Model-based Gradient Search for Permutation Problems

ANONYMOUS AUTHOR(S)*

Global random search algorithms are characterized by using probability distributions to optimize problems. Among them, generative methods iteratively update the distributions by using the observations sampled. For instance, this is the case of the well-known Estimation of Distribution Algorithms. Although successful, this family of algorithms iteratively adopts numerical methods for estimating the parameters of a model or drawing observations from it. This is often a very time-consuming task, especially in permutation-based combinatorial optimization problems.

In this work, we propose using a generative method, under the model-based gradient search framework, to optimize permutation-coded problems and address the mentioned computational overheads. To that end, the Plackett-Luce model is used to define the probability distribution on the search space of permutations. Not limited to that, a parameter-free variant of the algorithm is investigated. Conducted experiments, directed to validate the work, reveal that the gradient search scheme produces better results than other analogous competitors, reducing the computational cost and showing better scalability.

CCS Concepts: • **Mathematics of computing → Probabilistic algorithms**; **Combinatorial optimization**; **Permutations and combinations**.

Additional Key Words and Phrases: Gradient search, permutation, probability distribution, combinatorial problem, parameter-free

ACM Reference Format:
Anonymous Author(s). 2018. Model-based Gradient Search for Permutation Problems. *J. ACM* 37, 4, Article 111 (August 2018), 25 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In the last few decades, developing *Global Random Search (GRS)* algorithms [Zhigljavsky 1991] has been a strong alternative to approach optimization problems, either in the continuous or combinatorial domains. In a formal scheme of GRS, the algorithm involves some iterations and, at each iteration t, a "suitably" constructed distribution P_t is sampled. The construction of the distribution P_t, and especially its suitability to get solutions of interest for the problem at hand, is a critical part of GRS algorithms that conditions their behavior. In this sense, an important means of constructing efficient global (non-uniform) random search algorithms relies on the idea that, in the vicinity of *good* solutions (of a problem), it is possible to find *better* solutions. The corresponding methods, which go under the name of *generation methods*, consist of sequential multiple sampling of probability distributions where the good solutions drawn from P_t at iteration t are used to construct the distribution P_{t+1} at iteration $t + 1$.

Although, neither the term *Global Random Search (GRS)* nor *generation methods* are frequently referenced, these methods form the essence of numerous heuristic, metaheuristic, and machine learning-based random search algorithms. In fact, we find a vast number of algorithms that fall within the family of *generation methods*. To name a recent framework, evolutionary algorithms have a consolidated trajectory using probability distributions, and Ant Colony Optimization (ACO) [Dorigo et al. 2006], Estimation of Distribution Algorithms (EDAs) [Larrañaga and Lozano

0004-5411/2018/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

2001] or Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995] are some paradigmatic examples. Similarly, in machine learning, we find algorithms such as Bayesian Optimization [Brochu et al. 2010] that iteratively update a Gaussian Process to build a surrogate of the function to optimize.

However, using the algorithms above implies frequently large computational overheads. For instance, implementing EDAs that consider probability models with some graph structure (such as Bayesian or Markov Networks), or models for which exactly calculating the parameters is in many cases impossible. As a result, numerical methods to approximate the Maximum Likelihood Estimators (MLE) are used. Similarly, in the Bayesian Optimization paradigm, at each iteration, the algorithm updates a Gaussian Process model which requires inverting a $m \times m$ samples-matrix whose complexity is known to be $O(m^3)$ (note that the number of samples m increases monotonically). It would not be a problem at all unless both algorithm types are iterative, and the time-consuming steps are repeated numerous times, generating large computational overheads. In fact, such overheads make the algorithm impracticable when considering large problem sizes in EDAs, or numerous samples in Bayesian Optimization.

The renewed interest in Gradient Search (GS), principally motivated by their usage in Neural Network training [Kingma and Ba 2014; Ruder 2016], and also for optimizing continuous problems [Wierstra et al. 2014], has popularized this type of algorithm. Nevertheless, in the combinatorial domain, the application of GS has been limited to a few works. Clearly, the main reason is that, in a discrete space, there does not exist a well-defined notion of the gradient. Nevertheless, there have been proposals [Berny 2001; Ollivier et al. 2017; Zlochin et al. 2004] to apply a model-based approach that can enable the use of GS in the combinatorial case. The main idea consists of: (i) optimizing the expected objective value of a random variable defined over the discrete space, and (ii) defining the model underlying the random variable in terms of continuous parameters and such that its probability density function is differentiable. In this way, a gradient can be defined and computed over the parameters of the model.

Though this approach can be generally applied to practically any discrete search space, only applications to binary problems have been observed in the literature [Berny 2000; Malagò et al. 2011; Ollivier et al. 2017], and other discrete search spaces have been mostly ignored so far in the research.

In this paper, we aim to explore strategies which are different to the classical ones, but still remain under the GRS paradigm, and overcome the enumerated drawbacks. To that end, we revisit the preliminary work published in [Santucci et al. 2020] and formalize a Gradient Search (GS) framework to optimize permutation problems that resolve the limitations of the classical approaches. Specifically, we propose using the Plackett-Luce model and updating its parameters via gradient search, instead of MLE. Not limited to that, an adaptive version of the algorithm that is free of parameters is also proposed. The extensive experimentation carried out comparing GS to some reference algorithms (under the Linear Ordering Problem (LOP) [Martí and Reinelt 2011]), reveals that our proposal is able to produce good (even better) results, while solving the computational overheads already enumerated.

The remainder of the paper is organized as follows. In Section 2, we provide preliminaries and discuss related work on permutation problems and probability models for the permutation space. Afterward, in Section 3 we present the main scheme of the proposed gradient search methodology for permutation problems. The parameter-free procedures for the algorithmic parameters are introduced in Section 4. Then, a thorough experimental study that includes other algorithms from the literature is presented in Section 5. Finally, the work concludes in Section 6, which also provides some insights for future work.

2 PRELIMINARIES

In this section, the content related to the present work is summarized. For detailed explanations, we refer the interested readers to address the bibliographic references provided in the text.

2.1 Permutations, Probability Models, and Combinatorial Problems

Permutations. A permutation is a bijection of the set $\{1, \ldots, n\}$ onto itself and n is its size. We will use the Greek letters π, σ and γ to denote them. In algebra, the group of all possible permutations of size n is referred to as the *Symmetric Group* $\mathbb{S}_n$, and its cardinality is $n!$ [Diaconis 1988].

Permutations are among the richest combinatorial structures in combinatorics. Motivated principally by their versatility - ordered set of items, collection of disjoint cycles, rankings, transpositions, matrices or graphs - permutations appear in a vast range of domains, thus problems whose solutions are codified as permutations are quite common in combinatorial optimization. To name a few, the Travelling Salesman Problem, the Quadratic Assignment Problem, the Flowshop Scheduling Problem or the Linear Ordering Problem (LOP) fall within this range of problems [Ceberio 2014; Santucci et al. 2019].

Probability models. As mentioned in the introduction, researchers in combinatorial optimization have developed a great number of algorithms that, by means of probability models that induce a particular distribution, try to approximate near optimal solutions. That is, a probability value is assigned to each solution and the aim is to iteratively evolve such values in such a way that solutions that have high quality receive high probabilities. Consequently, the expectancy of sampling new solutions with high objective values is more likely. Although this approach is not novel in the field, and integer, binary or continuous problems have been successfully addressed in the past, learning and sampling probability distributions over the symmetric group is not so straightforward.

Modelling permutation data, either orderings or rankings[1], has been extensively addressed in the literature of probability and statistics since the beginning of the previous century. In this sense, authors have named a number of probability models to deal with ranking data. It is not the aim of this paper to enumerate each of the works proposed throughout the decades, however, there are some reference authors that have reviewed them, and studied their properties. For detailed studies, we refer the interested reader to Fligner and Verducci [1986, 1988], Critchlow et al. [1991], Marden [1996], and more recently, Meila et al. [2012] and Irurozki et al. [2019].

In general, probability models for permutations have been divided into two big families: (1) distance-based exponential models and (2) order statistic models. With respect to the first family, the most well-known is the Mallows model [Mallows 1957] and its generalized form [Fligner and Verducci 1986]. The Mallows model is analogous to the normal distribution over the domain of permutations. It is defined by two parameters, a central permutation σ_0 and a concentration parameter θ, and the probability of any permutation decays exponentially with its distance to σ_0, measured with a particular metric function d. The probability of any σ is then calculated as

$$P(\sigma|\sigma_0, \theta) \propto e^{-\theta d(\sigma_0, \sigma)}.$$

The Generalized Mallows model is an extension of that model in which it is required that the metric decomposes into a number of terms, and the model considers as many concentration parameters θ as terms in the decomposition. As noted in [Ceberio et al. 2015], choosing a suitable distance metric is critical when approaching any optimization problem. In this trend, Irurozki et al. [2019] reviewed, and also developed, efficient learning and sampling methods for the Kendall's-τ, Cayley, Ulam and Hamming distance metrics on permutations.

[1]We refer the interested reader to a detailed explanation by Marden [1996].

