distance. Typically, the objective function value of a candidate neighboring solution must be computed from scratch, which considerably slows the search process, particularly for large instances.

To overcome this problem, we propose an incremental evaluation mechanism to speed up the computation of the objective function for RAP (line 9). Given the Kendall distance

$d(\pi, \pi_k)$ between a candidate ranking π and a given ranking π_k, we assume π' constitutes a neighboring solution of π by performing a swap operation (SWAP for short) between two different positions i and j of π, i.e., $\pi' \leftarrow \pi \oplus \text{SWAP}(i, j), i \neq j \in \{1, \ldots, n\}$. Then, the Kendall distance $d(\pi', \pi_k)$ between π' and π_k can be incrementally calculated as follows:

$$d(\pi', \pi_k) = d(\pi, \pi_k) + \Delta d(\pi, \pi', \pi_k, i, j) \tag{4}$$

using the calculation of $\Delta d(\pi, \pi', \pi_k, i, j)$ described in Algorithm 4.

The incremental evaluation mechanism computes the objective function value of a neighboring ranking more efficiently as follows:

$$f(\pi') = f(\pi) + \frac{1}{n} \sum_{i=1}^{n} \Delta d(\pi, \pi', \pi_k, i, j) \tag{5}$$

This reduces the complexity from $O(n \cdot m \log(m))$ to $O(n \cdot m)$. Our incremental evaluation mechanism shares a similar idea with the partial evaluation proposed in (Aledo et al. 2017b). Both incremental evaluation and partial evaluation mechanisms consider the fact that a significant part of the objective function does not change after the application of standard mutation or neighborhood operators. Note that partial evaluation operates on a secondary structure (i.e., the pair order matrix) computed from the dataset, while our incremental evaluation evaluates a given ranking against the dataset directly.

3.6. Population Updating Strategy

Diversity is a property of a group of individuals that indicates the degree to which these individuals are different from each other. A suitable population updating strategy is necessary to maintain population diversity during the search, thus preventing the algorithm from premature convergence and stagnation (Neri and Cotta 2012). Diversity is often used to determine whether the offspring solution should be inserted into the population or discarded. In this study, we adopt a simple strategy that always replaces the worst individual if the offspring has a better solution quality and is different from any existing individual in the population.

3.7. Computational Complexity of HER

To analyze the computational complexity of the proposed HER algorithm, we consider the main procedures in one generation in the main loop of Algorithm 1. At each generation, the HER executes three procedures: CPSX, ELAHC and population updating. The CPSX

Algorithm 4 Pseudo Code of Incremental Evaluation for Calculating $\Delta d(\pi, \pi', \pi_k, i, j)$

Input: A given ranking π_k, a ranking π and its neighbor π' obtained by performing SWAP(i, j) on π

Output: The incremental distance $\Delta d(\pi, \pi', \pi_k, i, j)$

$count \leftarrow 0$;
if $\pi_k(\lambda_i) > \pi_k(\lambda_j)$ **then**
  SWAP(i, j);
end if
if $\pi(\lambda_i) > \pi(\lambda_j)$ **then**
  $count \leftarrow count + 1$;
else
  $count \leftarrow count - 1$;
end if
for $\forall temp \in (\pi_k(\lambda_i), \pi_k(\lambda_j))$ **do**
  $\lambda_v \leftarrow \arg \pi_k(\lambda_v) = temp$;
  if $(\pi(\lambda_i) > \pi(\lambda_v)$ and $\pi(\lambda_j) < \pi(\lambda_v))$ or $(\pi(\lambda_i) < \pi(\lambda_v)$ and $\pi(\lambda_j) > \pi(\lambda_v))$ **then**
    if $\pi(\lambda_i) > \pi(\lambda_v)$ **then**
      $count \leftarrow count - 1$;
    else
      $count \leftarrow count + 1$;
    end if
    if $\pi(\lambda_v) > \pi(\lambda_j)$ **then**
      $count \leftarrow count - 1$;
    else
      $count \leftarrow count + 1$;
    end if
  end if
end for
$\Delta d(\pi, \pi', \pi_k, i, j) \leftarrow count$;
return The incremental distance $\Delta d(\pi, \pi', \pi_k, i, j)$

operator can be performed in $O(m^2 + m\log(m) + m)$ time. The time complexity of the ELAHC procedure is $(\zeta \cdot (n \cdot m + \rho))$, where ζ denotes the total number of iterations executed in ELAHC and ρ denotes the length of the history cost list. The computational complexity for population updating is $O(\mu(m^2 + \mu))$, where μ is the population size. To summarize, the total computational complexity of the proposed HER for one generation is $O(m^2 + n \cdot m \cdot \zeta)$.

4. Computational Studies

In this section, we present a computational assessment of the HER algorithm and its ELAHC procedure. We first describe the benchmark instances and the experimental settings. Then, we present the computational results obtained on the benchmark instances and compare them with the state-of-the-art algorithms.

4.1. Benchmark Instances

Our studies are conducted on both synthetic and real-world instances.

- **Synthetic instances** were sampled from the Mallows distribution. To define a standard Mallows distribution, three parameters are required: the center permutation π_0, the spread parameter θ, and the length of the permutation m. In addition, the number of permutations to be sampled n is also needed to define a practical instance. For this category of instances, π_0 is always set to the identity permutation $\pi_0 = (1, 2, \ldots, m)$, $\theta \in \{0.001, 0.01, 0.1, 0.2\}$, $m \in \{50, 100, 150, 200, 250\}$, and $n = 100$. For each of the 20 combinations of θ and m, 20 instances with $n = 100$ permutations were generated. These instances were originally generated and used in (Aledo et al. 2013), which identifies the most complex instances to be those with a small θ and a large permutation size m.
- **Real-world instances** were obtained from practical applications: *Sushi* consists of 5000 responses to a questionnaire in which the participants have to rank flavors of sushi in order of preference, where $m = 100$ and $n = 5000$; *F1* is the set of the orders in which the 25 drivers finished at each one of the 20 Grand Prix celebrated at the Formula 1 driver championship during 2012, hence $m = 25$ and $n = 20$; *Tour* is based on the 2012 edition of the Tour of France, where each ranking contains the order in which the 153 cyclists that complete the Tour finished at each of the 20 stages. *ATPMen50*, *ATPMen100*, and *ATPMen200* are based on the ATP ranking of male tennis players along 2014, and *ATPWomen50*, *ATPWomen100*, and *ATPWomen200* are based on the ATP ranking of female tennis players along 2014. There are $n = 52$ rankings corresponding to the weeks of 2014.

4.2. Experimental Settings

Our algorithms were programmed in C++ and compiled using GNU gcc 4.1.2 with the '-O3' option on an Intel E5-2670 with 2.5GHz and 2GB RAM under the Linux OS. Please refer to Zhou et al. (2023a) for the instances, codes, and results of the experiments. We ran

each algorithm on each instance with a given time limit $\hat{t}$. Following (Aledo et al. 2013), we also set $\hat{\xi}$ to 60 as one of the stopping conditions. The detailed parameter settings of our algorithms are listed in Table 1. To determine the suitable parameter values, we employ the well-known automatic parameter configuration tool called IRACE (Vicente-López et al. 2016).

Table 1 Parameter Settings of Our Algorithms

Parameter	Description	Candidate Values	Final Value	p-value > 0.05?	Section
μ	Population Size	{10,15,20,25,30}	20	✓	Section 3.3
β	Randomized factor	{0.1,0.2,0.3,0.4,0.5}	0.2	✓	Section 3.3
$\hat{\zeta}$	Maximum Idle Iteration Count	{1000,5000,10000,15000,20000}	5000	✓	Section 3.5
ρ	Length of History Cost List	{1,5,10,15,20}	5	✓	Section 3.5

For each parameter, IRACE requires some candidate values as input, as shown in the column "Candidate Values" of Table 1. The best parameter configuration is provided in the column of "Final Value". During the parameter tuning, we run IRACE with the default settings, and set the total time budget at 2000 executions. The experiments are conducted on ten representative instances with different sizes selected from the benchmarks with a time limit of $\hat{t} = 3600$ seconds for solving each instance.

We apply the Friedman test (Demšar 2006) to further check whether there is a significant difference between each pair of candidate values in terms of HER performance. To evaluate the sensitivity of the parameters, we consider the "Candidate Values" for each parameter from Table 1 while fixing other parameters to their "Final Value". HER was run 30 times on each instance recording the average objective value. All p-values are larger than 0.05, which confirms that the parameters of HER exhibit no particular sensitivity at a significance level of 0.05.

4.3. Results for RAP with Complete Rankings

This section compares our HER algorithm and its ELAHC procedure with the following six state-of-the-art (SOTA) algorithms.

- **Borda** is a well-established greedy heuristic algorithm for RAP. Simple and fast, it can perform rank aggregation in linear time $O(nm)$ (Borda 1781).
- **CSS** is a graph-based approximate algorithm that implements a greedy version of the method introduced by Cohen et al. (1999).

- **DK** is an exact solver proposed by Davenport and Kalagnanam (2004), which the authors have subsequently enhanced with improved heuristics.
- **Branch and bound (B&B)** is an approximate version of the general branch-and-bound algorithm that makes a good tradeoff between memory requirements and solution quality (Aledo et al. 2013).
- **Genetic algorithm (GA)** is a population-based algorithm for estimating the consensus permutation of rank aggregation problems, which achieves SOTA results on instances from the Mallows model (Aledo et al. 2013).
- **Iterated local search (ILS)** is a multi-start local search algorithm based on the HC algorithm, which demonstrates the best performance when the algorithms are allowed to perform a large number of fitness evaluations (Aledo et al. 2019).

