Unbalanced Mallows Models for Optimizing Expensive Black-Box Permutation Problems

Ekhine Irurozki*
Telecom Paris
Paris, France
irurozki@telecom-paris.fr

Manuel López-Ibáñez
University of Málaga
Málaga, Spain
manuel.lopez-ibanez@uma.es

ABSTRACT

Expensive black-box combinatorial optimization problems arise in practice when the objective function is evaluated by means of a simulator or a real-world experiment. Since each fitness evaluation is expensive in terms of time or resources, the number of possible evaluations is typically several orders of magnitude smaller than in non-expensive problems. Classical optimization methods are not useful in this scenario. In this paper, we propose and analyze UMM, an estimation-of-distribution (EDA) algorithm based on a Mallows probabilistic model and unbalanced rank aggregation (uBorda). Experimental results on black-box versions of LOP and PFSP show that UMM outperforms the solutions obtained by CEGO, a Bayesian optimization algorithm for combinatorial optimization. Nevertheless, a slight modification to CEGO, based on the different interpretations for rankings and orderings, significantly improves its performance, thus producing solutions that are slightly better than those of UMM and dramatically better than the original version. Another benefit of UMM is that its computational complexity increases linearly with both the number of function evaluations and the permutation size, which results in computation times an order of magnitude shorter than CEGO, making it specially useful when both computation time and number of evaluations are limited.

CCS CONCEPTS

• **Mathematics of computing → Combinatorial optimization**;
• **Theory of computation → Random search heuristics**.

KEYWORDS

Combinatorial optimization, Bayesian optimization, Expensive black-box optimization, Estimation of distribution algorithms

ACM Reference Format:
Ekhine Irurozki and Manuel López-Ibáñez. 2021. Unbalanced Mallows Models for Optimizing Expensive Black-Box Permutation Problems. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459366

*Also with Basque Center for Applied Mathematics.

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07…$15.00
https://doi.org/10.1145/3449639.3459366

1 INTRODUCTION

In many practical optimization problems, the objective function is not explicitly available and solutions are evaluated by means of expensive prediction models, simulations or physical experiments. When decision variables are continuous, the use of Bayesian surrogate-models (e.g., Gaussian processes) is widespread in optimization [11, 14]. Motivated by this success, there have been attempts at adapting such surrogate-based optimization algorithms to the combinatorial case [15], a notable example being Combinatorial Efficient Global Optimization (CEGO) [18, 19]. However, the ruggedness of combinatorial landscapes lessens the effectiveness of global surrogate models [8]. Moreover, surrogate models are expensive to train and optimize. The additional time required by the Bayesian optimizer may impose a significant overhead in computation time, specially if computation time of each function evaluation is measured in "few" minutes or hours rather than days and when "expensive" refers to resources or economical cost rather than wall-clock time. Pérez Cáceres et al. [16] recently showed that ant colony optimization (ACO) is competitive with CEGO on a black-box version of the travelling salesman problem under a budget of 1 000 function evaluations. On the other hand, 1 000 evaluations is still a relatively large budget for CEGO, whereas ACO was not designed for such short budgets.

In this work, we propose and analyze the Unbalanced Mallows Model (UMM) algorithm, a population-based probabilistic algorithm specifically designed for optimizing black-box problems over a permutation landscape under a limited budget. There exists a wide body of research on probabilistic models for permutations, such as Mallows [10], that have already been used in optimization problems successfully, with the most prominent examples being estimation of distribution algorithms (EDAs) [5]. ACO may also be considered a type of EDA, since it builds a probability distribution model from which solutions are sampled with a bias towards the best solutions evaluated so far. These classical EDAs assume that function evaluations are cheap and/or the fitness function is white-box.

In contrast, UMM aims at learning as much as possible with a limited number of function evaluations. Moreover, unlike EDAs, UMM does not rely on a fixed-size population of solutions that evolves over "generations". Instead, taking inspiration from Bayesian optimizers, UMM considers a sample of permutations during the whole execution that is increased by one new permutation at each iteration. The sample is used to learn a Mallows model from which a single new permutation is sampled. The model is learned by using uBorda [13], a generalization of the well-known rank aggregation algorithm Borda [1]. Rank aggregation algorithms summarize in a single permutation the information of a sample of permutations,

and they can be seen as functions to compute the *sample mean*. The uBorda algorithm computes a *weighted sample mean* and has been shown to return the correct estimator with high probability on streaming contexts [13]. Finally, the balance between intensification and exploration is dynamically adapted by controlling the variance of the probabilistic Mallows model.

This paper is structured as follows. Section 2 summarizes fundamental concepts. Section 3 presents the UMM algorithm, which is the main contribution of this paper. UMM is analyzed using the experimental setup presented in Section 4 and results are presented in Section 5. We end the paper with overall conclusions and a call for further research on this type of optimization scenario (Section 6).

2 BACKGROUND

2.1 Permutations: rankings vs orderings

Permutations are bijections of the set $[n]$ of integers onto itself. The set of all permutations of n values is denoted by S_n and has cardinality $n!$. We will use the one-line notation and denote permutations with Greek letters, $\sigma = (\sigma(1), \ldots, \sigma(n))$. The composition of σ and π is $\sigma\pi = (\sigma(\pi(1)), \ldots, \sigma(\pi(n)))$ and the inverse of σ is σ^{-1}, for which the relation $\sigma\sigma^{-1} = (1, 2, 3, \ldots, n)$ always holds.

Throughout this paper and for consistency, permutation σ represents a *ranking*: $\sigma(i) = j$ denotes that item i has rank j. The *ordering* associated with σ is given by its inverse σ^{-1}, therefore, $\sigma(i) = j \Leftrightarrow \sigma^{-1}(j) = i$. This distinction is importante since statistical methodologies typically assume rankings whereas the optimization literature often assumes orderings. Mixing both concepts is a very common confusion with grave consequences, as we will illustrate in the experimental section.