With regard to the second family of models, we find those inspired by the *order statistic model*, for which there are numerous references such as Thurstone [Thurstone 1927], Mosteller [Mosteller 1951] and Babington-Smith [Joe and Verducci 1993]. But probably the Bradley-Terry (BT) [Bradley and Terry 1952] and Plackett-Luce (PL) [Luce 1959; Plackett 1975] models are the most relevant. Parameterized by n positive weights $\mathbf{w} = (w_1, \ldots, w_n)$, one per each item in the permutation, under the BT model, the probability of item i to be preferred to item j is calculated as

$$P(i \prec j) = \frac{w_i}{w_i + w_j},$$

where w_i and w_j denote the weights associated to the items i and j. The preference relation is translated to the permutation as the preference for $\sigma(i) < \sigma(j)$, i.e., better rank.

Subsequently, the probability rule of the BT model is obtained by extending the previous expression to all pairwise comparisons within a permutation as

$$P(\sigma|\mathbf{w}) \propto \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} \frac{w_{\sigma(i)}}{w_{\sigma(i)} + w_{\sigma(j)}}.$$

The PL model is parameterized as the BT model, but follows a different strategy to generate the probability distribution. Specifically, w_i denotes the preference of item i to appear in a top rank and, the higher the rank, the more likely it is. So, the probability of an item i to appear at the top rank, is calculated as $p_i = w_i/(w_1 + \ldots + w_n)$. If we sample such distribution to choose an item for the first position of the permutation, we remove it from the eligible set of items, and we repeat the process, then, we are choosing the item for the second position of the permutation. Continuing through the iterations, leaving aside the items already chosen in previous stages, then it is possible to sample a permutation from $\mathbf{w}$. Such process is summarized in Eq. 1.

$$P(\sigma|\mathbf{w}) = \prod_{i=1}^{n-1} \frac{w_{\sigma(i)}}{\sum_{j=i}^{n} w_{\sigma(j)}} \tag{1}$$

By intuition, the vector $\mathbf{w}$ can be normalized to sum 1, so that w_i becomes the probability that the item i is most preferred among the full set of items. The mode of the PL distribution is the permutation that sorts the weights in descending order.

Either the Mallows and Generalized Mallows models [Ceberio et al. 2014], and the Bradley-Terry and Plackett-Luce models [Alza et al. 2018] have been used with optimization purposes in the framework of EDAs. However, from these references, we see that the estimation of models and sampling imply very costly processes. In fact, in the first two, learning the MLE of the concentration parameters requires running numerical methods such as Newton-Raphson, and similarly, the Maximization-Minorization algorithm is required in the second two. Not only that, in the case of BT, Metropolis-Hastings algorithms are run for sampling non-biased solutions. As a result, although the models can be successfully fitted to the data, the large overheads introduced at each iteration of the algorithms makes them unlikely to be applied in permutation problems[2], and usually, evaluation-intensive heuristics are preferred.

Case of study. With illustrative purposes, in this paper we have considered the Linear Ordering Problem (LOP) as a case of study. Given a matrix $\mathbb{B} = [b_{ij}]_{n \times n}$ of numbers, the LOP consists of finding the simultaneous permutation of rows and columns such that the sum of the entries allocated in positions above the main diagonal is maximized. Formally, for any permutation solution σ, its

[2]This drawback equally appears in EDAs that learn probabilistic graphical models, or other complex structure models. However, the contribution of this paper is focused on permutation problems, and therefore, that review has been considered out of the scope of the paper.

objective value is calculated as $f(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{\sigma(i)\sigma(j)}$ where n is the number of rows/columns of the matrix, and also the number of items of σ. Then, any permutation $\sigma \in \mathbb{S}_n$, describes a candidate solution, as a result, the optimization becomes a challenging task with $n!$ potential solutions in the search space. In fact, it was proved to be an NP-hard problem many decades ago by Garey and Johnson [1979]. A reference works on this problem is [Martí et al. 2012].

2.2 Gradient Search Optimization

Gradient Search. Optimizing objective or loss functions through gradient-based search algorithms is a crucial task in the field of machine learning [Kingma and Ba 2014; Ruder 2016] and also in many disparate computational problems such as, for instance, those addressed in [Hauswirth et al. 2016; Ma and Huang 2007; Wang 2008].

Given a differentiable objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ that, without loss of generality, is required to be maximized, the most basic Gradient Search (GS) technique – *gradient ascent* – tries to maximize f by initially guessing a random solution $x_0 \in \mathbb{R}^n$ and then iteratively updating it by taking steps in the direction of steepest ascent according to

$$x_{t+1} \longleftarrow x_t + \eta \nabla f(x_t), \tag{2}$$

where $\nabla f(x_t) \in \mathbb{R}^n$ is the gradient of f at the point $x_t \in \mathbb{R}^n$, while $\eta > 0$ is the learning rate parameter, which regulates the convergence speed of GS and has to be carefully set by the practitioner.

It is easy to prove that the iterative application of the update rule in Eq. (2) reaches a solution which is only local optimal. Anyway, there are many cases where this is not a limitation. For instance, in machine learning applications, the true gradient is usually approximated by considering a random sample from a dataset. This makes the gradient computation stochastic and may allow the algorithm to escape from local optima [Kleinberg et al. 2018]. Moreover, many other modifications to the plain GS approach have been recently proposed for circumventing local optima (see e.g., [Ruder 2016] for a review).

Model-based Gradient Search. Evolution Strategies (ESs) [Beyer 1995; Beyer and Schwefel 2002] can be seen as a form of gradient search which maximizes the expected objective value of a probability model defined over the domain of an objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ that, contrary to the previous case, requires being neither differentiable nor analytically defined.

Formally, given a probability distribution over the solutions space $\mathbb{R}^n$, parameterized by $\theta \in \mathbb{R}^m$, whose probability density function is denoted by $\phi_\theta : \mathbb{R}^n \longrightarrow \mathbb{R}_{\geq 0}$ and such that it is differentiable with respect to θ, [3] then a model-based GS scheme aims at maximizing $J(\theta) = \mathbb{E}[f(x)]$, whose gradient with respect to the model's parameters is

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int_{\mathbb{R}^n} f(x)\phi_\theta(x)\,dx = \\ &= \int_{\mathbb{R}^n} f(x)\nabla_\theta\phi_\theta(x)\,dx = \\ &= \int_{\mathbb{R}^n} f(x)\nabla_\theta\phi_\theta(x)\frac{\phi_\theta(x)}{\phi_\theta(x)}\,dx = \\ &= \int_{\mathbb{R}^n} f(x)\left[\nabla_\theta \log\phi_\theta(x)\right]\phi_\theta(x)\,dx = \\ &= \mathbb{E}\left[f(x)\nabla_\theta \log\phi_\theta(x)\right], \end{aligned} \tag{3}$$

where the first three equivalences are due to simple mathematical rules, the fourth line is due to the so-called "log derivative trick", while the last equivalence is the definition of the expectation (taken over the solution space $\mathbb{R}^n$).

Therefore, even if the proper objective function $f(x)$ is not differentiable or not analytically defined, the differentiability of $\phi_\theta(x)$ allows the use of the GS scheme previously seen to iteratively update θ in order to maximize $J(\theta)$ that, as a sort of side effect, optimizes $f(x)$ as well. In fact, it is

[3]Note that, being ϕ_θ a probability density function, we have that $\int_{x\in\mathbb{R}^n} \phi_\theta(x) = 1$ and $\phi_\theta(x) \geq 0$ for all $x \in \mathbb{R}^n$.

easy to see that $J(\theta)$ is maximized when all the probability density is concentrated in (an arbitrarily small neighborhood of) the solution $x^* = \arg\max f(x)$.

Practically, an estimate of the gradient in Eq. (3) is obtained by sampling λ solutions $x_1, \dots, x_\lambda \in \mathbb{R}^n$ from ϕ_θ and then averaging as follows:

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{i=1}^{\lambda} f(x_i) \nabla_\theta \log \phi_\theta(x_i). \tag{4}$$

Eq. (4) introduces a further GS hyperparameter – the sample size $\lambda \in \mathbb{N}^+$ – and, by making the gradient computation stochastic, also allows a better resilience to local optima.

Most of the $(1+\lambda)$-ES schemes in the literature [Bäck et al. 2013; Hansen and Ostermeier 1996; Li et al. 2020] are (explicitly or implicitly) based on this approach. Usually, the probability model takes the form of a (univariate or multivariate) Gaussian distribution, while sporadically a fat-tailed distribution such as the Cauchy distribution is adopted (see e.g. [Yao and Liu 1997]). Moreover, a fitness-based selection operator is often applied to restrict the set of sampled solution used to estimate the gradient [Bäck et al. 2013].