Aledo et al. (2013) experimentally compared the GA with the SOTA algorithms (i.e., Borda, CSS, DK, B&B), to obtain the following outcomes. The GA provides excellent performance, beating the CS and Borda algorithms in all cases. The DK and B&B methods are competitive with the GA only on less complex instances with large θ and small n values, being outperformed by the GA on instances with small θ values. It should be noted that Borda, CSS, and DK are greedy algorithms. They are considerably faster than the B&B and GA methods, but often produce poor results. The GA is far slower than the approximate version of B&B due to the large number of fitness evaluations required during the evolutionary search. In particular, the CPU time ratios between the GA and B&B are 9.6, 15.6, 219.4, and 639.9 on four extreme instances with $\theta \in \{0.001, 0.2\}$ and $m \in \{50, 250\}$. With the condition that the GA stops after 60 generations without improving the best solution, the GA achieved results matching the SOTA results on the benchmark instances. A further comparative study was carried out by Aledo et al. (2019) among different local search based metaheuristics (e.g., HC, ILS, VNS and GRASP) to deal with RAP. Comparative results show ILS has the best performance when the algorithms are allowed to perform a large number of fitness evaluations. Therefore, we use ILS as the reference algorithm in our experiments.

Since the original source code of ILS was written in Java and is not available to us, we have re-implemented it in the C++ programming language. The original ILS employs HC to perform local optimization, which exhibits a poor performance compared to the SOTA algorithms. Consequently, we implement an improved ILS using our ELAHC method

with $\rho = 1$ instead of HC to perform local optimization. Our improved ILS significantly outperforms the original ILS, as demonstrated by detailed comparative results summarized in an online supplement (Zhou et al. 2023a).

4.3.1. Results on Synthetic Instances: In our experiments, we solve each instance once and terminate the HER algorithm after 60 generations without improving the best solution or when the execution time reaches the time limit $\hat{t} = 2$ hours. Our stopping condition is much stricter than that of the GA. We then recorded the best result ($\hat{f}$), the average result ($\bar{f}$) and average computation time ($\bar{t}$) over each group of 20 instances. We also use the Wilcoxon signed-rank to compare two algorithms, as recommended in (Demšar 2006). The results of our algorithms and the SOTA algorithms are summarized in Table 2.

Table 2 Comparison of Our Algorithms and SOTA Algorithms on Synthetic Instances with Complete Rankings

Instance		Borda		CSS	DK	B&B	GA	ILS			ELAHC (this work)			HER (this work)		
θ	m	$\hat{f}$	t	$\hat{f}$	$\hat{f}$	$\hat{f}$	$\hat{f}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
0.200	50	187.837	0.001	188.342	187.816	187.815	187.815	**183.140**	**187.913**	723.482	**183.140**	187.914	2.633	**183.140**	**187.913**	3.662
0.100	50	320.194	0.001	320.883	320.128	320.104	320.104	**311.950**	**320.296**	30.213	**311.950**	320.304	2.446	**311.950**	**320.296**	4.767
0.010	50	559.915	0.001	560.720	559.582	558.928	558.769	**550.970**	559.797	333.231	551.030	559.755	2.035	**550.970**	**559.607**	434.461
0.001	50	569.701	0.001	570.499	569.546	568.662	568.469	561.740	569.968	1137.292	561.760	569.906	1.973	**561.480**	**569.718**	405.243
0.200	100	412.571	0.001	413.201	412.554	412.554	412.154	**405.140**	**411.884**	389.789	**405.140**	411.888	18.735	**405.140**	**411.884**	31.638
0.100	100	788.279	0.001	790.126	788.058	788.102	788.026	**776.020**	787.745	1190.498	776.060	787.779	16.392	**776.020**	**787.742**	89.533
0.010	100	2155.301	0.001	2157.450	2154.294	2152.986	2152.247	2126.890	2153.107	3401.334	2126.610	2152.856	15.056	**2126.390**	**2152.623**	3166.568
0.001	100	2308.277	0.001	2310.266	2308.038	2304.983	2303.231	2290.990	2306.096	3662.618	2290.570	2305.656	15.254	**2290.190**	**2305.296**	4277.235
0.200	150	637.245	0.002	638.903	637.177	637.177	637.176	**629.410**	**636.948**	152.134	629.430	636.956	54.732	**629.410**	**636.948**	308.377
0.100	150	1260.964	0.002	1264.171	1260.645	1260.610	1260.583	**1245.730**	1260.166	2464.411	1245.810	1260.194	45.819	**1245.730**	**1260.146**	619.723
0.010	150	4595.137	0.002	4599.431	4593.498	4590.917	4589.672	4533.430	4586.122	2082.753	4533.030	4585.319	118.519	**4532.710**	**4585.154**	4610.010
0.001	150	5206.998	0.002	5210.015	5208.340	5201.233	5196.731	5144.480	5191.362	3788.802	5143.660	5189.971	116.381	**5143.140**	**5189.620**	4126.092
0.200	200	862.707	0.003	865.154	862.650	862.685	862.648	**851.290**	**862.693**	115.287	851.330	862.704	120.458	**851.290**	**862.693**	1672.241
0.100	200	1734.810	0.003	1739.429	1734.336	1734.395	1734.303	1711.440	1734.231	2286.683	1711.400	1734.219	103.823	**1711.320**	**1734.181**	2263.225
0.010	200	7699.995	0.004	7706.323	7697.136	7694.639	7692.271	7625.320	7698.460	3830.479	**7624.120**	**7697.132**	280.606	7624.140	7697.138	3000.011
0.001	200	9250.210	0.003	9253.655	9256.021	9241.557	9232.840	9182.900	9240.152	3768.522	**9179.980**	9237.531	211.585	9180.140	**9237.178**	4051.452
0.200	250	1087.719	0.004	1090.796	1087.623	1087.654	1087.622	**1075.790**	**1087.684**	456.757	**1075.790**	1087.697	205.948	**1075.790**	**1087.684**	2917.056
0.100	250	2207.223	0.004	2213.130	2206.631	2206.665	2206.564	2186.420	2206.824	3190.976	2186.480	2206.800	187.542	**2186.260**	**2206.732**	3751.534
0.010	250	11311.189	0.004	11319.547	11307.085	11303.063	11300.249	11182.830	11301.168	3641.871	**11180.290**	**11299.402**	408.436	11180.350	11299.597	4170.573
0.001	250	14448.840	0.004	14451.179	14453.746	14435.551	14422.276	14337.260	14434.251	3097.111	**14333.660**	14430.421	422.379	14333.840	**14430.309**	4253.846
#Wins|Ties|Loses		20|0|0	–	20|0|0	20|0|0	20|0|0	20|0|0	11|9|0	14|6|0	–	12|4|4	18|0|2	–	–	–	–
p-value		**8.858e-5**	–	**8.858e-5**	**8.858e-5**	**8.858e-5**	**8.858e-5**	**9.766e-4**	**1.221e-4**	–	**2.970e-2**	**1.300e-3**	–	–	–	–

Notes. The results of each combination of θ and m are averaged over 20 instances. ILS employs ELAHC with $\rho = 1$ as local optimization instead of HC.

In Table 2, columns 1 and 2, describe θ and m values for each combination, respectively. Columns 3-4 present the results of the Borda method, i.e., the best result ($\hat{f}$) and computation time in seconds (t), while columns 5-8 list the best results ($\hat{f}$) of the four algorithms CSS, DK, B&B, and GA. Because their source codes are not available, we only list the results of these algorithms provided in (Aledo et al. 2013). Columns 9-11 list the results of ILS, including the best result ($\hat{f}$) over 20 instances, the average result ($\bar{f}$), and

the average time in seconds ($\bar{t}$) needed to obtain the best solution for each instance. Correspondingly, columns 12-14 and 15-17 list the results of ELAHC and HER, respectively. The best values for each performance indicator are highlighted in bold. In addition, we provide the number of combinations in which the HER method obtains better (#Wins), equal (#Ties), and worse (#Loses) results in terms of each indicator compared to the corresponding algorithms. At the end of Table 2, we also show the p-values of the Wilcoxon signed-rank test.

Table 2 indicates that our algorithms demonstrate excellent performances for all 20 combinations of θ and m. Note that the Borda method quickly converges to a poor solution. At a significance level of 0.05, our HER algorithm significantly outperforms the SOTA algorithms (Borda, CSS, DK, B&B, GA and ILS) in terms of $\hat{f}$. HER also exhibits significantly better performance than ILS (referring here to the improved ILS method noted above). Compared to ELAHC, the HER method performs better in terms of both $\hat{f}$ and $\bar{f}$ at a significance level of 0.05. It is also to be noted that ELAHC converges to its best local optimum in approximately 400s, whereas HER has a better long-term search ability by improving its results until about 4000s. These observations confirm the competitiveness of the proposed algorithms compared to the SOTA algorithms.