Kendall's-τ distance is the metric for rankings that counts the pairs of items ranked in different order in two permutations:

$$d(\sigma_1, \sigma_2) = \left| \left\{ (i,j) : i < j \wedge \Big(\big(\sigma_1(i) < \sigma_1(j) \wedge \sigma_2(i) > \sigma_2(j)\big) \vee \big(\sigma_1(i) > \sigma_1(j) \wedge \sigma_2(i) < \sigma_2(j)\big) \Big) \right\} \right| \quad (1)$$

An equivalent definition for the Kendall's-τ distance counts the number of adjacent swaps that have to be made to convert σ_1^{-1} into σ_2^{-1} and, therefore, it is sometimes called *swap distance* [18] or adjacent swap distance.

2.2 Distributions over permutations

Distributions over permutations are functions that assign a probability value $p(\sigma) \in [0, 1]$ to each permutation $\sigma \in S_n$. One of the most popular distributions is the Mallows Model (MM), which is considered as an analogous to the Gaussian distribution for permutations. The MM defines the probability of each permutation $\sigma \in S_n$ as follows:

$$p(\sigma \mid \sigma_0, \theta) = \frac{\exp(-\theta d(\sigma, \sigma_0))}{\psi} \sim MM(\sigma_0, \theta) \quad (2)$$

with two parameters, θ and σ_0. Permutation $\sigma_0 \in S_n$ is a reference permutation and has the largest probability value, i.e., the mode of the distribution. The probability of every permutation $\sigma \in S_n$ decays exponentially as its distance $d(\sigma, \sigma_0)$ increases, and the dispersion parameter θ controls this decay. The normalization constant ψ can be easily computed for the Kendall's-τ distance (Eq. 1) as well as for the Hamming, Cayley and Ulam distances [12].

We will use the standard maximum likelihood estimation (MLE) approach to fit the parameters of a MM for a given collection of permutations. The MLE process is divided in two stages: first, we estimate the central permutation of the distribution, $\hat{\sigma}_0$ and, second, compute the dispersion parameter, $\hat{\theta}$ [12].

The exact MLE for σ_0 is given by the well-known Kemeny ranking [7]. Unfortunately, obtaining such ranking is NP-hard [7]. An alternative to the Kemeny ranking is the Borda ranking [1], which has several advantages: (i) it requires polynomial computational time; (ii) it guaranties high quality parameters for a sample distributed according to the MM [4]; and (iii) when no distribution is assumed, Borda is an approximation to Kemeny [6].

The Borda ranking of a sample of permutations $S \subset S_n$ is computed as follows. First, for each item $i \in [n]$, compute its Borda score $B(i) = \sum_{\sigma \in S} \sigma(i)$. Then, assign rank 1 to the item with the smallest Borda score, rank 2 to the item with the second smallest score and so on, i.e., order the positions by Borda score increasingly.

2.3 Unbalanced Borda

We introduce now the core of our probabilistic algorithm, the uBorda algorithm. Following the convention of the community, the ranking returned by the uBorda algorithm will be denoted uBorda ranking. The uBorda algorithm is a recent generalization of Borda [13]. In uBorda, we consider that each permutation in the sample $\sigma \in S$ has an associated weight $w(\sigma)$ where $w \colon S_n \to \mathbb{R}^+$. Then, the score of item $i \in [n]$ is defined as follows:

$$B(i) = \sum_{\sigma \in S} w(\sigma)\sigma(i) \ , \quad (3)$$

and, the same as Borda, orders the items $i \in [n]$ increasingly by their score.

Borda and uBorda are equivalent when $w(\sigma)$ is constant for every σ. Otherwise, uBorda ranking will be closer to the rankings in S with larger value of $w(\sigma)$. Roughly speaking, it is equivalent to replicating in the sample the σ with high values of $w(\sigma)$. The choice of $w(\sigma)$ depends on the relevant property of a ranking in a particular domain. Finally, uBorda is computed in polynomial time.

2.4 Black-box combinatorial optimization under a limited budget

Let us assume a black-box "fitness" function that must be minimized over a space of permutations $f \colon S_n \to \mathbb{R}$. Being black-box means that we can evaluate any candidate permutation $\sigma \in S_n$ to obtain $f(\sigma)$, however, we do not know anything else about the form of f. Moreover, the evaluation of f is expensive in computation time or resources and, thus, we can only evaluate a limited budget m of candidate permutations. Here, we will study budgets lower than 400. Such expensive black-box combinatorial problems arise in diverse contexts, for example, protein folding [17] and industrial production [9], where the fitness function may involve expensive simulations for which no closed-form mathematical description is available. The black-box formulation and limited budget preclude most of the optimization techniques that are successful in combinatorial optimization, such as constructive heuristics and local search,

Algorithm 1 UMM: Unbalanced Mallows Model Algorithm

Require: m_{ini}: number of initial permutations, m: total budget, r_1, r_2: parameters controlling the learning rate

1: $S := \emptyset$
2: **for** $i := 0$ **to** m_{ini} **do**
3: $\sigma :=$ generate uniformly at random
4: $S := S \cup \langle \sigma, f(\sigma) \rangle$ ▷ Evaluate it
5: **for** $i := m_{\text{ini}}$ **to** m evaluations **do**
6: $\hat{\sigma}_0 := \text{uBorda}(S, \rho)$ ▷ Learning Step
7: Initialize (first iteration) or decrease θ ▷ See *Sampling Step* and Eq. 5
8: $\sigma :=$ sample a permutation from $MM(\hat{\sigma}_0, \theta)$
9: $S := S \cup \langle \sigma, f(\sigma) \rangle$ ▷ Evaluate it
10: **return** $\langle \sigma^*, f(\sigma^*) \rangle := \arg\min_{\langle \sigma, f(\sigma) \rangle \in S} f(\sigma)$

leading to a particularly challenging scenario, perhaps even more challenging than its continuous counter-part.

3 UNBALANCED MALLOWS MODEL (UMM)

In this section we describe our main contribution: Unbalanced Mallows model (UMM), a novel algorithm for the optimization of an expensive black-box function over permutation spaces. This algorithm is a probabilistic method based on the Mallows model and uBorda learning.