Model-based Gradient Search for discrete spaces. The model-based GS approach can be straightforwardly extended to a combinatorial optimization problem defined over a discrete solution space Ω and whose objective function to maximize has the form $f : \Omega \longrightarrow \mathbb{R}$. What is required is a suitable probability model over Ω, parameterized by a vector $\theta \in \mathbb{R}^m$ and such that its probability mass function $p_\theta : \Omega \longrightarrow [0, 1]$ is differentiable with respect to θ.[4]

Analogously to the continuous case, the objective function maximized by GS is the expected objective value

$$J(\theta) = \mathbb{E}[f(x)] = \sum_{x \in \Omega} f(x) p_\theta(x), \tag{5}$$

whose true and estimated gradients are as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_\Omega \left[f(x) \nabla_\theta \log p_\theta(x) \right] \approx \frac{1}{\lambda} \sum_{i=1}^{\lambda} f(x_i) \nabla_\theta \log p_\theta(x_i). \tag{6}$$

Eq. (6) exploits very similar relations to those provided in Eqs. (3) and (4). Moreover, all the considerations previously made for the continuous case are valid also in the combinatorial case. The only aspect to note is that, though in a finite combinatorial domain it is theoretically possible to precisely compute the true gradient, the usually prohibitive size of Ω suggests that we approximate it as before, i.e., by computing an estimate of the gradient on the basis of λ samples $x_1, \dots, x_\lambda \in \Omega$ drawn from the probability mass p_θ — as indicated in the right part of Eq. (6).

For the sake of clarity, we provide the pseudocode of the model-based GS for combinatorial optimization in Algorithm 1, from which it is easy to see that an implementation of GS for the discrete space Ω at hand mainly requires:

(1) a possibly unconstrained formal representation for the model parameters vector θ_t,
(2) a procedure for sampling a solution from the probability model represented by θ_t (line 7), and
(3) a procedure for calculating the derivatives of the log-probability of a solution (line 9).

The very first attempts at using approaches similar to GS for optimizing discrete objective functions were proposed in [Berny 2000, 2001, 2002; Zlochin et al. 2004], while a proper systematization was introduced in [Malagò et al. 2011] and [Ollivier et al. 2017], both of which also extend the approach to the concept of *natural gradient*. However, all these works only address discrete problems defined over bit-strings – arguably the most simple (and used) discrete space

[4]Since it is a probability mass function, we have that $\sum_{x \in \Omega} p_\theta(x) = 1$ and $p_\theta(x) \geq 0$ for all $x \in \Omega$.

Algorithm 1 Gradient Search for Combinatorial Optimization

```
1: function GS($f : \Omega \rightarrow \mathbb{R}$, $\eta \in \mathbb{R}^+$, $\lambda \in \mathbb{N}^+$)
2:     initialize $\theta_0$ in such a way that probabilities are uniformly distributed
3:     $x^*$ maintains the best solution so far
4:     $t \leftarrow 0$
5:     while stopping criterion is not met do
6:         for $i \leftarrow 1$ to $\lambda$ do
7:             sample $x_i \in \Omega$ by drawing it from $p_{\theta_t}$
8:             evaluate $f(x_i)$ and update $x^*$ if improvement found
9:             calculate $\nabla_{\theta_t} \log p_{\theta_t}(x_i)$
10:        end for
11:        calculate $\nabla_{\theta_t} J(\theta_t)$ according to Eq. (6)
12:        $\theta_{t+1} \leftarrow \theta_t + \eta \nabla J(\theta_t)$
13:        $t \leftarrow t + 1$
14:    end while
15:    return $x^*$
16: end function
```

in combinatorial optimization. To the best of our knowledge, the only piece of research which considers a model-based GS approach for permutation-based combinatorial optimization problems is the preliminary paper proposed in [Santucci et al. 2020].

Model-based Gradient Search with natural gradient. Natural Evolution Strategies (NES) have been introduced in [Wierstra et al. 2014] for continuous optimization problems and extends the idea of model-based gradient search by considering the *natural gradient* in place of the plain gradient. The use of the natural gradient in the context of search algorithms has been further investigated in [Ollivier et al. 2017], which introduces a framework, named Information Geometric Optimization (IGO), that tries to systematize and subsumes most of the gradient-based algorithms, including NES, but also other algorithms such as the well known CMA-ES [Hansen 2016]. Moreover, [Ollivier et al. 2017] also mentions the possibility of applying the natural gradient search in the discrete solution space of binary problems. Another algorithm based on natural gradient has been proposed in [Malagò et al. 2011], which considers an exponential family of distributions over bit-strings. However, to the best of our knowledge, a practical NES implementation for the space of permutations has never been proposed before.

A NES-like algorithm behaves like a model-based GS, except that the plain gradient is left-multiplied with the inverse of the Fisher information matrix. Formally, given a solution space Ω and λ samples $x_1, \ldots, x_\lambda \in \Omega$ obtained according to a suitable probability model parameterized by $\theta \in \mathbb{R}^m$, the Fisher matrix and the update rule for θ are as follows:

$$F = \mathbb{E}_X \left[\nabla_\theta \log p_\theta(x) \cdot (\nabla_\theta \log p_\theta(x))^T \right] \approx \frac{1}{\lambda} \sum_{i=1}^{\lambda} \nabla_\theta \log p_\theta(x_i) \cdot (\nabla_\theta \log p_\theta(x_i))^T, \tag{7}$$

$$\theta_{t+1} \leftarrow \theta_t + \eta \left(F^{-1} \cdot \nabla J(\theta_t) \right). \tag{8}$$

The vector $F^{-1} \cdot \nabla J(\theta_t)$ in Eq. (8) is the natural gradient, which is known to allow the search to match the local space curvature imposed by the intrinsic characteristics of the chosen distribution model [Amari 1998]. This, in turn, avoids the instability of the search when the variance of the parameterized model decreases – a phenomenon that can be observed in plain model-based GS [Wierstra et al. 2014]. However, although the improved stability may decrease the number of iterations required to converge, the computation of the natural gradient introduces a certain overhead, as the inversion of a $n \times n$ matrix is known to cost $O(n^3)$ operations.

3 GRADIENT SEARCH METHODOLOGY FOR PERMUTATION PROBLEMS

Here we propose an implementation of the previously presented model-based gradient search schemes for the combinatorial space of permutations. In particular, we will use the acronyms GS and NES to denote the two variants of gradient search algorithms which use, respectively, the plain gradient and the natural gradient. To the best of our knowledge, apart from the preliminary work [Santucci et al. 2020], this is the first proposal of such schemes for the space of permutations.

We select the Plackett-Luce (PL) model described in Section 2.1 as probability distribution over permutations. In fact, the PL model is suitable for our purposes because: (i) its parameters are continuous, (ii) its probability mass function is efficiently differentiable, and (iii) it allows an efficient and unbiased sampling.

Anyway, it is worthwhile to note that, according to Eq. (1), the PL model requires positive weight parameters. Clearly, this constraint may be violated by the update rule of the parameters of a GS algorithm. Therefore, instead of maintaining the true PL weights $w \in \mathbb{R}^n_{>0}$, we maintain their logarithms

$$z = \log w. \tag{9}$$

In this way, the log-weights in $z \in \mathbb{R}^n$ have no constraint, while the original and positive PL weights can be easily recovered as $w = \exp z$. Hence, the probability of a permutation $\sigma \in \mathbb{S}_n$, given the log-weights z, is defined as

$$P(\sigma|z) = \prod_{i=1}^{n-1} \frac{\exp z_{\sigma(i)}}{\sum_{j=i}^{n} \exp z_{\sigma(j)}}. \tag{10}$$

Therefore, according to the general case formulation provided in Eq. (5), a model-based GS scheme optimizes the objective function $f : \mathbb{S}_n \to \mathbb{R}$ of the permutation problem at hand, by following the trajectories induced by the gradient of

$$J(z) = \mathbb{E}[f(\sigma)] = \sum_{\sigma \in \mathbb{S}_n} f(\sigma) P(\sigma|z). \tag{11}$$

A further modification to the algorithmic scheme is introduced in order to make it invariant to monotonic transformations of the objective function: a desirable property in a lot of real-world applications. Bearing this in mind, note that the gradient of the expected objective value, as shown in Eq. (6), clearly depends on the objective values. Therefore, we introduce a utility function which reshapes the objective values at every iteration by considering their ranks, thus making the algorithm invariant to monotonic transformation of the objective function.

The pseudocode of the "Plackett-Luce"-based GS and NES schemes is provided in Algorithm 2. The method requires as input: the objective function f to be (without loss of generality) maximized, the learning rate η, the sample size λ, and a utility function U.