4.3.2. Results on Real-world Instances: Detailed results comparing our algorithms and the SOTA algorithms on real-world instances are summarized in Table 3. These results show that the HER method establishes the best performance in terms of both $\hat{f}$ and $\bar{f}$ on all instances except ATPWomen_200, for which HER achieves the second best performance in terms of $\hat{f}$ and $\bar{f}$, only slightly behind the performance of ELAHC. HER significantly outperforms the Borda method at a significance level of 0.05. HER also demonstrates excellent performance compared to ILS obtaining better results in terms of $\hat{f}$ than ILS on 2 out of 9 instances, and equal results on the remaining seven instances. For the indicator $\bar{f}$, the HER method obtains 3 better results and 6 equal results. However, there is no appreciable difference between HER and ILS at the significance level of 0.05. We also find HER significantly outperforms ELAHC in terms of $\bar{f}$. For the indicator $\hat{f}$, HER achieves better results than ELAHC on 2 instances and equal results on 6 remaining instances, but there is no significant performance difference between them relative to $\hat{f}$. In sum, we conclude that our algorithms are highly competitive compared to the SOTA algorithms on real-world instances.

Table 3 Comparison of Our Algorithms and SOTA Algorithms on Real-world Instances with Complete Rankings

		Borda		ILS			ELAHC (this work)			HER (this work)		
Instance	m	$\hat{f}$	t	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
Sushi	10	19.662	0.001	**19.181**	**19.181**	1.003	**19.181**	**19.181**	4.821	**19.181**	**19.181**	236.419
F1	25	69.500	0.001	**68.750**	**68.750**	0.197	**68.750**	**68.750**	0.495	**68.750**	**68.750**	2.615
Tour	153	3544.350	0.001	**3480.100**	3480.800	2938.547	3480.400	3480.750	341.234	**3480.100**	**3480.245**	3091.072
ATPWomen_50	50	234.500	0.001	**185.750**	**185.750**	4.264	**185.750**	185.754	24.186	**185.750**	**185.750**	40.328
ATPWomen_100	100	821.615	0.001	**715.019**	**715.019**	954.638	**715.019**	715.058	266.308	**715.019**	**715.019**	631.416
ATPWomen_200	200	3032.058	0.002	2823.423	2823.596	3467.266	**2823.077**	**2823.323**	2422.625	2823.192	2823.339	3946.326
ATPMen_50	50	244.135	0.001	**197.962**	**197.962**	7.795	**197.962**	198.019	22.575	**197.962**	**197.962**	43.341
ATPMen_100	100	960.654	0.001	**862.365**	**862.365**	1947.933	**862.365**	862.477	251.569	**862.365**	**862.365**	1716.577
ATPMen_200	200	3029.654	0.002	2810.519	2810.823	3543.888	2810.096	2810.408	2483.960	**2810.019**	**2810.352**	4632.466
#Wins|Ties|Loses		9|0|0	–	2|7|0	3|6|0	–	2|6|1	6|2|1	–	–	–	–
p-value		**3.900e-3**	–	5.000e-1	2.500e-1	–	7.500e-1	**4.690e-2**	–	–	–	–

Note. ILS employs ELAHC with $\rho = 1$ as local optimization instead of HC.

4.4. Results for RAP with Partial Rankings

To extend the HER algorithm to solve the RAP problem with partial rankings, the objective function must be modified. Given a dataset with partial rankings $\mathcal{D} = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, the objective function value of a candidate solution π is calculated as follows.

$$f(\pi) = \frac{1}{n} \sum_{k=1}^{n} d'(\pi, \sigma_k) \tag{6}$$

where $d'(\pi, \sigma_k)$ represents the extended Kendall distance between π and σ_k.

To demonstrate the effectiveness of our HER and ELAHC methods for solving RAP with partial rankings, we experimentally analyze them on benchmark instances and compare them with the extended Borda method, which operates as follows. Given a set of rankings $\sigma_1, \ldots, \sigma_n$, for each label λ_i in a partial ranking of only $m' < m$ labels, if the label is missing, then the label receives a Borda score of $s_{ij} = (m+1)/2$ votes, while if the label is an existing label with rank $r \in \{1, 2, \ldots, m'\}$, then its Borda score is $s_{ij} = (m' + 1 - r)(m + 1)(m' + 1)$. The average Borda score s_i is defined as $\frac{1}{n} \sum_{j=1}^{n} s_{ij}$. The labels are then sorted in the decreasing order of their average Borda scores. There are 20 instances for each combination of θ and m as well as the complete ranking data.

To transform a complete ranking into a partial ranking, we resorted to a simple transformation procedure. Given a complete ranking of n items, the transformation proceeds from the most to the least preferred item. When an item is visited it is discarded with a probability p_d. If the item is retained, then it stays in the current bucket with probability p_k; otherwise, it is randomly assigned to a new bucket. In our experiment, we select $p_d = \frac{2}{3}$ and $p_k = \frac{5}{6}$. Note that our transformation procedure follows the general practice for modeling partial ranking (Aledo et al. 2016, 2019). Detailed comparative results between our algorithms and the SOTA algorithms on synthetic and real-world instances are presented in Tables 4 and 5, respectively.

Table 4 Comparison of Our Algorithms and SOTA Algorithms on Synthetic Instances with Partial Rankings

Instance		Borda		ILS			ELAHC (this work)			HER (this work)		
θ	m	$\hat{f}$	t	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
0.200	50	110.146	0.001	159.160	169.834	3209.041	104.200	108.439	201.001	**104.010**	**108.253**	1474.880
0.100	50	161.129	0.001	190.280	198.107	4275.339	146.350	155.746	259.866	**145.710**	**155.283**	2006.546
0.010	50	260.357	0.001	240.060	252.980	3511.680	**215.040**	227.334	357.546	216.440	**225.363**	1477.692
0.001	50	271.714	0.001	246.350	254.774	4350.480	219.400	229.024	380.800	**216.920**	**227.225**	1598.063
0.200	100	322.204	0.001	760.380	783.084	2969.186	300.060	318.294	2072.679	**299.780**	**318.143**	1920.937
0.100	100	456.761	0.001	778.030	808.775	4166.726	**423.830**	**444.452**	2818.127	424.530	444.579	2064.247
0.010	100	993.393	0.001	1005.660	1028.769	3200.744	**865.670**	**891.501**	3517.686	875.050	897.143	1928.427
0.001	100	1087.184	0.001	1029.150	1049.141	4060.137	901.220	**918.750**	3504.064	**899.790**	923.105	1902.893
0.200	150	633.113	0.002	1784.320	1886.182	4632.456	606.310	629.422	3562.714	**603.630**	**626.871**	2143.940
0.100	150	847.017	0.002	1842.060	1916.826	3945.828	805.570	833.089	3561.592	**802.890**	**829.185**	2367.000
0.010	150	2119.302	0.002	2257.840	2310.586	4306.733	1950.840	1993.287	3561.519	**1924.550**	**1964.673**	2533.523
0.001	150	2439.313	0.002	2313.820	2391.442	2877.894	2089.050	2152.863	3552.684	**2051.680**	**2099.691**	2911.502
0.200	200	1038.027	0.002	3344.060	3512.498	4203.844	996.220	1051.096	3569.241	**973.340**	**1030.689**	2505.553
0.100	200	1322.746	0.003	3463.820	3536.286	4525.436	1273.340	1323.132	3572.179	**1251.500**	**1300.955**	2784.889
0.010	200	3573.712	0.003	4000.580	4100.070	4838.988	3381.430	3463.497	3568.342	**3298.180**	**3366.975**	3503.892
0.001	200	4330.083	0.002	4218.550	4287.149	4063.925	3877.530	3940.761	3559.041	**3704.640**	**3765.796**	3365.421
0.200	250	1512.232	0.003	5500.320	5616.749	3901.263	1524.780	1574.204	3576.292	**1459.210**	**1504.668**	3600.000
0.100	250	1887.351	0.003	5514.590	5668.163	3828.247	1855.640	1922.433	3575.967	**1796.670**	**1862.538**	3600.000
0.010	250	5281.381	0.003	6219.300	6371.876	4570.296	5074.930	5209.651	3534.476	**4924.100**	**5044.982**	3600.000
0.001	250	6735.437	0.004	6592.310	6723.613	3766.999	6157.210	6261.303	3519.012	**5870.410**	**5963.002**	3600.000
#Wins|Ties|Loses		20|0|0	–	20|0|0	20|0|0	–	17|0|3	17|0|3	–	–	–	–
p-value		**8.858e-5**	–	**8.858e-5**	**8.858e-5**	–	**1.300e-3**	**1.500e-3**	–	–	–	–

Notes. The results of each combination of θ and m are averaged over 20 instances. ILS employs ELAHC with $\rho = 1$ as local optimization instead of HC.

4.4.1. Results on Synthetic Instances: Table 4 describes the comparative results between our algorithms and the SOTA algorithms on synthetic instances. From this table, we observe that the Borda method quickly converges to a bad solution, and our algorithms (i.e., ELAHC and HER) exhibit excellent performances on instances with partial rankings, significantly outperforming the Borda method for all 20 combinations in terms of both $\hat{f}$ and $\bar{f}$. The average results of the ELAHC and HER methods are also better than those achieved by the Borda method. At a significance level of 0.05, we obtain the same conclusion that our algorithms significantly outperform ILS in terms of both $\hat{f}$ and $\bar{f}$. The worse performance of ILS compared to ELAHC and HER may be explained by the fact that ILS starts from a random solution and uses a weak local search procedure to perform local optimization during the search. Regarding a comparison between the HER and ELAHC methods, it is not surprising to observe that HER outperforms ELAHC in terms of $\hat{f}$ and $\bar{f}$. This experiment demonstrates the effectiveness of both HER and ELAHC for solving the RAP problem with partial rankings.