As shown in Algorithm 1, UMM starts with a *small* sample of m_{ini} permutations that are generated uniformly at random (although it would be possible to employ other sampling methods such as max-min-distance sequential design) and evaluated. Then, using the current sample S, the uBorda algorithm is applied to learn a reference permutation $\hat{\sigma}_0$ (line 6). Together with the parameter θ, the algorithm defines the Mallows model $MM(\hat{\sigma}_0, \theta)$. A decreasing θ parameter along iterations (line 7) implies a decreasing variance of the Mallows model (Eq. 2), as explained in Section 3.2 below. Then, we sample a new candidate permutation from $MM(\hat{\sigma}_0, \theta)$ (line 8, also in Section 3.2). This candidate permutation (solution) is evaluated and added to the sample S. The process is repeated up to a given maximum budget of evaluations m.

The key idea behind UMM is that the probabilistic Mallows model (MM) used in UMM does not simply represent the current sample S. Instead, the model assigns higher probability values to the permutations with known (or expected) minimal fitness values. This is achieved by means of the uBorda ranking (Section 2.3) with $w(\sigma) = \rho^{f(\sigma)}$ and $\rho \in (0, 1)$. Intuitively, uBorda behaves as if there were many copies of the best solutions and few of the bad ones. A property of UMM is that the computational complexity of working with the original sample (Borda) and with the transformed sample (uBorda) is the same, i.e., polynomial. The most important steps are the update (*learning*) of the parameters σ_0, θ and ρ (line 6 in Algorithm 1) and the sampling from $MM(\sigma_0, \theta)$ (line 8)

3.1 Learning step

At each iteration, we start by fitting a MM to the sample S, which implies estimating $\hat{\sigma}_0$ and θ. Parameter θ is updated deterministically at the sampling step. The parameter σ_0, however, is learned from the current sample S by using the uBorda algorithm with a weight $w(\sigma) = \rho^{\text{scale}(f(\sigma))}$ (Section 2.3). The function scale: $\mathbb{R} \to [0, 1]$ is defined as $\text{scale}(f) = (f - f_{\min})/(f_{\max} - f_{\min})$, where $f_{\min}$ and $f_{\max}$ are, respectively, the minimum and maximum fitness values in S. A possible alternative to scaling would be ranking the elements in S with respect to their fitness values, which would make UMM invariant to non-linear monotonic transformations of the fitness function. However, in most applications, the fitness values have a real-world meaning (e.g., monetary value) and the relative scale of fitness differences is relevant.

The learning rate is controlled by parameter $0 < \rho < 1$. In UMM, the value of ρ depends on the set of true fitness evaluations saved in S and it is set at each iteration such that the largest $100r_1\%$ of the mass of the weights is concentrated in the best $100r_2\%$ of the solutions in S:

$$r_1 \cdot \sum_{\langle \sigma, f(\sigma) \rangle \in S} \rho^{\text{scale}(f(\sigma))} = \sum_{\langle \sigma', f(\sigma') \rangle \in S'} \rho^{\text{scale}(f(\sigma'))} \quad (4)$$

where $|S'| = r_2|S| \wedge \forall \langle \sigma', f(\sigma') \rangle \in S', \langle \sigma, f(\sigma) \rangle \in S \setminus S' : f(\sigma') \leq f(\sigma)$, i.e., S' contains the $r_2|S|$ elements of S with the best fitness.

3.2 Sampling step

In this step, the algorithm samples from the distribution the new data point to be evaluated. Sampling from $MM(\sigma_0, \theta)$ can be done efficiently given the parameters σ_0 and θ, as studied by Irurozki et al. [12], where the computational complexity is $O(n \log n)$. At each iteration of UMM, $\sigma_0 = \hat{\sigma}_0$ is the permutation obtained in the previous learning step. The scale θ controls the expected value and variance of the distance D of a random permutation $\sigma \sim MM(\sigma_0, \theta)$ to the location parameter σ_0. Both the expected value of D and its variance increase monotonically with the value of θ. In particular, the expected distance $\mathbb{E}[D]$ is given by [10]:

$$\mathbb{E}[D] = \frac{n \cdot \exp(-\theta)}{1 - \exp(-\theta)} - \sum_{j=1}^{n} \frac{j \cdot \exp(-j\theta)}{1 - \exp(-j\theta)} \quad (5)$$

Parameter θ is set automatically by the algorithm to control the diversification and intensification trade off. At the first iteration, θ is set so that $\mathbb{E}[D] = \binom{n}{2}/2$, i.e., half of the expected distance for the uniform distribution. Then the expected distance linearly decreases until the last iteration m where θ is set such that $\mathbb{E}[D] = 1$. Despite it is not possible to isolate θ in Eq. (5), its monotonicity allows the use of bisection methods for efficiently finding an approximation.

Setting $\mathbb{E}[D] = \binom{n}{2}$ will generate random permutations at early stages, slowing the algorithm convergence. A purely explotation approach will set $\mathbb{E}[D] = 0$ generating σ_0 in the sampling stage. However, the setting of a decreasing variance controls the exploration-explotation tradeoff as a function of the number of function evaluations (m) that the algorithm performs, thus the behaviour is different for different choices of m. Similar approaches have been taken in different contexts [3].

4 EXPERIMENTAL SETUP

UMM. UMM initially samples $m_{\text{ini}} = 10$ solutions uniformly. The other two parameters of UMM are r_1 and r_2 that determine the learning rate ρ in uBorda, and consequently, determine the balance

between exploration and exploitation. In the experimental section we will study their impact.

CEGO. Combinatorial Efficient Global Optimization (CEGO) [19] is an extension of the well-known EGO method [14] to unconstrained black-box combinatorial optimization problems. In EGO, Gaussian process models are used as a surrogate of the landscape of the expensive original problem. An optimization method searches for solutions in the surrogate model by optimizing the expected improvement criterion, which balances the expected mean and variance of the chosen solution. Once a solution is chosen, it is evaluated on the actual fitness function and the result is used to update the surrogate-model, hopefully increasing its predictive power.