In line 2, the z parameters are initialized to the same value, thus making the PL model equivalent to a uniform distribution over $\mathbb{S}_n$. Hence, the PL-based algorithms have no initialization bias. At any iteration of the main loop of lines 4–24, the z parameters are updated as follows. First, λ permutations are drawn from the current PL distribution (line 6) and their objective value is evaluated (line 7). Then, the gradients of the log-probabilities (line 8) and the utilities (line 10) are used to calculate the plain gradient in line 11. Thus, in the GS case, a δ vector is directly set to the plain gradient (line 13) while, in the NES case, δ is set by also considering the inverse of the Fisher matrix (lines 15–17). In line 19, the δ vector is scaled by the learning rate and used to update the PL parameters. Before moving to the next iteration, if any numerical problem was observed in the

Algorithm 2 Pseudocode of the "Plackett-Luce"-based GS and NES schemes

```
Input: $f : \mathbb{S}_n \to \mathbb{R},\ \eta \in \mathbb{R}^+,\ \lambda \in \mathbb{N}^+,\ U : \mathbb{R}^\lambda \to \mathbb{R}^\lambda$
 1: $t \leftarrow 0$
 2: $z_t \leftarrow (0, 0, \dots, 0)$                                    ▷ Uniform distribution
 3: $\sigma^*$ maintains the best permutation so far
 4: while stopping criterion is not met do
 5:     for $i \leftarrow 1$ to $\lambda$ do
 6:         draw $\sigma_i$ from the PL model parameterized by $z_t$
 7:         evaluate $f(\sigma_i)$ and update $\sigma^*$ if an improvement is found
 8:         $g_i \leftarrow \nabla_{z_t} \log P(\sigma_i | z_t)$                      ▷ See Eq. (12)
 9:     end for
10:     $u_1, u_2, \dots, u_\lambda \leftarrow U(f(\sigma_1), f(\sigma_2), \dots, f(\sigma_\lambda))$     ▷ See Algorithm 3
11:     $\nabla J(z_t) \leftarrow \frac{1}{\lambda} \sum_{i=1}^{\lambda} u_i g_i$
12:     if plain gradient case then
13:         $\delta \leftarrow \nabla J(z_t)$
14:     else natural gradient case
15:         $F \leftarrow \frac{1}{\lambda} \sum_{i=1}^{\lambda} g_i \cdot g_i^T$
16:         calculate the matrix inversion $F^{-1}$
17:         $\delta \leftarrow F^{-1} \cdot \nabla J(z_t)$
18:     end if
19:     $z_{t+1} \leftarrow z_t + \eta\delta$
20:     if numerical problems occurred then
21:         $z_{t+1} \leftarrow$ almost degenerate PL distribution with $\sigma^*$ as mode     ▷ Soft restart
22:     end if
23:     $t \leftarrow t + 1$
24: end while
25: return $\sigma^*$
```

current step, the z parameters are reinitialized by setting the best so far solution as mode of the PL distribution (line 21). Finally, the best permutation sampled is returned in line 25.

In the following subsections we describe: the efficient PL sampling procedure, the computation of the gradient of the PL log-probability, the objective value shaping scheme used to define the utility function and, finally, the soft restart mechanism used to overcome the numerical problems.

3.1 The PL sampling procedure

Sampling a permutation from a PL distribution can be efficiently performed by exploiting the so-called "Gumbel top-k trick" as outlined in [Kool et al. 2019].

Using this methodology, a permutation $\sigma \in \mathbb{S}_n$ can be sampled from a PL distribution parameterized by $z \in \mathbb{R}^n$ as follows. First, generate a vector v formed by n numbers taken uniformly at random from the interval $[0, 1]$, so $v \in [0, 1]^n$. Then, compute the standard Gumbel random variables $\varepsilon_i = -\log(-\log v_i))$, for $i = 1, \dots, n$, and finally set $\sigma = \arg\mathrm{sort}(z + \varepsilon)$.

Practically, the σ entries are set to the ranks of the PL weights perturbed by a Gumbel distributed noise. The ranks are obtained by sorting, thus the computational complexity of the sampling procedure is $\Theta(n \log n)$.

3.2 Gradient of the PL log-probability

In order to implement the PL-based gradient search schemes, a formula is required for computing the gradient of the PL log-probability (line 8 of Algorithm 2).

Given a permutation $\sigma \in \mathbb{S}_n$ and an unconstrained PL parametrization $z \in \mathbb{R}^n$, then, by using some calculus, it is possible to derive a formula for the $\sigma(i)$−th partial derivative of $\log P(\sigma|z)$, as follows:

$$\frac{\partial \log P(\sigma|z)}{\partial z_{\sigma(i)}} = 1 - \exp z_{\sigma(i)} \sum_{k=1}^{i} \frac{1}{\sum_{j=i}^{n} \exp z_{\sigma(j)}}. \tag{12}$$

These partial derivatives all together form the gradient $\nabla_z \log P(\sigma|z)$ required by line 8 of Algorithm 2. Interestingly, this computation can be efficiently implemented because, with simple bookkeeping and by following the order induced by σ, it is possible to calculate all the n entries of $\nabla_z \log P(\sigma|z)$ with $\Theta(n)$ time steps.

3.3 Utility function

On the basis of the preliminary experiments conducted in [Santucci et al. 2020], here we consider the *super linear* utility function as defined in Algorithm 3.

Algorithm 3 Super Linear utility function

```
1:  function U($f_1, f_2, \dots, f_\lambda$)
2:      $r_1, r_2, \dots, r_\lambda \leftarrow \arg\mathrm{sort}(f_1, f_2, \dots, f_\lambda)$ ▷ The sorting is from the best to the worst objective value
3:      $\mu \leftarrow \lambda/2$
4:      $T \leftarrow \sum_{i=1}^{\mu} \exp i$
5:      for $i \leftarrow 1$ to $\lambda$ do
6:          if $r_i \le \mu$ then
7:              $u_i \leftarrow \exp(\mu + 1 - r_i)/T$
8:          else
9:              $u_i \leftarrow 0$
10:         end if
11:     end for
12:     return $u_1, u_2, \dots, u_\lambda$
13: end function
```

Given the λ objective values $f_1, \dots, f_\lambda$, Algorithm 3 generates the corresponding utilities $u_1, \dots, u_\lambda$ as follows. The objective values are sorted from best to worst, thus that the rank $r_i \in \{1, \dots, \lambda\}$ of any objective value f_i is computed (line 2). Then, the $\mu = \lambda/2$ best samples receive a utility which is exponentially proportional to $\mu + 1 - r_i$ (line 7), while a null utility is assigned to the worst samples (line 9). Finally, note that the returned utilities sum up to 1.

Since utilities are used to weight the samples for the gradient computation at each single iteration of the main algorithm (see line 11 of Algorithm 2), this scheme allows us to:

- make the main algorithm invariant to monotonic transformation of the objective function to optimize (because utilities are calculated only on the basis of the objective value ranks);
- equally weight different iterations of the main algorithm (since utilities always sum up to 1), thus making the search less influenced by unfortunate iterations with large steps;
- increasingly award higher ranking samples (since utility values exponentially decay) and nullify the impact of bad samples (because worst samples receive null utilities);
- efficiently calculate the utilities with a computational cost of $\Theta(\lambda \log \lambda)$.

3.4 Soft restart

As iterations pass, the search naturally moves from the initial exploratory behaviour to a later exploitative behaviour. This is a phenomenon common to any iterative optimization heuristic that, in model-based gradient search schemes, results in the tendency of the probability model to converge towards a configuration where almost all the probability mass is concentrated on a single solution. When this happens, two issues may arise: (i) the search gets trapped into a local optimal configuration, and (ii) the PL weights get sparser and the computation of the log-probability gradient may result in numerical stability problems.

To address these issues, we devised a simple soft restart mechanism (see lines 20–22 of Algorithm 2). At every iteration of the algorithm, we verify if a computation resulted in a *inf* or *nan* value and, if so, the PL weights are reset to a configuration whose mode is the best so far permutation σ^* and such that

$$P(\sigma^*|w) > \left(1 - \frac{1}{s}\right)^{n-1}, \tag{13}$$

where $s > 1$ is a suitable lower bound between consecutive PL weights to be chosen. Note that inequality (13) shows that, as $s \to \infty$, the probability of the mode permutation tends to 1, thus concentrating the probability mass in a narrow neighborhood of σ^* for a suitable choice of s.

For the sake of clarity, in order to explain the Inequality (13) we use the classical parameterization w of the PL probability mass function as given in Eq. (1). Formally, we have $w_{\sigma^*(i)}/w_{\sigma^*(i+1)} \geq s$, with $s > 1$, for $i = 1, \dots, n$. This implies that any factor in the product of Eq. (1) is larger than $1 - 1/s$, thus the Inequality (13) follows straightforwardly.

Therefore, to have the probability guarantee of Inequality (13), it is enough to set the PL weights, following the order induced by σ^*, such that the ratio of consecutive weights is s. Moving to our unconstrained reparameterization of PL by log-weights, this translates to equally spacing the values $z_{\sigma^*(1)}, \dots, z_{\sigma^*(n)}$ such that $z_{\sigma^*(i)} - z_{\sigma^*(i+1)} = \log s$. Hence, we simplify the definition by selecting an interval $[lb, ub]$ for the z log-weights, such that:

$$z_{\sigma^*(i)} \leftarrow ub - (i-1)\left(\frac{ub - lb}{n-1}\right) \qquad \text{for } i = 1, \dots, n, \tag{14}$$

where the fractional factor is actually equivalent to $\log s$, and it is easy to see that $z_{\sigma^*(1)} = ub$ and $z_{\sigma^*(n)} = lb$ are the two extreme values, while the other log-weights are linearly assigned in the interval $[lb, ub]$.