4.4.2. Results on Real-world Instances: We further compare our algorithms with the SOTA algorithms on real-world instances with partial rankings. For each real-world problem, we generate 20 new instances with partial ranking in the same way as the synthetic instances. Detailed comparative results are summarized in Table 5, which shows that our HER algorithm performs significantly better than the SOTA algorithms (Borda and ILS)

Table 5 Comparison of Our Algorithms and SOTA Algorithms on Real-world Instances with Partial Rankings

		Borda		ILS			ELAHC (this work)			HER (this work)		
Instance	m	$\hat{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
Sushi	10	8.840	0.006	**5.782**	**5.849**	1369.078	**5.782**	**5.849**	0.147	8.689	8.754	197.310
F1	25	33.950	0.001	30.200	33.850	1812.103	**25.500**	**29.130**	1106.626	27.300	29.980	8.882
Tour	153	1647.500	0.001	2075.550	2169.660	3386.329	1554.200	1638.030	3051.146	**1313.900**	**1387.060**	3865.272
ATPWomen_50	50	104.654	0.001	156.173	162.412	1199.129	**98.789**	**102.969**	1043.898	103.615	108.150	1699.703
ATPWomen_100	100	418.404	0.001	761.942	793.204	1394.210	408.481	**426.004**	3115.755	**408.154**	426.439	2625.538
ATPWomen_200	200	1676.577	0.001	3518.327	3551.535	1206.199	1652.346	1682.508	2453.833	**1636.789**	**1662.823**	4651.294
ATPMen_50	50	112.558	0.001	161.385	166.827	1488.772	**107.039**	111.562	1238.126	111.346	**115.246**	1803.865
ATPMen_100	100	479.039	0.001	755.423	801.131	1270.339	463.539	477.792	1908.376	**462.654**	**475.585**	2137.048
ATPMen_200	200	1703.731	0.001	3501.308	3557.277	1571.770	1688.250	1719.808	2712.962	**1662.558**	**1696.846**	3513.714
#Wins|Ties|Loses		9|0|0	–	8|0|1	8|0|1	–	5|0|4	5|0|4	–	–	–	–
p-value		**3.900e-3**	–	**1.170e-2**	**7.800e-3**	–	6.523e-1	6.523e-1	–	–	–	–

Notes. The results of each type of instance are averaged over 20 instances. ILS employs ELAHC with $\rho = 1$ as local optimization instead of HC.

both in terms of $\hat{f}$ and $\bar{f}$ at a significance level of 0.05. Compared to ELAHC, we observe that the HER algorithm finds better results in terms of both $\hat{f}$ and $\hat{f}$ on five instances with $m \geq 100$. For three instances with $m < 100$, ELAHC shows better performance than HER.

5. Application to Label Ranking

To more fully demonstrate the practical interest of our proposed ranking aggregation method, we present its application to the well-known label ranking problem.

5.1. Label Ranking

Label ranking (Hüllermeier et al. 2008, Cheng et al. 2010, Zhou et al. 2014, Zhou and Qiu 2018, Alfaro et al. 2021) is an important machine learning task that seeks a mapping between an instance and a ranking of labels from a finite set, for the goal of representing the relevance of the ranking to the instance considered. Label ranking extends traditional classification and multi-label classification in being required to predict the ranking of all class labels rather than only one or several labels. The problem emerges naturally in applications such as drug design, recommendation systems, image recognition, text classification, and meta-learning (Hüllermeier et al. 2008, Adomavicius and Zhang 2016, de Sá et al. 2017).

Due to its wide applicability, label ranking has recently attracted considerable attention from the machine learning community (Har-Peled et al. 2003, Hüllermeier et al. 2008, Cheng et al. 2009, 2010, de Sá et al. 2017, Aledo et al. 2017a, Negahban et al. 2017, Zhou and Qiu 2018, Werbin-Ofir et al. 2019, Dery and Shmueli 2020, Alfaro et al. 2021, Fotakis et al. 2022). Existing label ranking algorithms can be divided into three categories: 1) Decomposition approaches that transform a label ranking problem into several binary classification problems whose outcomes are combined to produce output rankings,

as in constraint classification (Har-Peled et al. 2003) and ranking by pairwise comparison (Hüllermeier et al. 2008); 2) Probabilistic approaches that perform label ranking based on statistical models for ranking data, such as instance-based learning algorithms with Mallows models (Cheng et al. 2009) and Plackett-Luce models (Cheng et al. 2010); and 3) Ensemble approaches that aim to improve the accuracy of model outcomes by combining multiple models instead of using a single model, as represented by bagging (Aledo et al. 2017a, de Sá et al. 2017, Zhou and Qiu 2018), boosting (Dery and Shmueli 2020), and voting rules (Werbin-Ofir et al. 2019). Compared with decomposition and probabilistic approaches, ensemble approaches have achieved SOTA performance.

Rank aggregation plays a key role in label ranking in that the performance of a label ranking algorithm depends greatly on the results of the rank aggregation. Commonly, a set of rankings is aggregated by the weak Borda heuristic (Borda 1781) and a more powerful rank aggregation heuristic would offer the potential to improve the existing label ranking algorithms. To show the relevance of the methods proposed here, we consider the enhanced late acceptance hill climbing (ELAHC) method as an example, and integrate it into the label ranking forest (de Sá et al. 2017).

5.2. Label Ranking Forest

The Label Ranking Forest (LRF) (de Sá et al. 2017) is an ensemble approach that has been highly successful in application to many datasets, whose source code is publicly available at the GitHub website[1]. We undertook the challenge of seeing if we could enhance this approach using our ELAHC procedure.

Figure 2 presents the framework of LRF method. During the prediction phase, a dataset used as a test sample is passed through all K trees simultaneously (starting at the root node) until it reaches the leaf nodes. Each decision tree T_i generates a predicted ranking π_i by aggregating the target rankings (RA_{1st} for short) of the training examples stored in a leaf node, i.e., $\{\pi_{i1}, \pi_{i2}, \ldots, \pi_{ir_i}\}$. The resulting K predicted rankings, i.e., $\pi_1, \pi_2, \ldots, \pi_K$ are aggregated into a final predicted ranking (RA_{2nd} for short). LRF therefore performs K tasks of producing rankings of the RA_{1st} type together with a final RA_{2nd} rank aggregation task.

Both RA_{1st} and RA_{2nd} rank aggregation tasks can be solved by a heuristic algorithm (e.g., HER, ELAHC and Borda). To explore the usefulness of ELAHC to enhance the

[1] https://github.com/rebelosa/labelrankingforests

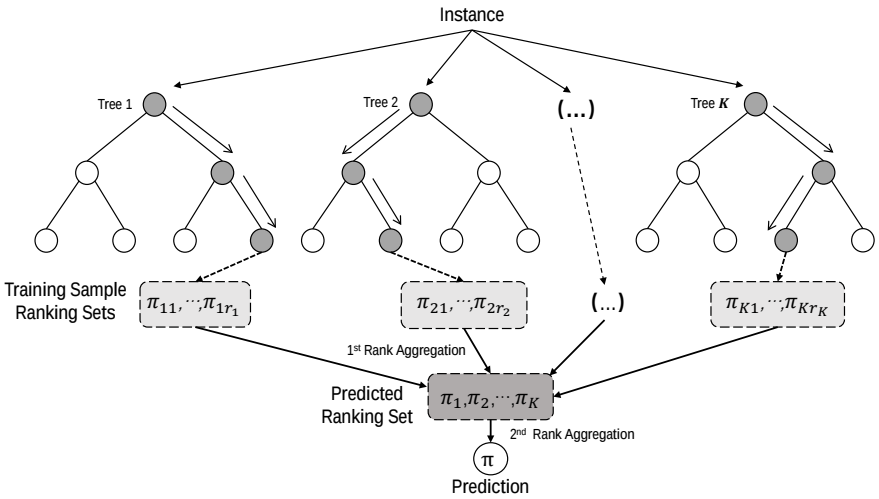

Figure 2 Framework of Label Ranking Forest

standard LRF approach, we experimentally compared LRF with three variants: 1) LRF_{10} is obtained from LRF by only performing the K RA_{1st} tasks with ELAHC; 2) LRF_{01} modifies LRF by only performing the RA_{2nd} task with ELAHC; and 3) LRF_{11} modifies LRF by performing the rank aggregation of both the RA_{1st} tasks and RA_{2nd}) tasks with ELAHC. Note that the HER method can similarly be applied to enhance LRF. However, HER is a time-consuming population-based algorithm, and therefore we focused on using the faster ELAHC method to perform rank aggregation in both the training phase and predicting phase.

5.3. Computational Results

Our experiments are conducted on 21 publicly available datasets consisting of 16 semi-synthetic and 5 real-world datasets[2]. Following general practice (Hüllermeier et al. 2008, Cheng et al. 2010, Zhou and Qiu 2018), we use Kendall's tau coefficient (Kendall 1938) to evaluate the performance of the label ranking algorithms tested. We applied LRF to generate $K = 100$ decision trees to provide the default parameters in our experiments. All results were obtained based on a four-fold cross validation.

5.3.1. Results on Semi-synthetic Datsets. Table 6 summarizes the comparative results of LRF and its three variants on the sixteen semi-synthetic datasets which were derived from multiclass datasets in the UCI Repository of machine learning databases and the Statlog collection (Hüllermeier et al. 2008). Values in bold identify results better than

[2] https://en.cs.uni-paderborn.de/de/is/research/research-projects/software/label-ranking-datasets

LRF, where larger values identify better results. The bottom row of the table also provides the average rank of each algorithm for all datasets. We rank the algorithms for each dataset separately, the best performing algorithm getting the rank of 1, the second best rank 2,. . . , and so on. In case of ties, average ranks are assigned. Finally, the average rank is obtained by averaging all the ranks of each algorithm on all datasets. For the indicator of average rank, the smaller the value, the better the algorithm.