CEGO replaces the Euclidean distance measure, used by the surrogate model in EGO, with a distance measure appropriate to combinatorial landscapes [18]. In CEGO, the surrogate model is explored by a GA with crossover and mutation operators appropriate for the particular combinatorial problem. A thorough comparison of CEGO with other optimizers for expensive black-box combinatorial optimization ranked it as the best-performing [19].

We use here the original implementation of GECO.[1] Although it is never stated in the original paper, the implementation of CEGO generates a set of initial solutions of size $m_{\text{ini}} = 10$ by means of a max-min-distance sequential design: new solutions are added to the set sequentially by maximizing the minimum distance to solutions already in the set. These initial solutions are then evaluated on the true fitness function and the result is used to build the initial surrogate model. Following the authors of CEGO [18, 19], the surrogate model is optimized by a GA with population size of 20, crossover rate of 0.5, mutation rate of $1/n$, tournament selection of size 2 with probability of 0.9, interchange mutation (i.e., exchanging two randomly selected elements) and cycle crossover for permutations. CEGO runs the GA with a budget of 10^4 evaluations of the surrogate model to generate a new solution, which is then evaluated on the true fitness function. The original paper [19, p. 875] notes that coupling the GA with local search does not improve the results significantly since the model is anyway an inexact estimation of the original fitness function.

Benchmark problems. Experiments with real-world expensive black-box problems would be computationally infeasible. Instead, we consider classical combinatorial optimization problems as black-box optimization problems. For consistency with the rest of the paper, a permutation σ denotes a ranking also in this section. Recall that the ordering associated to ranking σ is given by its inverse, σ^{-1}. Thus, we will use ordering σ^{-1} to formulate the problems. Later in the experimental section we discuss the effects of not performing this inversion.

The *Linear Ordering Problem (LOP)*, when minimizing as in our case, is defined as

$$\min_{\sigma \in S_n} f(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} a_{\sigma^{-1}(i),\sigma^{-1}(j)} \tag{6}$$

where $[a_{i,j}]$ is a matrix of size $n \times n$.

The *Permutation Flowshop Scheduling Problem (PFSP)* is defined by a matrix $[p_{ij}]$ of size $n \times M$ that gives the processing time of a job $i \in [n]$ on a machine $j \in [M]$. All jobs must be processed by all machines in the same order. Given a permutation (ranking) σ, $C_{i,j}$ denotes the completion time of job $\sigma^{-1}(i)$ at position i on machine j, and $C_{\max} = C_{n,M}$ is the completion time of the last job on the last machine, i.e., the makespan. The objective of the PFSP is to find:

$$\begin{aligned} \min_{\sigma \in S_n} \quad & f(\sigma) = C_{n,M} \\ \text{s.t.} \quad & C_{1,j} = 0 \quad j \in [M], \qquad C_{i,1} = 0 \quad i \in [n] \\ & C_{i,j} = p_{\sigma^{-1}(i),j} + \max\{C_{i-1,j}, C_{i,j-1}\} \quad i \in \{2,\dots,n\}, \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j \in \{2,\dots,M\} \end{aligned} \tag{7}$$

Datasets. We consider LOP datasets from the LOLIB repository.[2] In particular, we have taken two instances of various types focusing on relatively small instances, since he computation time of CEGO grows rapidly with instance size: `N-p40-01`, `N-p40-02`, `N-p50-01` and `N-p50-02` of type *RandB*; `N-t59d11xx` and `N-t59b11xx` of type *IO*; and `N-sgb75.01` and `N-sgb75.02` of type *SGB*. In the case of the PFSP, we consider the same instances as Zaefferer et al. [18], i.e., `reC05`, `reC13`, `reC19` and `rec31`, with $n \in \{20, 20, 30, 50\}$ and $M \in \{5, 20, 10, 15\}$, respectively.

Other settings. We consider three different values $m \in \{100, 200, 400\}$ for the maximum budget of evaluations of the actual objective function in UMM. In the case of CEGO, we only run experiments with the largest budget $m = 400$. In a white-box context, state-of-the-art algorithms for the LOP and PFSP typically evaluate thousands of solutions, thus, the budget considered here for the black-box context is extremely limited. For simplicity, we use Kendall's-τ distance (Eq. 1) in all experiments. We plan to extend UMM to other distance measures in the future.

Computing environment. Experiments were run on Intel Xeon *Ivybridge* E5-2650v2 CPUs at 2.60 GHz, 64 GB RAM running CentOS Linux release 7 (Core).

5 EXPERIMENTAL ANALYSIS

5.1 Analysis of r_1 and r_2 parameters

The first part of the experimental analysis concerns the analysis of the learning rate, which corresponds to parameter ρ of uBorda. As stated in Section 3.2, ρ is automatically set at each iteration in such a way that the $100r_1\%$ of the samples have the $100r_2\%$ of the weight. In this regard, we tried a few different values of parameters r_1 and r_2 in all the instances. Only values of $r_1 < r_2$ make sense and $r_1 = r_2 = 0.5$ would produce equal weights in the learning step. Thus, we tried values $r_1 = \{0.1, 0.2, \dots, 0.5\}$ and $r_2 = \{0.6, \dots, 0.9, 0.99\}$. Figure 1 shows the best solution found by each independent run (averaged over 10 runs) for instances: `N-t59d11xx` and `N-sgb75.02` for the LOP, and `reC13` and `reC19` for the PFSP. Results for all the datasets can be found in the supplemental material (doi: 10.5281/zenodo.4500974).