After some experimentation, lb and ub were set to, respectively, -10 and 10, independently of n. This setting was not affected by numerical problems in our experimentation and it guarantees that the drawn permutations (in the next algorithm iteration) are similar but not equal to σ^*, thus avoid moderate search stagnation.

4 PARAMETER ADAPTATION

The proposed scheme has two (hyper)parameters: the learning rate $\eta \in \mathbb{R}^+$ and the sample size $\lambda \in \mathbb{N}^+$. While η impacts on the convergence speed of the search and its proneness to escape from stagnation states, λ has implications both on the learning accuracy of the PL model and on the efficiency of the search (counted as number of objective function evaluations).

Both the parameters can be tuned offline by a trial-and-error approach based on a given experimental design (as done, for instance, in [Santucci et al. 2020]). However, due to the variability of the characteristics of the problem instances, a parameter setting which works well for a particular problem instance may not be suitable for another. Moreover, even inside a single execution, there may be different temporal stages where different parameters settings fit well with the current search state, possibly in accordance with the incumbent configuration of the PL weights. In light of this, it is apparent how important an online adaptation scheme for the η and λ parameters is.

Having considered that, here we introduce online adaptation schemes for both the learning rate and the sample size. Therefore, we will denote with η_t and λ_t, respectively, the learning rate and the sample size at iteration t. The η_t adaptation is introduced in Section 4.1, while the λ_t adaptation is presented in Section 4.2.

4.1 Adaptation of the Learning Rate

The learning rate adaptation is based on the Cumulative Step-size Adaptation (CSA) scheme, which has been originally introduced in the context of Evolution Strategies based on Gaussian mutations [Chotard et al. 2012; Hansen and Ostermeier 1996; Ostermeier et al. 1994a,b]. The main idea of CSA is to increase or decrease the learning rate in order to speedup or slowdown the search when the previously taken steps in the search trajectory, respectively, agree or disagree on their direction.

Here, we introduce a particular version of CSA suited to the characteristics of the Plackett-Luce model. To the best of our knowledge, this is also the first CSA implementation for a non-Gaussian model.

After choosing a suitable initial learning rate η_0, the CSA version we propose dynamically sets η_t, at any iteration $t > 0$, on the basis of η_{t-1} and the search history up to iteration t, according to

$$\eta_t = \eta_{t-1} \exp\left(\frac{||\rho_t||}{\ell} - 1\right), \tag{15}$$

where: $||\cdot||$ denotes the Euclidean norm of a vector, $\rho_t \in \mathbb{R}^n$ is the *cumulative path* vector that synthesizes the search history up to iteration t, and ℓ is a constant representing the expected length of a random walk in $\mathbb{R}^n$.

Before formally defining ρ_t and ℓ, for the sake of understanding, we depict the rationale of Eq. (15). The argument of the exp function implements a sort of "graded comparison" between the observed length of the cumulative path, i.e., $||\rho_t||$, and the expected length under a hypothetical random search behaviour, i.e., ℓ. Therefore, when the previous search steps point towards the same or a similar direction, we will have $||\rho_t|| > \ell$, thus the exp argument is positive and Eq. (15) increases η_t with respect to its previous value. Conversely, when the previous search steps point towards opposite or almost opposite directions, we will have $||\rho_t|| < \ell$, thus the exp argument is negative and Eq. (15) decreases η_t with respect to its previous value. Basically, ℓ is used as a threshold value to which we compare the observed historical directions – synthesized by ρ_t – of the search. The rationale is to speed up the search when there is consensus over time towards a direction and, conversely, to slow down the movements when the search starts to appear chaotic.

Definition of the cumulative path vector. The cumulative path is initialized to the zero-vector, i.e., $\rho_0 = \mathbf{0}$. Then, by denoting the last movement vector with δ_t – i.e., $\delta_t = \nabla J(\tilde{w}_t)$ for the plain gradient version and $\delta_t = F^{-1} \cdot \nabla J(\tilde{w}_t)$ for natural gradient version – and its normalization with $\hat{\delta}_t$, i.e., $\hat{\delta}_t = \frac{\delta_t}{||\delta_t||}$, the cumulative path is iteratively updated according to

$$\rho_t = (1-r)\rho_{t-1} + \hat{\delta}_t, \tag{16}$$

where $r \in [0, 1]$ is a hyperparameter which regulates the weight of the previous cumulative path with respect to the last direction vector. In this way, all the directions of the search movements up to iteration t are accumulated in ρ_t.

Moreover, note that the importance of a direction vector exponentially decays with the iterations. In fact, as a consequence of Eq. (16), a direction vector has a mean lifetime of $1/r$ iterations, i.e., after $1/r$ iterations it accounts for no more than $1/e \approx 37\%$ of the cumulative path. For this reason, we set the hyperparameter r in such a way that the mean lifetime of a direction vector increases together with the dimensionality n thus, after some preliminary experiments, we chose $1/r = \sqrt{n} \implies r = 1/\sqrt{n}$.

Definition of the expected length of a random walk. Now, in order to finalize the definition of Eq. (15), we need to formally derive the value of the constant ℓ, i.e., the expected length of a random walk in $\mathbb{R}^n$ with exponentially decaying step-sizes.

Firstly, let us motivate why we follow this choice for the definition of ℓ. With this in mind, note that the cumulative path definition of Eq. (16) can be unrolled to

$$\rho_t = \sum_{i=0}^{t-1} (1-r)^i \hat{\delta}_{t-i}. \tag{17}$$

Hence, ρ_t is the weighted sum of t unit vectors with weights taken from the geometric progression with common ratio $1-r$. Therefore, the length of each vector accumulated in ρ_t is $(1-r)^i$, for $i = 0, \dots, t-1$. Since the directions of the vectors $\left\{\hat{\delta}_i \in \mathbb{R}^n\right\}$ are a consequence of the search behaviour so far, we can make decisions about the speed of the search by comparing the cumulative path vector with the expected length of a random walk formed by independent and uniformly distributed unit vectors $\{d_i \in \mathbb{R}^n : ||d_i|| = 1\}$, with $i = 0, \dots, t-1$, weighted as in Eq. (17).

To simplify the calculations, we extend the summation to an infinite number of terms[5] and we define ℓ as

$$\ell = \mathbb{E}\left[\left\|\sum_{i=0}^{\infty} (1-r)^i d_i\right\|\right]. \tag{18}$$

Hence, by squaring both sides of Eq. (18), it is possible to write:

$$\begin{aligned} \ell^2 &= \textstyle\sum_{i=0}^{\infty}(1-r)^{2i}\mathbb{E}\left[d_i \cdot d_i\right] + 2\sum_{i=0}^{\infty}\sum_{j=i+1}^{\infty}(1-r)^{i+j}\mathbb{E}\left[d_i \cdot d_j\right] = \\ &= \textstyle\sum_{i=0}^{\infty}(1-r)^{2i} = \\ &= \frac{1}{2r-r^2}, \end{aligned} \tag{19}$$

which can be explained as follows. The first equivalence exploits the linearity of the expected value operator and simple arithmetic calculations. The second equivalence derives from the facts that the dot product of a unit vector by itself is 1 and that two independent and identically distributed vectors are orthogonal in expectation, thus their dot product is 0. Lastly, it is easy to see that the expression in the second line is a geometric series with common ratio $(1-r)^2$ and this explains the last equivalence in Eq. (19). It is now straightforward to derive the value of ℓ as

$$\ell = \frac{1}{\sqrt{2r - r^2}}, \tag{20}$$

thus, by using the previously discussed setting $r = 1/\sqrt{n}$, we have $\ell = \left(\frac{2}{\sqrt{n}} - \frac{1}{n}\right)^{-\frac{1}{2}}$, which is a function of only the search space dimensionality n.

4.2 Adaptation of the sample size

A good strategy to adapt the sample size $\lambda \in \mathbb{N}^+$ is to regulate it on the basis of an indicator of the spread of the PL model. Intuitively, when the PL model is very spread and similar to a uniform distribution, a relatively large number of samples is required to have a better gradient estimation. Conversely, when the PL model is concentrated in a small neighborhood of its mode, then it is

[5]Regarding this, note that the weights $(1-r)^i$ quickly decay with $i \to \infty$, thus our approximation is reasonable.

likely that most of the samples are very similar or even identical in extreme situations, thus not helping very much to improve the gradient estimation and also wasting time for the evaluation of duplicate solutions.

Moving from these considerations, here we propose a sample size adaptation scheme which is based on a spread measure of the current configuration of the PL model. A good spread measure is the entropy of the PL distribution, but its computation is clearly infeasible. Fortunately, we can derive good indications about the spread of the PL model by only considering its current weights – or log-weights – at any iteration of the gradient search scheme.