Table 6 Comparison of LRF Algorithms with Different Rank Aggregation Strategies on Semi-synthetic Datasets

Dataset	Instances	Features	Labels	LRF	LRF_{10}	LRF_{01}	LRF_{11}
Authorship	841	70	4	0.892	**0.893**	0.892	0.892
Bodyfat	252	7	7	0.203	0.200	**0.206**	**0.207**
Calhousing	20640	4	4	0.185	0.185	0.182	0.169
Cpu-small	8192	6	4	0.485	**0.490**	0.479	**0.487**
Elevators	16599	9	9	0.750	**0.752**	0.748	**0.756**
Fried	40760	9	5	0.856	0.855	**0.862**	**0.867**
Glass	214	9	6	0.885	**0.893**	**0.887**	**0.894**
Housing	506	6	6	0.804	**0.809**	**0.807**	**0.811**
Iris	150	4	3	0.956	0.956	**0.959**	**0.960**
Pendigits	10992	16	10	0.884	**0.885**	**0.890**	**0.906**
Segment	2310	18	7	0.941	**0.942**	0.940	**0.945**
Stock	950	5	5	0.905	0.905	**0.906**	**0.910**
Vehicle	846	18	4	0.860	**0.861**	0.860	**0.862**
Vowel	528	10	11	0.861	0.861	**0.862**	**0.864**
Wine	178	13	3	0.902	0.901	**0.905**	**0.906**
Winsconsin	194	16	16	0.860	**0.861**	0.860	**0.862**
avg. rank	–	–	–	3.250	2.563	2.813	**1.375**

From Table 6, we observe that all three variants of LRF obtain smaller average ranks than LRF, indicating that ELAHC can significantly improve LRF. Specifically, LRF_{10} achieves better results on 9 out of 16 tested datasets, the same results on 4 datasets, and slightly worse results on 3 datasets. (Only strictly better results are highlighted in bold.) LRF_{01} obtains better results on 9 out of 16 tested datasets, and the same result on 3 datasets. LRF_{11} achieves better results on 14 out of 16 tested datasets, the same result on 1 dataset, and a worse result on 1 dataset. This experiment demonstrates the interest of using the ELAHC algorithm to enhance the well-established LRF algorithm.

5.3.2. Results on Real-world Datasets. Table 7 lists the comparative results of LRF and its three variants on 5 real-world datasets from the bioinformatics fields. These datasets are obtained from 5 microarray experiments (cold, diau, dtt, heat, spo) (Hüllermeier et al. 2008). This table shows that ELAHC can significantly enhance LRF for these real-world datasets too. Each of the three variants of LRF using the ELAHC method outperforms the original LRF in terms of the average rank. In particular, LRF_{11} obtains the smallest

average rank 1.000, LRF_{01} obtains the second-best average rank 2.100, and LRF_{10} obtains the third-best average rank of 3.400, while the average rank of LRF is 3.500.

Table 7 Comparison of LRF Algorithms with Different Rank Aggregation Strategies on Real-world Datasets

Dataset	Instances	Features	Labels	LRF	LRF_{10}	LRF_{01}	LRF_{11}
cold	2465	24	4	0.076	**0.078**	**0.081**	**0.086**
diau	2465	24	7	0.229	0.228	**0.230**	**0.232**
dtt	2465	24	4	0.114	0.110	**0.118**	**0.121**
heat	2465	24	6	0.028	**0.029**	**0.029**	**0.030**
spo	2465	24	11	0.145	0.145	**0.152**	**0.156**
avg. rank	–	–	–	3.500	3.400	2.100	**1.000**

6. Analysis and Discussion

This section presents additional experiments to gain a deeper understanding of the HER and ELAHC methods. We perform two groups of experiments: 1) to demonstrate the superiority of the ELAHC procedure, and 2) to evaluate the effectiveness of the CPSX operator. The following experiments were conducted on 10 representative instances, where each instance is selected based on its θ and m values.

6.1. Assessment of Enhanced Late Acceptance Hill Climbing

Recall that ELAHC is designed to reinforce the well-known late acceptance hill climbing (LAHC) heuristic (Burke and Bykov 2017) by introducing a relaxed acceptance and replacement strategy and a fast incremental evaluation mechanism. Consequently, it is of interest to experimentally compare ELAHC with LAHC. For each tested instance, we execute both LAHC and ELAHC 10 times with different random seeds under the same time limit $\hat{t} = 600$ seconds, and then record the best solution value ($\hat{f}$) found during 10 runs, the average solution value ($\bar{f}$), and the average computation time in seconds ($\bar{t}$) needed to achieve the best solution value at each run. In this experiment, the lengths of history cost list ρ of LAHC and ELAHC are $\rho = 3000$ and $\rho = 5$, respectively. They are experimentally determined based on the parameter sensitivity analysis, as shown in the online supplement (Zhou et al. 2023a).

Figure 3 shows the comparative performances of ELAHC and LAHC in terms of the best solution value, average solution value and average computation time, where the x-axis presents instances, and the y-axis denotes the corresponding performance, and Figure 3 (a) and (b) demonstrate the performance gaps. By treating LAHC as a baseline algorithm, we

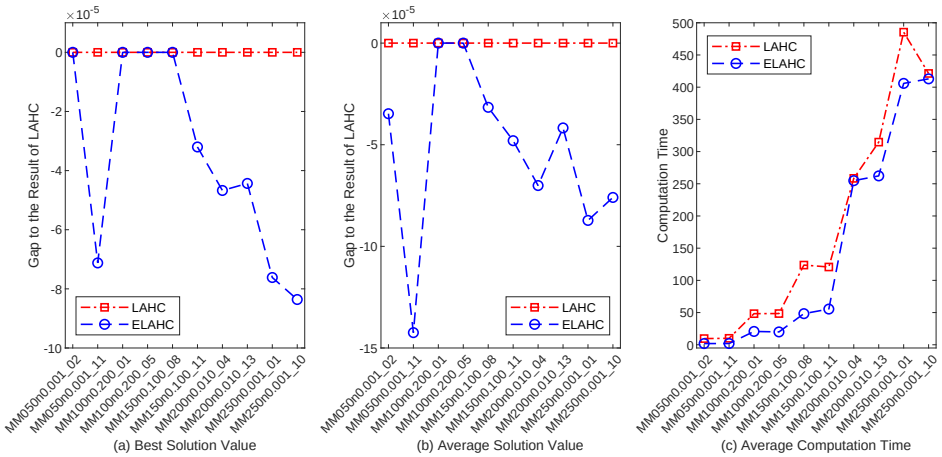

Figure 3 **Comparison between ELAHC and LAHC in terms of (a) Best Solution Value, (b) Average Solution Value, and (c) Average Computation Time**

calculate the performance gap as $(f - f')/f'$, where f' is the result of LAHC (i.e., baseline algorithm) and f is the result of ELAHC. A performance gap smaller than zero indicates that ELAHC obtains a better result on the corresponding instance. From 3(a), we observe that ELAHC performs better than LAHC in terms of the best solution value ($\hat{f}$), obtaining better results in 6 instances, and the same results in the remaining 4 instances. In terms of average solution value ($\bar{f}$), ELAHC shows a much better performance than LAHC with improved results in 8 instances and the same results in the 2 remaining instances. In terms of average computation time ($\bar{t}$), Figure 3(c) shows that ELAHC uses less computation time than LAHC for all 10 instances. The experiment confirms that ELAHC outperforms LAHC on the problem addressed in this paper.

6.2. Effectiveness of Concordant Pairs-based Semantic Crossover

To evaluate the effectiveness of the concordant pairs-based semantic crossover (CPSX), we experimentally compare HER with its three variants, namely HER′, HER″, and HER‴, where CPSX is replaced by three popular permutation crossover operators (Pavai and Geetha 2016): order crossover (OX), order-based crossover (OBX), and position-based crossover (PBX).

Table 8 summarizes the comparative results of the HER and its three variants on the 10 selected instances, reporting the best result $\hat{f}$ and the average result $\bar{f}$ of each algorithm over ten runs. We also list the average value, and the average rank of each performance indicator in the two bottom rows of the table. We observe that HER outperforms all three

Table 8 Comparison of HER Algorithms with Different Crossover Operators (OX, OBX, PBX and CPSX)

	HER′(with OX)		HER″ (with OBX)		HER‴ (with PBX)		HER (with CPSX)	
Instance	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$
MM050n0.001_02	**575.480**	575.516	**575.480**	575.534	**575.480**	575.506	**575.480**	**575.502**
MM050n0.001_11	**561.480**	**561.480**	**561.480**	561.486	**561.480**	**561.480**	**561.480**	**561.480**
MM100n0.200_01	**416.000**	**416.000**	**416.000**	**416.000**	**416.000**	**416.000**	**416.000**	**416.000**
MM100n0.200_05	**411.720**	**411.720**	**411.720**	**411.720**	**411.720**	**411.720**	**411.720**	**411.720**
MM150n0.100_08	**1249.580**	**1249.598**	**1249.580**	1249.622	**1249.580**	**1249.598**	**1249.580**	**1249.598**
MM150n0.100_17	**1266.090**	**1266.100**	**1266.090**	1266.114	**1266.090**	**1266.100**	**1266.090**	**1266.100**
MM200n0.010_04	**7667.850**	**7668.052**	7668.030	7668.366	**7667.850**	7668.056	7667.910	7668.054
MM200n0.010_13	7704.840	7704.978	7704.920	7705.284	**7704.800**	**7704.938**	**7704.800**	7705.060
MM250n0.001_01	**14353.330**	14353.888	14353.870	14354.682	14353.490	14353.912	**14353.330**	**14353.650**
MM250n0.001_10	14442.240	14442.674	14442.340	14443.086	14442.540	14442.964	**14442.140**	**14442.604**
avg. value	4864.861	4865.001	4864.951	4865.189	4864.903	4865.027	**4864.853**	**4864.977**
avg. rank	2.300	2.100	3.000	3.700	2.500	2.300	**2.200**	**1.900**

variants, achieving a better average value and average rank in terms of both $\hat{f}$ and $\bar{f}$, confirming the effectiveness of CPSX used in HER.