The plot shows that UMM is indeed very sensitive to the values of r_1 and r_2, yet, low values of r_1 and high values of r_2 almost always produce the best results, independently of the problem. In particular, we observe that the pattern on the two top figures differs from the two bottom ones. While on the top figures the fitness gets worse homogeneously from the top-right corner to the bottom-left one, the pattern on the bottom figures is not as

[1] https://cran.r-project.org/package=CEGO

[2] http://grafo.etsii.urjc.es/optsicom/lolib/

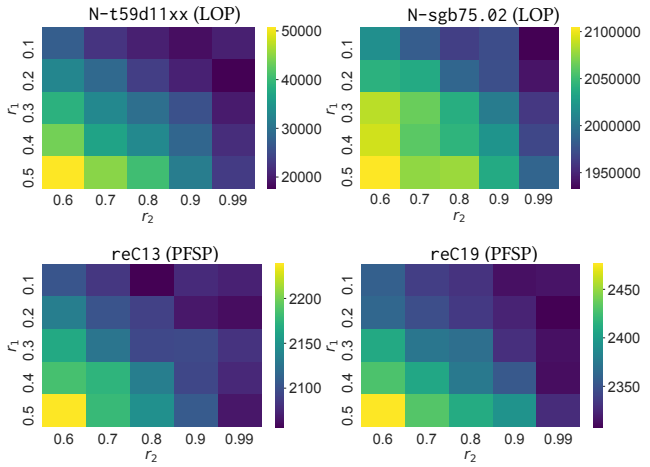

Figure 1: Mean fitness (over 10 runs) of different configurations of r_1 and r_2 (darker is better).

homogeneous, yet still fitness gets worse with decreasing r_2. We observe that this behavior is related to the adequacy of UMM to the problem: the homogeneous behavior of parameters r_1 and r_2 depends on the problem (not on the particular instances) and the more homogeneous the behavior, the better performance of UMM. We elaborate this argument in detail in the next section.

From this preliminary experiment, it is not obvious how the best settings are related to problem features, such as permutation length n. On the other hand, given the sharp transitions in the plots, it is clear that further fine-tuning may improve performance and uncover further patterns.

Over all the instances, we observe that the configuration $r_1 = 0.1$, $r_2 = 0.9$ has good performance, i.e., its mean rank (over 10 runs) relative to all the other configurations is less than 4 for all the instances. Although automatic parameter tuning may likely provide even more fine-tuned settings, our goal here is to understand how UMM performs in comparison to an existing Bayesian method (CEGO) and automatically tuning the parameters of CEGO is not computationally feasible. For consistency, in the following sections, we run UMM with this parameter configuration. Moreover, we recommend this configuration as the current default of UMM.

5.2 Experiments with rankings and orderings

In Section 2.1, we highlighted the important difference between rankings and orderings, usually neglected in the optimization literature. Statistical models (Borda or the Kendall's-τ distance) are usually defined for rankings while the optimization literature usually builds upon orderings. For example, the (adjacent) swap distance $d_s(\sigma, \pi)$ is the Kendall's-τ distance counterpart in optimization, $d_s(\sigma, \pi) = d(\sigma^{-1}, \pi^{-1})$.

As a consequence of this important distinction, when dealing with permutations, in particular in combinatorial problems, we must convert rankings to orderings by inverting the permutation, as done in Eq. 6 and Eq. 7, for the LOP and PFSP, respectively. This might seem an implementation detail but a failure to use the correct interpretation has grave consequences.

During our experiments, we realized that distance metric called Swap in CEGO [18] actually assumes rankings, that is, it actually implements the Kendall's-τ distance, yet CEGO does not invert the permutation before evaluating the PFSP. We verified that this is indeed how the implementation works and that those results match the published ones [18]. Thus, the obvious question arises about the effect of inverting the permutation before calling the objective function, so that the rankings are transformed to orderings, as done by our UMM algorithm. In the following, we show results of both variants, that is, the original CEGO (CEGO_{ORI}) and our modified version that inverts the permutation before evaluation (CEGO_{INV}).

5.3 Experimental Analysis

Since the number of evaluations is small and the algorithms evaluate just one solution at each step, we chose to record and plot the fitness of the solution evaluated at each step of each run, instead of the usual plots of the best-so-far solution over number of evaluations. We believe that plotting each evaluated solution provides better insights on the search dynamics of the algorithms analyzed here. Please note that the algorithms do not return the final solution shown in the plots, but rather the best one (minimum fitness) of each run. Nevertheless, the minimum values achieved in the plot up to a particular number of function evaluations gives an estimation of the mean fitness of the best-so-far solution up to that point in the run. We provide later an analysis of the best solution found by each run of the algorithms.

Figure 2 shows the results on the real-world instances from LOLIB and PFSP. There is one plot per instance. The horizontal line indicates the best-known fitness for that instance. Besides, there are 5 different lines representing each of the different algorithm variants tested, in particular:

- UMM_{100}, UMM_{200} and UMM_{400} are three different versions of UMM, each considering a different total budget of function evaluations. As can be expected, the lines corresponding to UMM_{100} and UMM_{200} do not reach the right corner of the figure. As mentioned in Sec. 3.2, the behavior of UMM depends on the value of m.
- CEGO_{ORI} and CEGO_{INV} are, respectively, the original version of CEGO and the modification discussed in Section 5.2, i.e., the ranking generated by the CEGO algorithm at each iteration, σ, is inverted before being evaluated, so the evaluation is performed on σ^{-1}.

For each algorithm, we plot the mean fitness of the solution evaluated at each step over 10 runs with different random seeds. The shaded area shows one standard deviation around the mean. Ideally, each new evaluation will monotonically improve in fitness, since each solution evaluated helps to refine the model and leads to a better solution being evaluated in the next step. However, it is possible that new solutions are worse than their predecessors.

Convergence analysis of UMM. We start our analysis with the three variants of UMM. We can see a common trend in all instances: After the 10 random initial permutations, the fitness of new solutions decreases with a linear trend along iterations. The (close to) linear decrease is given by the linear decrease in the expected distance of the model, as described in Section 3.2. Overall, the rate of improvement does not seem to converge, not even for the longer

runs of 400 evaluations. Therefore, we conjecture that longer runs are likely to produce further improved solutions. Moreover, a non-linear decrease in the expected distance should result in a faster convergence for larger evaluation budgets.