Formally, given the log-weights $z \in \mathbb{R}^n$, we consider the probabilities of each item $i \in \{1, \dots, n\}$ being in the first position of the sampled permutation, that is given by

$$p_i = \frac{\exp z_i}{\sum_{j=1}^n \exp z_j}. \tag{21}$$

We also note that the vector $p \in \mathbb{R}^n$ represents a proper multinomial distribution since $\sum_{i=1}^n p_i = 1$ and $p_i \geq 0$, for $i = 1, \dots, n$. Therefore, we compute the normalized entropy $H \in [0, 1]$ of this multinomial distribution as usual:

$$H = -\frac{\sum_{i=1}^n p_i \log p_i}{\log n}. \tag{22}$$

We now argue that H is a good indicator of the true entropy of the PL distribution modeled by the log-weights z at hand. In fact, when all the weights are equal, it is easy to see that the PL model reduces to a uniform distribution over the permutations and that $H = 1$. Conversely, when the minimum ratio between a pair of PL weights tends to infinity, i.e., when $\min_{z_i \geq z_j} \{{}^{\exp z_i}/_{\exp z_j}\} \longrightarrow \infty$, we have that $H \longrightarrow 0$. Hence, since it is known that very sparse PL weights result in an almost-degenerate distribution concentrated around the mode permutation, this is in accordance with the true PL entropy.

Therefore, after choosing a suitable initial sample size λ_0, at any iteration t, we dynamically adapt the sample size λ_t on the basis of the entropy-like measure of the PL spread as follows:

$$\lambda_t = \lambda_{lower} + H_t \left(\lambda_{upper} - \lambda_{lower}\right), \tag{23}$$

where $H_t \in [0, 1]$ is the entropy-like indicator for the current iteration as given in Eq. (22), while λ_{lower} and λ_{upper} are, respectively, the lower and upper bound for the sample size. After a few preliminary experiments, we set $\lambda_{lower} = 10$ and $\lambda_{upper} = 1000$.

In this way, we have that the sample size is dynamically adapted inside the interval $[\lambda_{lower}, \lambda_{upper}]$ by moving towards λ_{lower} or λ_{upper} when, respectively, the PL model gets very concentrated or very spread.

5 EXPERIMENTAL STUDY

In what follows, a comprehensive experimental study is presented. It is the aim of this section to introduce a rigorous validation of the proposed gradient search algorithm, and compare it to other literature proposals that iteratively update probability distributions. In fact, we do not set out to obtain state-of-the-art results but to understand the behavior of the evaluated algorithms, and to identify the strong points, as well as weaknesses. To that end, taking the paper [Santucci et al. 2020] as starting point, we have extended the experimentation by considering larger instances, other state-of-the-art paradigms, and a deeper analysis in the convergence and scalability of the proposal.

5.1 Experimental setting

Case of study. In order to carry out the experimentation, we have chosen the Linear Ordering Problem as case study [Martí and Reinelt 2011]. Regarding the optimization instances, we have considered the well-known IO benchmark of 50 instances with sizes varying from 44 to 79, and the xLOLIB benchmark [Schiavinotto and Stützle 2003] with 78 instances of size $n = 150$ and 250, respectively[6].

Algorithms. We have included three algorithms in the comparison: the gradient search approaches using both the plain gradient (GS) and the natural gradient (NES) are contrasted to an estimation of distribution algorithm (EDA) [Ceberio et al. 2013] based on the Plackett-Luce model. Moreover, when the parameter-free version of GS and NES are adopted, we denote them by GS* and NES*. The parameters of GS and NES are tuned offline as described in Section 5.2, while for GS* and NES* the two adaptation schemes described in Section 4 are adopted. With this regard, note that the learning rate and sample size are initialized to the choice made in the parameter tuning and described in the next section, while η_t is also clipped in the range $[0.0001, 0.9]$.

Computing infrastructure. The experimentation was conducted on a cluster of 55 nodes, each one equipped with two Intel Xeon X5650 CPUs and 64GB of memory.

5.2 Tuning and impact of the parameters

In order to maximize the performance of the algorithms, it is important to properly set the parameters of GS, in this case, η (learning rate) and λ (sample size) parameters. To that end, we have performed a grid search for the two parameters, either for GS and NES. Particularly, the values $\{0.001, 0.01, 0.05, 0.1, 0.5, 0.75, 0.9\}$ for η and $\{10, 10^2, 10^3, 10^4\}$ for λ have been considered. In addition, five instances of different sizes from the IO benchmark, and eight instances from the xLOLIB benchmark have been chosen[7]. For each instance, algorithm and parameter combination, 10 repetitions were executed, each of which with a limit of $1000n^2$ function evaluations. Due to the different size of the instances, and the expected variation of the performance of the algorithms with their size, the tuning was performed separately for IO, xLOLIB $n = 150$ and xLOLIB $n = 250$. To statistically assess the differences among the different alternatives (combination of parameters), a Bayesian performance analysis was carried out [Rojas-Delgado et al. 2022][8]. The outcome of the analysis is summarized in Fig. 1 in the form of credibility intervals. In the y-axis of the plots the different parameter combinations are listed, and for each case, a credibility interval is depicted. Such interval, formed with a green dot (the expectancy) and a range of values, describes the probability of that combination being the best alternative among the compared ones. The values needed to build the intervals are obtained by sampling the posterior distribution of the Bayesian model computed.

For the GS algorithm, the analysis suggests that there is no combination that is superior to the rest, at least 8 alternatives overlap at probability 0.08, however, $\lambda = 100$ and $\eta = 0.5$, seems to be the preferred setting with the highest expected probability of winning for IO, and $\lambda = 100$ and $\eta = 0.05$, and $\lambda = 1000$ and $\eta = 0.1$ for $n = 150$ and $n = 250$, respectively. With regard to NES, $\lambda = 100$ and $\eta = 0.75$, is the best option for the IO benchmark, and $\lambda = 10$ and $\eta = 0.5$ for $n = 150$ and $n = 250$.

In addition, the raw results collected in the tuning of the parameters are plotted in Fig. 2 as scatter plots with x and y-axes depicting execution time and normalized f[9], respectively. Specifically, each

[6]Both benchmarks can be found at: https://grafo.etsii.urjc.es/optsicom/

[7]The instances included in the parameters tuning are *N-be75eec, N-stabu70, N-t59b11xx, N-tiw56n54, N-usa79, N-be75eec_150, N-be75eec_250, N-stabu1_150,N-stabu1_250, N-t59b11xx_150, N-t59b11xx_250, N-tiw56n54_150* and *N-tiw56n54_250.*

[8]A detailed procedure for the analysis is provided in [Rojas-Delgado et al. 2022].

[9]f is normalized by calculating the relative deviation with respect to the best known results reported in [Santucci and Ceberio 2020].

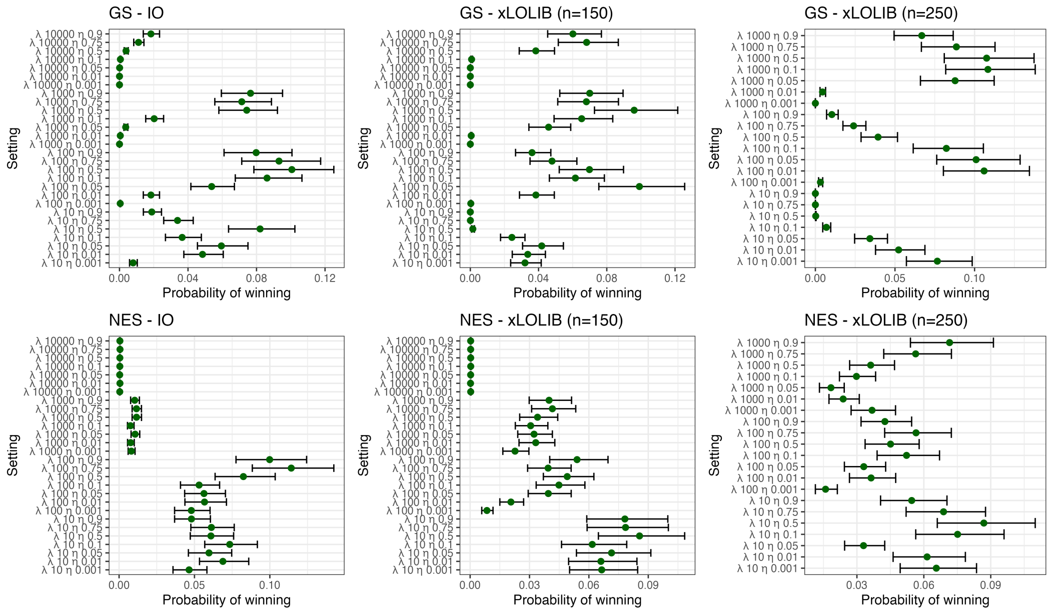

Fig. 1. Credibility intervals of the evaluated parameter settings for the GS and NES, for the IO benchmark, and the group of instances of size $n = 150$ and $n = 250$ from the xLOLIB benchmark. The intervals describe for each setting the probability of being the best option, based on the experimental data provided.

point in the plot represents a repetition of the algorithm, colors denote λ and the size of the points is proportional to η. Plots reveal that GS and NES are differently affected by the parameter settings. For GS, good results are subject to high η values, while, for NES, good results rely on the number of samples used, large values are preferred. In general, it seems that NES requires more execution time than GS, obtaining worse results.

Finally, for EDA, the parameters reported in the original paper [Ceberio et al. 2013] have been adopted.