7. Concluding Remarks

Our hybrid evolutionary ranking algorithm for solving the challenging rank aggregation problem with both complete and partial rankings integrates two distinguishing components that underlie its effectiveness. To generate promising offspring rankings, the algorithm uses a problem-specific crossover based on concordant pairs of two parent rankings. Supplementing this, the algorithm introduces an enhanced late acceptance hill climbing procedure to perform local optimization, which reinforces the well-known late acceptance hill climbing heuristic by a relaxed acceptance and replacement strategy and a fast incremental evaluation mechanism. Empirical results on synthetic and real-world benchmark instances of both complete and partial rankings show that the algorithm performs significantly better than state-of-the-art methods.

To further demonstrate the usefulness of the proposed method for practical problems, we applied our method to label ranking, which is a relevant task in machine learning. The computational outcomes demonstrate that our proposed method can benefit existing label ranking algorithms by generating better rank aggregations. We also perform three groups of experiments to verify the effectiveness of the method's key algorithmic components.

Several avenues exist for future research. First, it would be interesting to test the proposed method for other applications. The codes of the proposed algorithms are made publicly available to facilitate such applications. Second, our concordant pairs-based semantic crossover for permutation encoding enriches the pool of existing permutation crossovers.

We envision that this crossover may find uses in applications where an order relation among a permutation of elements is relevant. Third, we plan to experimentally compare existing label ranking algorithms that are enhanced by our advanced rank aggregation methods. Considerable effort has been devoted to using machine learning techniques to improve optimization methods in recent years. As a complement to this, the present work may be viewed as a contribution to research on the use of optimization methods to solve machine learning problems more efficiently.

Acknowledgments

The authors are grateful to the editors and the reviewers for their insightful comments and suggestions which have helped us to improve the paper considerably. The authors would like to express their special thanks to Dr Mosab Bazargani, Bangor University, UK, whose comments clearly contributed to the improvement and correctness of this paper. The authors also would like to thank Zhihao Wu and Chenhui Qu for helping to execute some experiments. This work was partially supported by the National Natural Science Foundation of China under Grant No. 72371157.

References

Adomavicius G, Zhang J (2016) Classification, ranking, and top-k stability of recommendation algorithms. *INFORMS J. Comput.* 28(1):129–147.

Aledo JA, Gámez JA, Molina D (2013) Tackling the rank aggregation problem with evolutionary algorithms. *Appl. Math. Comput.* 222:632–644.

Aledo JA, Gámez JA, Molina D (2016) Using extension sets to aggregate partial rankings in a flexible setting. *Appl. Math. Comput.* 290:208–223.

Aledo JA, Gámez JA, Molina D (2017a) Tackling the supervised label ranking problem by bagging weak learners. *Inf. Fusion* 35:38–50.

Aledo JA, Gámez JA, Molina D (2019) Approaching the rank aggregation problem by local search-based metaheuristics. *J. Comput. Appl. Math.* 354:445–456.

Aledo JA, Gámez JA, Rosete A (2017b) Partial evaluation in rank aggregation problems. *Comput. Oper. Res.* 78:299–304.

Aledo JA, Gámez JA, Rosete A (2018) Approaching rank aggregation problems by using evolution strategies: the case of the optimal bucket order problem. *Eur. J. Oper. Res.* 270(3):982–998.

Alfaro JC, Aledo JA, Gámez JA (2021) Learning decision trees for the partial label ranking problem. *Int. J. Intell. Syst.* 36(2):890–918.

Ali A, Meilă M (2012) Experiments with kemeny ranking: What works when? *Math. Soc. Sci.* 64(1):28–40.

Bartholdi J, Tovey CA, Trick MA (1989) Voting schemes for which it can be difficult to tell who won the election. *Soc. Choi. Welfa* 6(2):157–165.

Borda J (1781) Memoire sur les elections au scrutin. *Histoire de l' Academie des Sciences, Paris* 657–665.

Burke EK, Bykov Y (2008) A late acceptance strategy in hill-climbing for examination timetabling problems. *Proc. 7th Int. Conf. PATAT 2008*, 1–7.

Burke EK, Bykov Y (2017) The late acceptance hill-climbing heuristic. *Eur. J. Oper. Res.* 258(1):70–78.

Chen Y, Hao JK (2016) Memetic search for the generalized quadratic multiple knapsack problem. *IEEE Trans. Evol. Comput.* 20(6):908–923.

Cheng W, Dembczynski K, Hüllermeier E (2010) Label ranking methods based on the plackett-luce model. *Proc. 27th Int. Conf. on Mach. Learn.*, 215–222.

Cheng W, Hühn J, Hüllermeier E (2009) Decision tree and instance-based learning for label ranking. *Proc. 26th Int. Conf. on Mach. Learn.*, 161–168.

Cohen WW, Schapire RE, Singer Y (1999) Learning to order things. *J. Artif. Intell. Res.* 10:243–270.

DAmbrosio A, Mazzeo G, Iorio C, Siciliano R (2017) A differential evolution algorithm for finding the median ranking under the kemeny axiomatic approach. *Comput. Oper. Res.* 82:126–138.

Davenport A, Kalagnanam J (2004) A computational study of the kemeny rule for preference aggregation. *Proc. 19th Nat. Conf. Artif. Intell.*, volume 4, 697–702.

de Sá CR, Soares C, Knobbe A, Cortez P (2017) Label ranking forests. *Expert Syst.* 34(1):e12166.

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* 7(Jan):1–30.

Dery L, Shmueli E (2020) BoostLR: A boosting-based learning ensemble for label ranking tasks. *IEEE Access* 8:176023–176032.

Destercke S, Masson M, Poss M (2015) Cautious label ranking with label-wise decomposition. *Eur. J. Oper. Res.* 246(3):927–935.

Dwork C, Kumar R, Naor M, Sivakumar D (2001) Rank aggregation methods for the web. *Proc. 10th Int. World Wide Web Conf.*, 613–622.

Fotakis D, Kalavasis A, Psaroudaki E (2022) Label ranking through nonparametric regression. *Proc. 39th Int. Conf. on Mach. Learn.*, 6622–6659.

Franzin A, Stützle T (2018) Comparison of acceptance criteria in randomized local searches. Lutton E, Legrand P, Parrend P, Monmarché N, Schoenauer M, eds., *Artificial Evolution*, 16–29 (Springer International Publishing).

Fu ZH, Hao JK (2015) Dynamic programming driven memetic search for the steiner tree problem with revenues, budget, and hop constraints. *INFORMS J. Comput.* 27(2):221–237.

Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. *J. Comb. Optim.* 3(4):379–397.

Gharaei A, Jolai F (2021) An ERNSGA-III algorithm for the production and distribution planning problem in the multiagent supply chain. *Int. Trans. Oper. Res.* 28(4):2139–2168.

Har-Peled S, Roth D, Zimak D (2003) Constraint classification for multiclass classification and ranking. *Adv. Neur. Inform. Process. Syst.* 809–816.

Huang Z, Zeng DD (2011) Why does collaborative filtering work? transaction-based recommendation model validation and selection by analyzing bipartite random graphs. *INFORMS J. Comput.* 23(1):138–152.

Hüllermeier E, Fürnkranz J, Cheng W, Brinker K (2008) Label ranking by learning pairwise preferences. *Artif. Intell.* 172(16-17):1897–1916.

Kendall MG (1938) A new measure of rank correlation. *Biometrika* 30(1/2):81–93.

Li D, Chen C, Lu T, Chu SM, Gu N (2021) Mixture matrix approximation for collaborative filtering. *IEEE Trans. Knowl. Data Eng.* 33(6):2640–2653.

Namazi M, Sanderson C, Newton MAH, Polash MMA, Sattar A (2018) Diversified late acceptance search. *AI 2018: Adv. Artif. Intell. - 31st Australasian Joint Conf., Proc.*, 299–311.

Nápoles G, Falcon R, Dikopoulou Z, Papageorgiou E, Bello R, Vanhoof K (2017) Weighted aggregation of partial rankings using ant colony optimization. *Neurocomputing* 250:109–120.

Negahban S, Oh S, Shah D (2017) Rank centrality: Ranking from pairwise comparisons. *Oper. Res.* 65(1):266–287.

Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: A literature review. *Swarm Evol. Comput.* 2:1–14.

Pavai G, Geetha T (2016) A survey on crossover operators. *ACM Comput. Surv.* 49(4):1–43.

Rodrigo EG, Alfaro JC, Aledo JA, Gámez JA (2021) Mixture-based probabilistic graphical models for the label ranking problem. *Entropy* 23(4):420.