Convergence analysis of CEGO. The red and the gold lines in the plots shown in Fig. 2 correspond to the two variants of CEGO the original CEGO_{ORI} and our modified version CEGO_{INV}. The difference between these two variants is overwhelmingly in favour of CEGO_{INV}. With the exception of the `reC19` PFSP instance, the search behavior of CEGO_{ORI} resembles a random search, after a brief improvement phase at the start of the run. By contrast, CEGO_{INV} shows a very fast improvement in fitness within the first 50 evaluations and tends to converge around the 150^{th} evaluation, showing little improvement after that. Since the only difference between these two variants is the transformation from rankings to orderings before evaluating the objective function (Sec. 5.2), we must conclude that this difference has tremendous effect in performance.

Comparison of UMM and CEGO. When comparing the UMM and CEGO variants shown in Fig. 2, we can observe that all UMM variants perform much better than the original CEGO (CEGO_{ORI}), while they converge slower than our modified CEGO_{INV}. However, the rate of improvement of the UMM variants does not seem to slow down near the maximum budget given and, in several instances such as `rec31` and `N-sgb75-01`, matches and surpasses the solutions found by CEGO_{INV}. Interestingly, this is the case, even for the shorter budget variants of UMM, i.e., UMM_{100} and UMM_{200}. We also observe a pattern that UMM performs better, or CEGO_{INV} performs worse, on larger instances, as can be seen by comparing `rec19` with `rec31`.

Comparison of the best solutions found. We now turn our analysis to the best (minimum) fitness found at the end of the run by the two best algorithms, that is, CEGO_{INV} and UMM_{400}. We show, in Table 1, the mean fitness (and standard deviation), over the 10 runs of each algorithm, of the best solution found at the end of the run, together with a 95% confidence interval (CI) around their mean fitness difference. Except for PFSP instances `rec05` and `rec31`, the differences are statistically significant in favour of CEGO_{INV} (the intervals do not contain the value zero). Nevertheless, the CIs indicate that the differences, although consistent, are relatively small. Detailed results of UMM_{100} and UMM_{200} and the original CEGO_{ORI} are given in Table 2 for completeness. In this table, it can be seen that the quality of CEGO_{ORI} solutions is similar to the UMM_{100} version of UMM.

Runtime comparison. Table 1 also shows the runtime (in minutes) of CEGO_{INV} and UMM when both are run for 400 evaluations. For the same number of function evaluations, CEGO requires almost a day, whereas UMM never requires more than one hour, often much less than that. Details for other algorithm variants are given in Table 3. The average runtime for all the instances is around 0.3 minutes, 1.15 minutes and 4.5 minutes respectively for the UMM variants with 100, 200 and 400 fitness evaluations. For CEGO, in both versions, the average runtime for 400 evaluations is more than 22 hours. This is 300 times slower than UMM for the same number of function evaluations. Moreover, the runtime of UMM scales well for larger instances since its computational complexity is $O(n \log n)$ at each iteration.

6 CONCLUSIONS

In this paper, we have introduced UMM, a population-based probabilistic algorithm based on an unbalanced Mallows model. The algorithm is designed for expensive black-box combinatorial problems on permutation landscapes when the budget of fitness evaluations is severely limited (here, up to 100, 200 and 400 evaluations). Although we were inspired by previous work on ACO for expensive black-box combinatorial problems [16], UMM is, to the best of our knowledge, the first EDA specifically designed for such problems.

We also discuss the importance of the representation of permutations in applied mathematics problems. Permutations can bee seen as rankings or orderings of items and this difference is often neglected. Based on this differentiation, we propose an improvement for CEGO over the original version. The results of UMM clearly improve over the original CEGO and they are competitive with the improved version of CEGO proposed here.

In addition, UMM is specially well-suited for budget-limited scenarios that are still time-sensitive, e.g., when the overhead incurred by the optimizer should be no more than a few second per fitness evaluation (no more than one hour for 400 evaluations), due to its quasi-linear time complexity. By comparison, CEGO may require several minutes per fitness evaluation (around 22 hours for 400 evaluations), with the time increasing non-linearly with permutation size and larger budgets. Hence, UMM is a computationally feasible alternative for relatively large problem sizes.

The current version of UMM uses the Kendall's-τ distance and, therefore, it is appropriate for problems where the pairwise ordering among the items is relevant (as opposed to the actual exact position of each item in the permutation), such as the Linear Ordering Problem and the Permutation Flow-shop Scheduling. However, we are certain that it is possible to extend UMM to other distance metrics such as Hamming, which will allow us to dynamically select among various distance metrics for an unknown black-box permutation landscape. In addition, a more detailed analysis would be needed of the parameters (r_1 and r_2) of UMM to provide either generally good static values or an online adaptation approach. Finally, an even more diverse range of problems would be needed to understand the behavior of UMM on real-world black-box combinatorial landscapes.

The results presented here show that, despite the intrinsic difficulty of the expensive black-box combinatorial scenario, there are still significant advances yet to be made. Thus, we hope that this paper will motivate further research.

Reproducibility. Source code, datasets and scripts necessary to reproduce the results are available at doi: 10.5281/zenodo.4500974.

ACKNOWLEDGMENTS

We would like to thank Hao Wang (Leiden University) for pointing us to the arguments of [8]. This article is based upon work from COST Action CA15140 'Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)' supported by COST (European Cooperation in Science and Technology). M. López-Ibáñez is a "Beatriz Galindo" Senior Distinguished Researcher (BEAGAL 18/00053) funded by the Ministry of Science and Innovation of the Spanish Government. This work is also partially funded by the Elkartek program (Basque Government) and the Industrial Chair "Machine Learning for Big Data" from Télécom Paris, France.