5.3 Effectiveness and efficiency analysis

Using the parameters tuned in the previous section, 20 repetitions of the GS, EDA and NES algorithms have been performed on the mentioned benchmarks, and each algorithm was run a maximum number of $1000n^2$ evaluations. Results are summarized in Tables 1 and 2[10] as Median Relative Deviations (MRD) with respect to the best known results reported in [Santucci and Ceberio 2020].

According to the conducted experiments, GS obtained better results in 43 instances out of 50 instances in the IO benchmark, EDA did so in 7 instances, and NES did not succeed in any instance. In the case of the xLOLIB benchmark, for $n = 150$, GS was preferred in the great majority of 29 instances out of 39. EDA only outperformed GS in two instances. Again, NES was the worst performing strategy, obtaining errors three times higher to the best proposal. For $n = 250$, GS and EDA have similar results, being GS better (wins in 25 instances out of 39), and NES, again, is the worst algorithm.

[10]See Appendix A for Table 2.

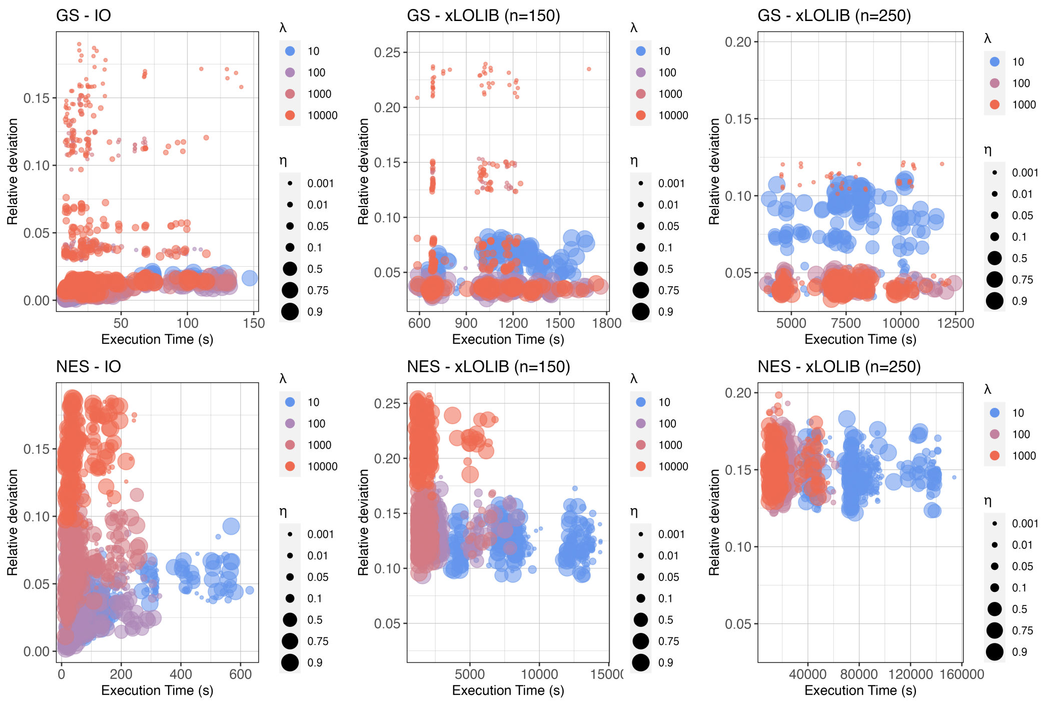

Fig. 2. Each of the executions performed for the different parameter settings and repetitions of the GS and NES on the IO and xLOLIB benchmarks is depicted. The x-axis is the execution time and the y-axis corresponds to the relative deviation of the objective value with respect to the best known results. The size of the points is proportional to the value of η. Colors denote the value of λ.

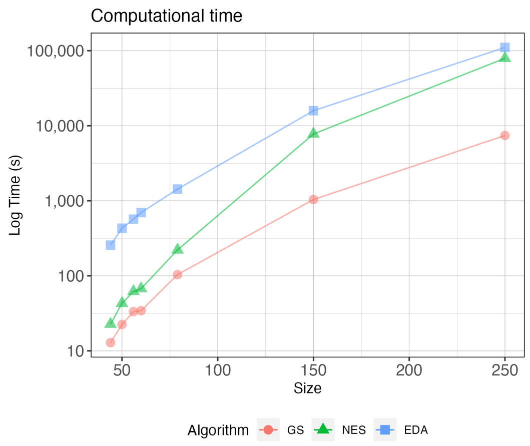

Fig. 3. Average execution time (log seconds) of each run of the GS (red), EDA (blue) and NES (green) algorithms for running $1000n^2$ evaluations.

Table 1. Results of the GS ($\lambda = 100, \eta = 0.5$), EDA and NES ($\lambda = 100, \eta = 0.75$) for the LOP instances in the IO benchmark. The Median Relative Deviations (MRD) measures of the values found across the 20 repetitions of the best known results are reported. Results in bold highlight the algorithm that obtained the lowest MRD. A maximum number of $1000n^2$ evaluations were performed by each of the algorithms.

Size	Instance	Best Known	GS	EDA	NES	Size	Instance	Best Known	GS	EDA	NES
50	N-be75eec	236464	**0.00955**	0.00986	0.01312	44	N-t70k11xx	60659200	**0.00108**	0.00311	0.00963
50	N-be75np	716994	**0.00185**	0.00425	0.00816	44	N-t70l11xx	25253	**0.00000**	0.00048	0.00160
50	N-be75oi	111171	**0.00219**	0.00409	0.01087	44	N-t70n11xx	52704	**0.00394**	0.00424	0.01541
50	N-be75tot	980516	**0.00165**	0.00421	0.01650	44	N-t70u11xx	21716400	0.00193	**0.00169**	0.02038
60	N-stabu70	362512	**0.00796**	0.01357	0.03051	44	N-t70w11xx	224319954	**0.00409**	0.00526	0.02003
60	N-stabu74	541393	**0.00624**	0.01080	0.02236	44	N-t70x11xx	283808865	**0.00254**	0.00277	0.01901
60	N-stabu75	553303	**0.00662**	0.01252	0.02628	44	N-t74d11xx	566089	0.00133	**0.00113**	0.02311
44	N-t59b11xx	209320	0.00454	**0.00155**	0.00933	44	N-t75d11xx	578304	**0.00205**	0.00323	0.02281
44	N-t59d11xx	147354	**0.00482**	0.01823	0.01086	44	N-t75e11xx	2739219	**0.00253**	0.00319	0.00660
44	N-t59f11xx	122520	**0.00032**	0.00214	0.00861	44	N-t75i11xx	63567735	**0.00080**	0.00163	0.01315
44	N-t59i11xx	8261545	**0.00020**	0.00208	0.00218	44	N-t75k11xx	108844	**0.00104**	0.00189	0.01041
44	N-t59n11xx	20928	**0.00287**	0.00583	0.01121	44	N-t75n11xx	93777	**0.00520**	0.00543	0.02078
44	N-t65b11xx	356758	0.00569	**0.00481**	0.02661	44	N-t75u11xx	52708323	**0.00126**	0.00128	0.01482
44	N-t65d11xx	237739	0.00764	**0.00407**	0.02230	56	N-tiw56n54	91554	0.00372	**0.00301**	0.01713
44	N-t65f11xx	217295	**0.00515**	0.00568	0.01042	56	N-tiw56n58	125224	**0.00337**	0.00350	0.01382
44	N-t65i11xx	14469163	**0.00106**	0.00156	0.01108	56	N-tiw56n62	176715	**0.00354**	0.00707	0.02364
44	N-t65l11xx	16719	**0.00153**	0.00287	0.00182	56	N-tiw56n66	226547	**0.00398**	0.01227	0.02872
44	N-t65n11xx	32157	**0.00236**	0.00516	0.02152	56	N-tiw56n67	226033	**0.00299**	0.00813	0.02523
44	N-t65w11xx	138181029	**0.00369**	0.00591	0.01466	56	N-tiw56n72	365146	**0.00332**	0.02243	0.02460
44	N-t69r11xx	771149	0.00454	**0.00433**	0.00681	56	N-tiw56r54	102948	**0.00270**	0.00650	0.01928
44	N-t70b11xx	528419	**0.00220**	0.00222	0.01759	56	N-tiw56r58	129568	**0.00383**	0.00548	0.01849
44	N-t70d11xx	376725	**0.00250**	0.00594	0.02644	56	N-tiw56r66	209491	**0.00468**	0.01239	0.02038
44	N-t70d11xxb	366469	**0.00205**	0.00263	0.02301	56	N-tiw56r67	222810	**0.00267**	0.00633	0.01793
44	N-t70f11xx	360336	**0.00512**	0.00631	0.01286	56	N-tiw56r72	270663	**0.00401**	0.01108	0.02812
44	N-t70i11xx	24785782	**0.00069**	0.00208	0.00743	79	N-usa79	1813986	**0.01182**	0.01303	0.02439

In addition to the MRD values, for each algorithm the computational time needed to perform the $1000n^2$ evaluations was collected. Results are depicted in Fig. 3 in the form of a line plot (note y-axis is in log-scale). As can be observed, GS (in red) is the fastest algorithm among the compared ones, being the second-fastest NES and, finally, EDA. It is worth noting that NES and EDA involve at each iteration very time-consuming processes that increase the computational time in each case, i.e., NES inverts a matrix that it is $O(n^3)$, and EDA uses Minorization-Maximization algorithm for estimating the parameters of the model which is a very intensive iterative procedure. Contrarily, GS employs a simpler process that does not require computationally heavy operations. Overall, GS seems to be the best algorithm, balancing the performance and computational cost.