Tamine L, Goeuriot L (2022) Semantic information retrieval on medical texts: Research challenges, survey, and open issues. *ACM Comput. Surv.* 54(7):146:1–146:38.

Vicente-López E, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. *Oper. Res. Persp.* 3:43–58.

Werbin-Ofir H, Dery L, Shmueli E (2019) Beyond majority: Label ranking ensembles based on voting rules. *Expert Syst. Appl.* 136:50–61.

Yoo Y, Escobedo AR (2021) A new binary programming formulation and social choice property for kemeny rank aggregation. *Decis. Anal.* 18(4):296–320.

Zhou Y, Hao JK, Duval B (2017) Opposition-based memetic search for the maximum diversity problem. *IEEE Trans. Evol. Comput.* 21(5):731–745.

Zhou Y, Hao JK, Fu ZH, Wang Z, Lai X (2021) Variable population memetic search: A case study on the critical node problem. *IEEE Trans. Evol. Comput.* 25(1):187–200.

Zhou Y, Hao JK, Li Z, Glover F (2023a) Heuristic search for rank aggregation with application to label ranking. URL http://dx.doi.org/10.1287/ijoc.2022.0019.cd, https://github.com/INFORMSJoC/2022.0019.

Zhou Y, Kou Y, Zhou M (2023b) Bilevel memetic search approach to the soft-clustered vehicle routing problem. *Transp. Sci.* 57(3):701–716.

Zhou Y, Liu Y, Gao XZ, Qiu G (2014) A label ranking method based on gaussian mixture model. *Knowl. Based Syst.* 72:108–113.

Zhou Y, Qiu G (2018) Random forest for label ranking. *Expert Syst. Appl.* 112:99–109.

Zhou Y, Wang G, Hao JK, Geng N, Jiang Z (2023c) A fast tri-individual memetic search approach for the distance-based critical node problem. *Eur. J. Oper. Res.* 308(2):540–554.

Appendix. Online Supplement

A. Parameter Sensitivity Analysis

The enhanced late acceptance hill climbing (ELAHC) procedure reinforces the well-known late acceptance hill climbing (LAHC) method by a relaxed acceptance and replacement strategy (to increase the diversity of the search) and a fast incremental evaluation mechanism (to speed up the computation of cost function for the rank aggregation problem). The length of the history cost list ρ is the only parameter of both the LAHC and ELAHC methods, which strongly influences the convergence speed and solution quality.

To study the effect of the parameter ρ value, we experimentally compare the performance of both LAHC and ELAHC algorithms under different ρ values. Our experiments are conducted on ten representative instances, each solved ten times with $\hat{t} = 1000$ seconds. Detailed comparative results of LAHC and ELAHC algorithms are summarized in Tables 1 and 2, respectively. In these two tables, column 1 presents the instance name (Instance), columns 2-4 report the result of the algorithm with $\rho = 1$, identifying the best result ($\hat{f}$), the average result ($\bar{f}$) and the computation time ($\bar{t}$) over 10 runs. Columns 5-7, 8-10, 11-13 present the corresponding results of the algorithm for other three ρ values.

A.1. Sensitivity of LAHC to ρ

Table 1 Comparisons among LAHC Algorithms with Different ρ Values (i.e., $\rho \in \{1, 1000, 3000, 10000\}$)

	$\rho = 1$			$\rho = 1000$			$\rho = 3000$			$\rho = 10000$		
Instance	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
MM050n0.001_02	575.780	576.170	0.303	575.680	575.802	3.376	575.620	575.730	9.755	**575.560**	**575.706**	31.835
MM050n0.001_11	561.720	561.968	0.258	561.520	561.668	3.450	**561.500**	561.588	10.233	561.520	**561.578**	33.134
MM100n0.200_01	**416.000**	416.050	1.329	**416.000**	416.040	16.694	**416.000**	**416.012**	48.043	**416.000**	416.014	159.280
MM100n0.200_05	**411.720**	411.730	1.342	**411.720**	411.732	17.407	**411.720**	**411.726**	49.882	**411.720**	411.730	162.409
MM150n0.100_08	1249.720	1249.876	7.092	1249.680	1249.836	43.814	1249.660	1249.736	120.153	**1249.620**	**1249.706**	392.398
MM150n0.100_17	1266.190	1266.262	6.572	**1266.110**	1266.190	45.945	1266.130	1266.196	122.270	**1266.110**	**1266.160**	391.908
MM200n0.010_04	7669.290	7670.330	32.744	7668.570	7668.916	101.805	7668.010	7668.366	267.294	**7667.750**	**7668.006**	850.411
MM200n0.010_13	7706.640	7707.304	28.301	7705.140	7705.628	99.559	7704.820	7705.220	259.608	**7704.720**	**7704.990**	838.776
MM250n0.001_01	14358.490	14359.676	56.230	14354.470	14355.376	196.067	**14353.650**	**14354.220**	432.508	14359.390	14359.774	997.502
MM250n0.001_10	14446.360	14448.256	61.728	14443.260	14444.150	179.042	**14442.300**	**14443.120**	417.091	14448.260	14448.568	997.404
avg.value	4866.191	4866.762	19.590	4865.215	4865.534	70.716	**4864.941**	**4865.191**	173.684	4866.065	4866.223	485.506
avg.rank	3.500	3.650	–	2.500	2.800	–	**1.900**	**1.700**	–	2.100	1.850	–

Table 1 summarizes the results of the LAHC algorithms with different ρ values $\{1, 1000, 3000, 10000\}$. The bottom line gives the average value and average rank of each performance indicator. From this we observe that LAHC with a small ρ value (e.g., $\rho = 1$) quickly becomes trapped in a local optimum, leading to poor performance (with smallest average value and average rank). For large values of ρ (e.g., $\rho = 3000$), the search is less prone to becoming trapped but incurs the cost of a slower convergence speed. The solution quality can be poor if the computation time is not enough. To make a balance between the solution quality and computation time, a $\rho = 3000$ is suitable for LAHC, which achieves the smallest average values in terms of both $\hat{f}$ and $\bar{f}$. For the average rank, LAHC with $\rho = 3000$ also obtains the smallest rank value in terms of $\hat{f}$ and $\bar{f}$.

A.2. Sensitivity of ELAHC to ρ

Table 2 summarizes the results of the ELAHC algorithms with different ρ values $\{1,5,10,15\}$. The bottom line gives the average value and average rank of each performance indicator, disclosing that ELAHC with $\rho=1$ quickly converges to an unattractive local optimum, leading to poor performance (with smallest average value and average rank). For larger values of ρ (e.g., $\rho=5$), the search is less prone to becoming trapped but exhibits a slower convergence speed. The larger ρ value, the slower convergence speed. To make a balance between the solution quality and computation time, $\rho=5$ is suitable for ELAHC, achieving the smallest average values in terms of both $\hat{f}$ and $\bar{f}$. For the average rank, ELAHC with $\rho=5$ also obtains the second smallest rank value in terms of both $\hat{f}$ and $\bar{f}$.

Table 2 Comparisons among ELAHC Algorithms with Different ρ Values (i.e., $\rho\in\{1,5,10,15\}$)

	$\rho=1$			$\rho=5$			$\rho=10$			$\rho=15$		
Instance	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
MM050n0.001_02	575.900	576.194	0.160	575.640	575.728	1.861	575.600	575.688	8.723	**575.540**	**575.670**	18.766
MM050n0.001_11	561.700	561.922	0.130	561.500	561.574	1.940	**561.480**	561.554	9.326	**561.480**	**561.518**	22.007
MM100n0.200_01	**416.000**	416.048	0.675	**416.000**	416.012	19.953	**416.000**	416.004	110.484	**416.000**	**416.000**	278.871
MM100n0.200_05	**411.720**	411.736	0.738	**411.720**	411.730	20.690	**411.720**	**411.724**	126.336	**411.720**	**411.724**	302.653
MM150n0.100_08	1249.800	1249.918	3.270	**1249.580**	1249.636	55.452	**1249.580**	**1249.612**	263.505	**1249.580**	1249.618	645.462
MM150n0.100_17	1266.110	1266.202	4.579	1266.110	1266.140	48.320	**1266.090**	**1266.110**	249.941	**1266.090**	**1266.110**	599.331
MM200n0.010_04	7670.010	7670.356	15.841	7667.750	7667.952	262.215	**7667.670**	**7667.776**	983.677	7669.250	7669.456	991.769
MM200n0.010_13	7706.240	7707.396	23.169	**7704.580**	7704.756	254.853	7704.600	**7704.698**	989.951	7706.120	7706.390	993.734
MM250n0.001_01	14358.130	14359.780	28.262	**14352.570**	**14353.268**	412.931	14353.970	14354.246	994.869	14358.850	14359.134	994.320
MM250n0.001_10	14447.120	14448.066	44.503	**14441.640**	**14442.230**	405.860	14442.880	14443.088	996.642	14447.480	14447.902	995.623
avg.value	4866.273	4866.762	12.133	**4864.709**	**4864.903**	148.408	4864.959	4865.050	473.345	4866.211	4866.352	584.254
avg.rank	3.450	4.000	–	2.150	2.400	–	**1.900**	**1.600**	–	2.500	2.000	–

ELAHC is an enhanced version of LAHC. Due to the use of a relaxed acceptance and replacement strategy, more non-improving solutions are accepted and their cost values are updated in the list. Therefore, ELAHC requires a smaller ρ value than LAHC, as confirmed by the comparison between results summarized in Tables 1 and 2.