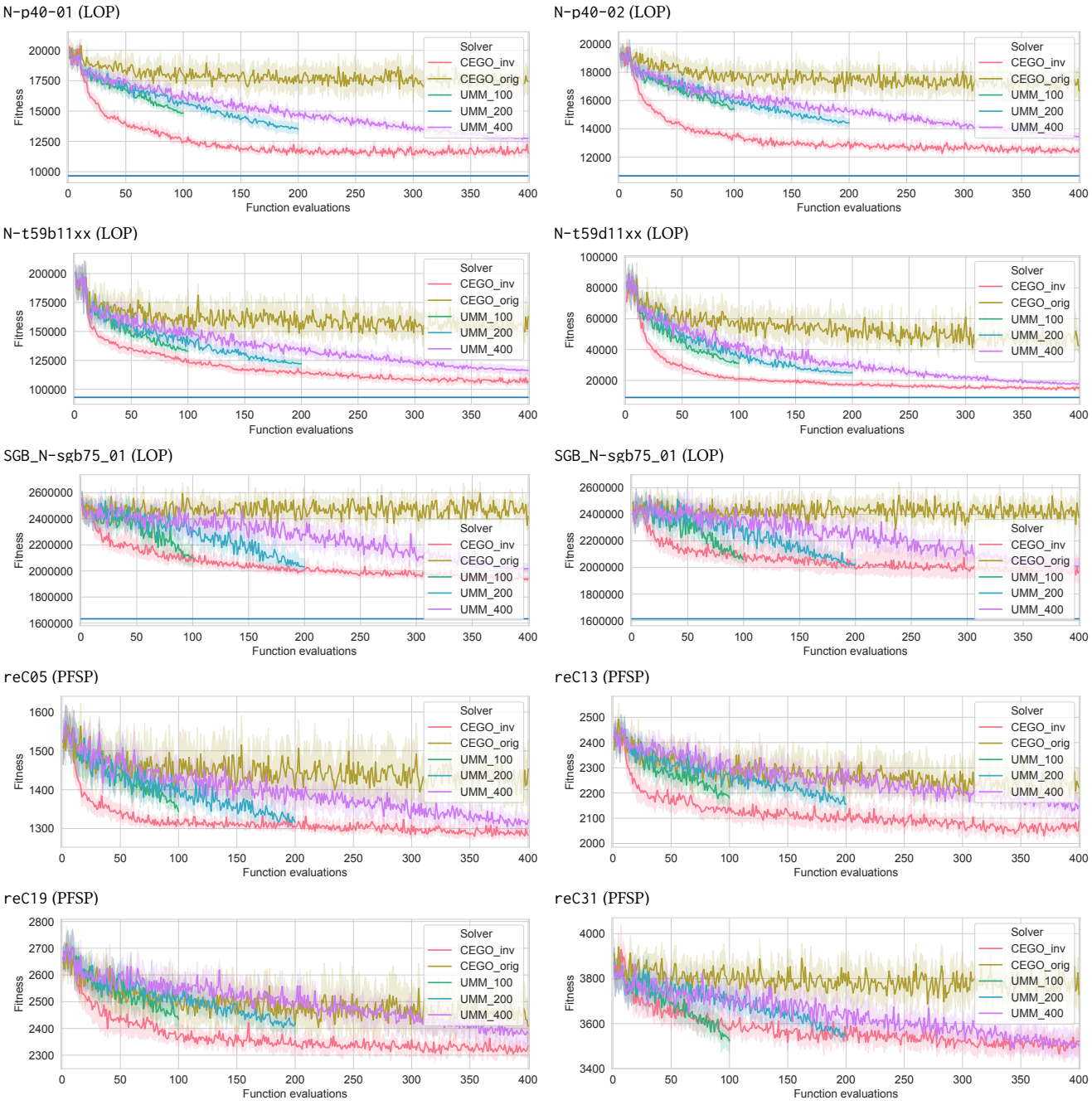

Figure 2: Mean fitness (and standard deviation) of each solution evaluated by each algorithm on various instances.

Table 1: Mean fitness (and standard deviation) of the best solution found at the end of each run, over 10 independent runs for each problem instance. The 95% confidence interval corresponds to the two independent samples t-test on the mean difference between the fitness of CEGO_{INV} minus the one of UMM_{400}. Mean runtime is shown in hours.

		Mean fitness (sd)		95% CI of the mean difference	Mean runtime	
Problem	**Instance**	**CEGO_{INV}**	**UMM_{400}**		**CEGO_{INV}**	**UMM_{400}**
LOP	N-p40-01	10496.6 (116.8)	12369.1 (389.9)	[-2157.1, -1587.9]	19.9	0.3
	N-p40-02	11714.7 (84.7)	13160.5 (454.8)	[-1773.5, -1118.1]	20.1	0.3
	N-t59b11xx	102408.0 (3800.2)	114075.5 (5389.2)	[-16084.2, -7250.8]	21.1	0.4
	N-t59d11xx	12692.2 (1097.9)	16662.9 (2155.3)	[-5618.5, -2322.9]	21.2	0.4
	N-sgb75.01	1893533.2 (28594.6)	1992059.9 (40235.7)	[-131577.1, -65476.3]	33.2	1.0
	N-sgb75.02	1895801.8 (82847.9)	1967132.3 (49228.1)	[-136421.2, -6239.8]	33.3	1.1
PFSP	rec05	1276.2 (15.1)	1276.3 (8.4)	[-11.8, 11.6]	18.3	0.1
	rec13	1992.9 (27.6)	2057.8 (22.8)	[-88.8, -41.0]	18.5	0.1
	rec19	2264.5 (45.0)	2304.8 (37.7)	[-79.4, -1.2]	18.2	0.2
	rec31	3415.5 (62.1)	3419.3 (37.0)	[-52.6, 45.0]	23.1	0.5

Table 2: Mean fitness, over 10 independent runs for each problem instance, of the best solution found at the end of each run of the original CEGO (CEGO_{ORI}) and its improved version (CEGO_{INV}) and three variants of UMM that differ in the budget of evaluations $m \in \{100, 200, 400\}$. The runs of CEGO used a budget of $m = 400$.