A.3. Sensitivity of HER to ρ

As indicated in the description of HER, the ELAHC method is used to perform local optimization during the search. To investigate the effect of ρ on the performance of HER, we run HER with four different ρ values $\{1,5,10,15\}$. Detailed results in terms of the best result (f_{best}) and average result (f_{avg}) are presented in Figure 1. The x-axis indicates the ρ values, and y-axis shows the performance gaps. By treating HER with $\rho=1$ as a baseline, we calculate the performance gap as $(f-\tilde{f})/\tilde{f}$, where f is the result of the HER algorithm with $\rho\in\{1,5,10,15\}$ and $\tilde{f}$ is the result of the HER algorithm with $\rho=1$. A performance gap smaller than zero means a better result for the corresponding instance.

From Figure 1, we observe that HER with a small $\rho\in\{1,5\}$ value demonstrates a significantly better performance than HER with a large $\rho\in\{10,15\}$. Taking the MM050n0.001_02 instance as an example,

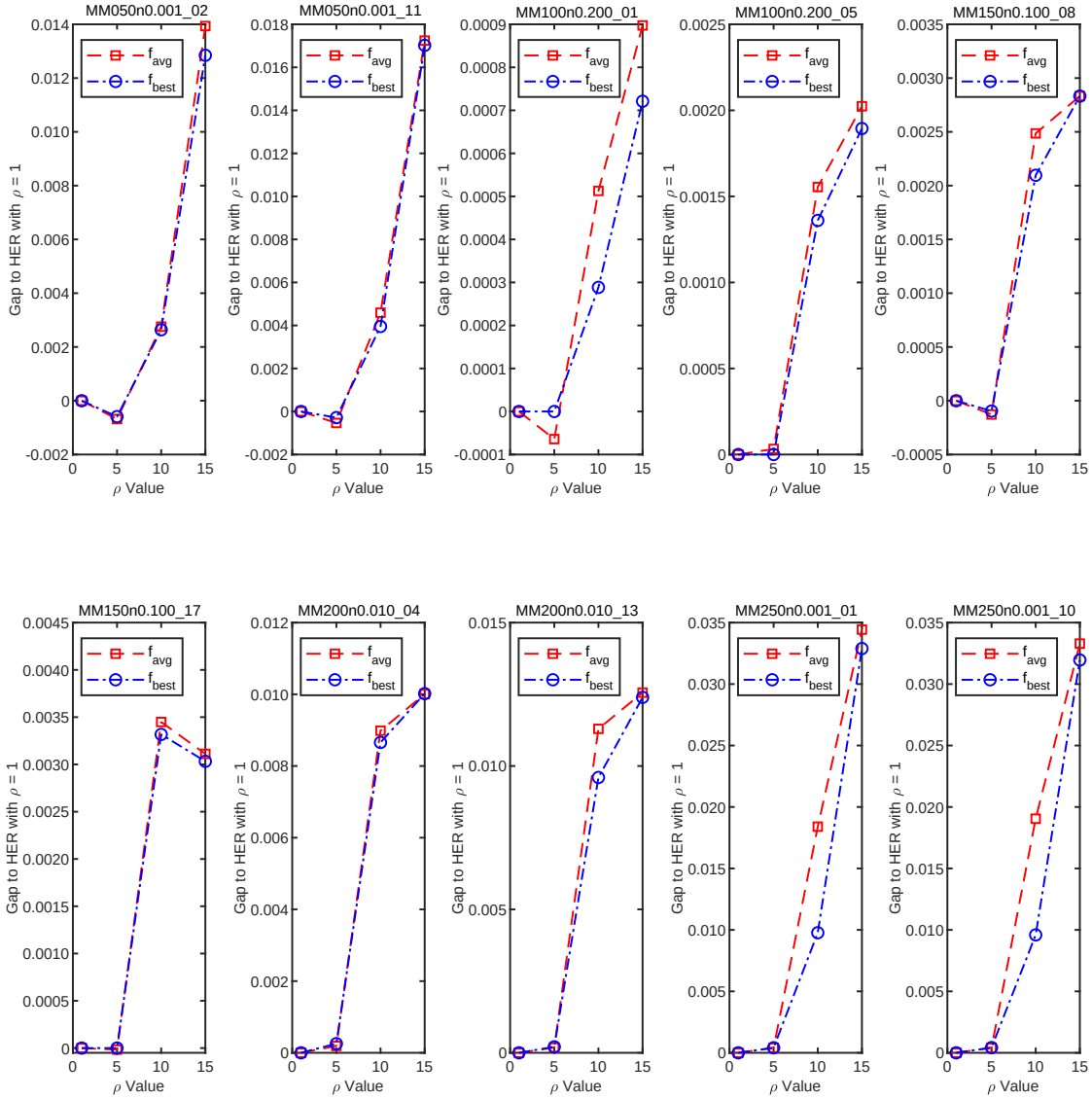

Figure 1 Comparison among HER Algorithms with different ρ Values ($\rho \in \{1, 5, 10, 15\}$)

we see that HER with $\rho = 5$ achieves the best performance in terms of both the best result and average result. Similar observations apply to the other nine instances. These results confirm that $\rho = 5$ is a suitable parameter value for ELAHC used in HER.

B. Comparison Between Iterated Local Search and its Improved Version

The original iterated local search (ILS) method for the rank aggregation problem is a multi-start local search algorithm based on the hill climbing (HC) algorithm, which demonstrates the best performance when the algorithms are allowed to perform a large number of fitness evaluations. Note that ILS employs HC to perform local optimization, which causes it to achieve a bad performance. To remedy this, we implemented an improved ILS method by replacing HC with the enhance late acceptance hill climbing (ELAHC) with $\rho = 1$. We conducted a detailed performance comparison between the original ILS and improved ILS on both synthetic and real-world instances with complete rankings, whose outcomes are as follows.

B.1. Results on Synthetic Instances

Table 3 summarizes the detailed comparative results on synthetic instances. For each algorithm, we report the best result ($\hat{f}$), the average result ($\bar{f}$), and the average time in seconds ($\bar{t}$). From Table 3, we observe that the improved ILS method performs better than the original method in all 20 cases in terms of both $\hat{f}$ and $\bar{f}$.

Table 3 Comparisons of the Original ILS and Improved ILS on Synthetic Instances

Instance		Original ILS (with HC)			Improved ILS (with ELAHC)		
θ	m	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
0.200	050	337.900	357.308	3164.665	**183.140**	**187.913**	723.482
0.100	050	417.180	426.984	3607.289	**311.950**	**320.296**	30.213
0.010	050	571.910	577.889	4331.544	**550.970**	**559.797**	333.231
0.001	050	579.480	584.879	4090.002	**561.740**	**569.968**	1137.292
0.200	100	1590.090	1663.185	4357.084	**405.140**	**411.884**	389.789
0.100	100	1697.160	1756.552	3571.267	**776.020**	**787.745**	1190.498
0.010	100	2303.440	2320.391	3074.636	**2126.890**	**2153.107**	3401.334
0.001	100	2376.530	2393.986	3547.358	**2290.990**	**2306.096**	3662.618
0.200	150	3911.680	4072.344	4091.270	**629.410**	**636.948**	152.134
0.100	150	4028.860	4164.913	3813.916	**1245.730**	**1260.166**	2464.411
0.010	150	5123.720	5188.640	3637.212	**4533.430**	**4586.122**	2082.753
0.001	150	5406.240	5431.022	4491.537	**5144.480**	**5191.362**	3788.802
0.200	200	7491.710	7628.279	2548.153	**851.290**	**862.693**	115.287
0.100	200	7535.900	7714.423	3483.566	**1711.440**	**1734.231**	2286.683
0.010	200	9070.050	9185.279	3625.224	**7625.320**	**7698.460**	3830.479
0.001	200	9667.810	9706.857	3599.662	**9182.900**	**9240.152**	3768.522
0.200	250	11819.520	12354.986	3268.002	**1075.790**	**1087.684**	456.757
0.100	250	11888.110	12359.140	4139.456	**2186.420**	**2206.824**	3190.976
0.010	250	14002.130	14276.435	3707.153	**11182.830**	**11301.168**	3641.871
0.001	250	15180.200	15219.913	3593.352	**14337.260**	**14434.251**	3097.111

* The results of each combination of θ and m are averaged over 20 instances.

B.2. Results on Real-world Instances

Table 4 summarizes the detailed comparative results on real-world instances, leading to the same conclusion as in Table 3, that the improved ILS method significantly outperforms the original ILS method.

Table 4 Comparisons of the Original ILS and Improved ILS on Real-world Instances

	Original ILS (with HC)			Improved ILS (with ELAHC)		
Instance	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
Sushi	**19.181**	19.190	2772.938	**19.181**	**19.181**	1.003
F1	77.450	79.930	3714.395	**68.750**	**68.750**	0.197
Tour	4882.300	4913.240	4001.894	**3480.100**	**3480.800**	2938.547
ATPWomen_50	353.481	361.592	3531.377	**185.750**	**185.750**	4.264
ATPWomen_100	1693.558	1739.742	3072.500	**715.019**	**715.019**	954.638
ATPWomen_200	7743.269	7858.192	4309.973	**2823.423**	**2823.596**	3467.266
ATPMen_50	346.269	364.565	2777.543	**197.962**	**197.962**	7.795
ATPMen_100	1746.250	1779.123	4194.137	**862.365**	**862.365**	1947.933
ATPMen_200	7742.212	7899.108	4777.542	**2810.519**	**2810.823**	3543.888