Problem	**Instance**	CEGO_{INV}	CEGO_{ORI}	UMM_{400}	UMM_{200}	UMM_{100}
LOP	N-p40-01	10496.6	15027.7	12453.7	1.336170e+04	14649.9
	N-p40-02	11714.7	15150.2	13191.9	1.420900e+04	15206.2
	N-t59b11xx	102408.0	126204.0	114726.5	1.202020e+05	132039.5
	N-t59d11xx	12692.2	30199.8	16775.4	2.352089e+04	29884.1
	N-sgb75.01	1893533.2	2095899.6	1983521.2	2.004006e+06	2046895.1
	N-sgb75.02	1895801.8	2069286.9	1973083.9	2.001244e+06	2039356.1
PFSP	rec05	1276.2	1318.7	1282.1	1.287000e+03	1320.2
	rec13	1992.9	2096.0	2077.7	2.108500e+03	2131.6
	rec19	2264.5	2328.4	2318.5	2.352100e+03	2386.5
	rec31	3415.5	3512.3	3420.5	3.490900e+03	3474.3

Table 3: Mean runtime in minutes, over 10 independent runs for each problem instance, of the original CEGO (CEGO_{ORI}) and its improved version (CEGO_{INV}) and three variants of UMM that differ in the budget of evaluations $m \in \{100, 200, 400\}$. The runs of CEGO used a budget of $m = 400$.

Problem	**Instance**	CEGO_{INV}	CEGO_{ORI}	UMM_{400}	UMM_{200}	UMM_{100}
LOP	N-p40-01	1195.90	1170.81	3.12	0.85	0.21
	N-p40-02	1207.24	1183.09	3.11	0.83	0.20
	N-t59b11xx	1265.98	1240.36	3.78	0.95	0.23
	N-t59d11xx	1269.93	1252.89	3.93	0.97	0.25
	N-sgb75.01	1994.22	2004.70	10.88	2.73	0.72
	N-sgb75.02	1995.09	1975.41	10.88	2.76	0.69
PFSP	rec05	1099.63	917.81	0.80	0.20	0.10
	rec13	1109.51	1043.08	0.85	0.20	0.10
	rec19	1090.06	1087.38	1.76	0.47	0.10
	rec31	1387.09	1367.26	4.77	1.28	0.32

REFERENCES

[1] Ali, A., Meilă, M.: Experiments with Kemeny ranking: What works when? Mathematical Social Science **64**(1), 28–40 (Jul 2012)

[2] Anstreicher, K., Brixius, N., Goux, J.P., Linderoth, J.: Solving large quadratic assignment problems on computational grids. Mathematical Programming Series B **91**(3), 563–588 (Feb 2002)

[3] Arza, E., Ceberio, J., Pérez, A., Irurozki, E.: Approaching the quadratic assignment problem with kernels of mallows models under the hamming distance. In: López-Ibáñez, M., Auger, A., Stützle, T. (eds.) GECCO'19 Companion, ACM Press, New York, NY (2019), ISBN 978-1-4503-6748-6

[4] Caragiannis, I., Procaccia, A.D., Shah, N.: When do noisy votes reveal the truth? In: Kearns, M.J., McAfee, R.P., Tardos, É. (eds.) Proceedings of the Fourteenth ACM Conference on Electronic Commerce, pp. 143–160, ACM Press, New York, NY (2013)

[5] Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A.: A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. IEEE Transactions on Evolutionary Computation **18**(2), 286–300 (2014)

[6] Coppersmith, D., Fleischer, L.K., Rurda, A.: Ordering by weighted number of wins gives a good ranking for weighted tournaments. ACM Transactions on Algorithms **6**(3), 1–13 (Jul 2010)

[7] Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: Shen, V.Y., Saito, N., Lyu, M.R., Zurko, M.E. (eds.) Proceedings of the Tenth International World Wide Web Conference, WWW 10, pp. 613–622, ACM Press, New York, NY (2001), ISBN 1-58113-348-0

[8] Eriksson, D., Pearce, M., Gardner, J., Turner, R.D., Poloczek, M.: Scalable global optimization via local Bayesian optimization. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems (NIPS 32), pp. 5496–5507 (2019)

[9] Fernández, S., Álvarez, S., Díaz, D., Iglesias, M., Ena, B.: Scheduling a galvanizing line by ant colony optimization. In: Dorigo, M., et al. (eds.) Swarm Intelligence, 9th International Conference, ANTS 2014, Lecture Notes in Computer Science, vol. 8667, pp. 146–157, Springer, Heidelberg, Germany (2014)

[10] Fligner, M.A., Verducci, J.S.: Distance based ranking models. Journal of the Royal Statistical Society: Series B (Methodological) **48**(3), 359–369 (1986)

[11] Forrester, A.I.J., Keane, A.J.: Recent advances in surrogate-based optimization. Progress in Aerospace Sciences **45**(1-3), 50–79 (2009)

[12] Irurozki, E., Calvo, B., Lozano, J.A.: PerMallows: An R package for mallows and generalized mallows models. Journal of Statistical Software **71** (2019), ISSN 15487660

[13] Irurozki, E., Lobo, J., Perez, A., Del Ser, J.: Rank aggregation for non-stationary data streams. Arxiv preprint arXiv: (2020), submitted

[14] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization **13**(4), 455–492 (1998)

[15] Moraglio, A., Kattan, A.: Geometric generalisation of surrogate model based optimization to combinatorial spaces. In: Merz, P., Hao, J.K. (eds.) Proceedings of EvoCOP 2011 – 11th European Conference on Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol. 6622, pp. 142–154, Springer, Heidelberg, Germany (2011)

[16] Pérez Cáceres, L., López-Ibáñez, M., Stützle, T.: Ant colony optimization on a limited budget of evaluations. Swarm Intelligence **9**(2-3), 103–124 (2015)

[17] Romero, P.A., Krause, A., Arnold, F.H.: Navigating the protein fitness landscape with Gaussian processes. Proceedings of the National Academy of Sciences **110**(3), E193–E201 (Dec 2012)

[18] Zaefferer, M., Stork, J., Bartz-Beielstein, T.: Distance measures for permutations in combinatorial efficient global optimization. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014, Lecture Notes in Computer Science, vol. 8672, pp. 373–383, Springer, Heidelberg, Germany (2014)

[19] Zaefferer, M., Stork, J., Friese, M., Fischbach, A., Naujoks, B., Bartz-Beielstein, T.: Efficient global optimization for combinatorial problems. In: Igel, C., Arnold, D.V. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2014, pp. 871–878, ACM Press, New York, NY (2014)