5.4 Convergence analysis

Trying to understand the performance results observed in the previous section, we performed a convergence analysis of five indicators that can be beneficial regarding such behaviors: entropy, the best objective value, sample size λ, learning rate η and number of restarts (that, we recall, are only performed when numerical errors occur). To that end, 10 repetitions of the algorithms were executed on the $N - be75eec$ instance of the IO and xLOLIB benchmarks (with sizes $n = 50$, $n = 150$ and $n = 250$), and the average value of the indicators across the iterations are depicted in Fig. 4.

As stated in the introduction, one of the contributions in this paper is the parameter-free nature of the proposed algorithm. Specifically, in the previous section, we elaborated on the theoretical reasoning of the automated calculation of λ and η parameters. For that reason, we include in the analysis the parameter-free versions of the GS and NES, denoted as GS* and NES*, respectively.

From the figure, we can note some observations as follows.

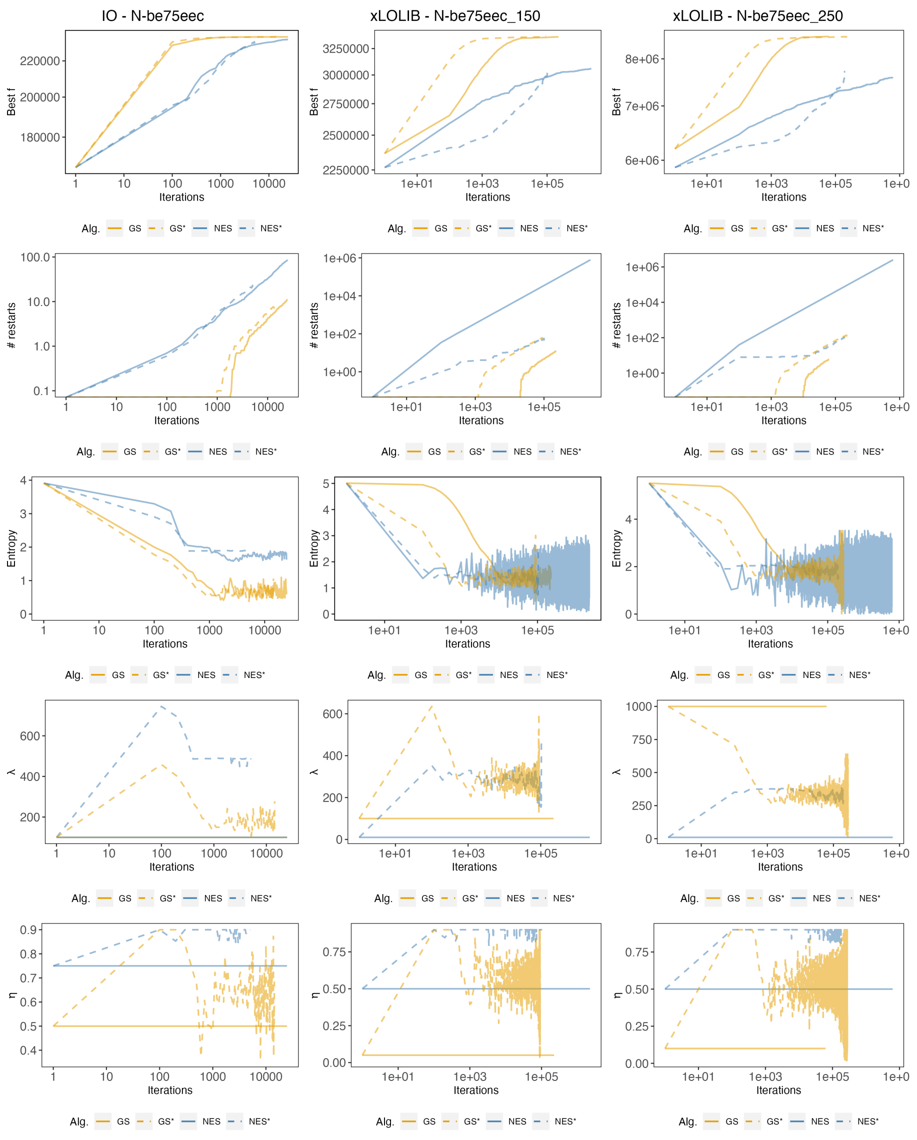

Fig. 4. Convergence plots on instances $N - be75eec$, $N - be75eec\_150$ and $N - be75eec\_250$. Average measures over 10 repetitions of GS, GS*, NES and NES* algorithms of the entropy, the best objective value, sample size λ, learning rate η and number of restarts.

- GS algorithms converge faster than NES algorithms in terms of best f values. Moreover, GS* quickly reaches good objective values than its non-adaptive counterpart GS, especially with instances of large size ($n = 150$ and $n = 250$).
- The behavior of the number of restarts shows that NES algorithms produces great amount of numerical errors, which suggests that NES is not suited to run under the Plackett-Luce model. This is especially significant on $n = 150$ and $n = 250$.

- The entropy plot agrees with respect to the behavior of the number of restarts. When the entropy decreases, the PL distribution gets concentrated on the mode permutation, thus its probability increases. Moreover, when $n = 150$ and $n = 250$, the entropy plot starts to oscillate as an effect of the restarts performed.
- In the case of GS* and NES$*$, the "oscillation after restarts" phenomenon is observable also for behavior plots of the learning rate η and the sample size λ. It is interesting to analyze the left part of these plots (before oscillations starts to be observed). In particular, these parts of the plots seem to coincide with the time required by the best objective value to converge. Interestingly, they clearly show that constant values for η and λ (as is the case for GS and NES) are far to be the desirable settings in the different stages of the optimization.

5.5 Evaluation of the parameter-free scheme

In order to evaluate the effectiveness of the parameter-free scheme, and in view of the results of NES in previous sections, we have exclusively focused on the GS approach, which resulted in the most successful option. The experimentation in Section 5.3 has been reproduced, including the GS* algorithm for the IO and xLOLIB benchmarks.

In order to statistically assess the results obtained, we have followed the Bayesian approach presented in [Benavoli et al. 2017], as it provides a deeper insight into the results than the classical null hypothesis significance tests. In particular, as we cannot assume that the experimental results come from a Gaussian distribution, we have used the Bayesian equivalent of the Wilcoxon's test[11]. The Bayesian analysis was conducted on the average value obtained by each algorithm, GS and GS*, in the 20 repetitions. The procedure used requires the definition of what is understood as "practical equivalence" or "rope" in [Benavoli et al. 2017]. In our case, we have considered that both approaches are equivalent when the difference in MRD is smaller than 10^{-4}. Results are illustrated in Fig. 5 as Bayesian Signed Rank test plots (exact numeric results are presented in Appendix A, Table 3). Briefly, the points in the plot represent a sampling of the posterior distribution of the

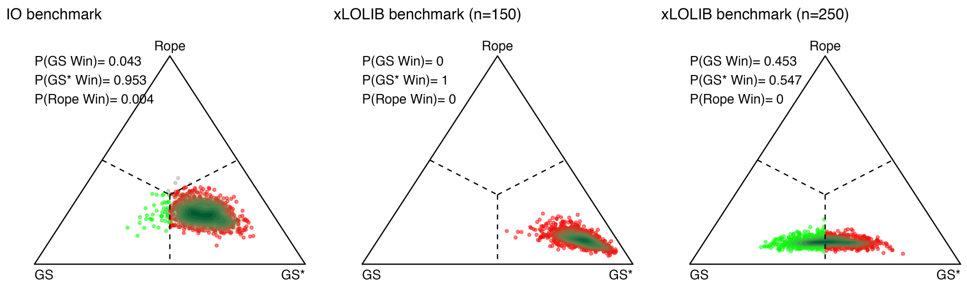

Fig. 5. Bayesian signed rank test plots comparing GS with the respect to the parameter-free version, GS*, on IO and xLOLIB benchmarks. Together with each of the plots, the posterior probability of each algorithm being the best is depicted.

probability of win-lose-tie modeled by the Bayesian model. In other words, the closer a point is to the GS vertex of the triangle (or, equivalently, to the GS* or the Rope vertices), the more probable it is for GS to produce better results (or equivalently, GS* or both algorithms being equal). Therefore, the three areas delimited by the dashed lines show the dominance regions, i.e., the area where the highest probability corresponds to its vertex.

[11]We have used the implementation available in the development version of the scmamp R package [Calvo and Santafe 2016] available at https://github.com/b0rxa/scmamp.

The plots show that the parameter-free version of the algorithm GS* outperforms the standard version, GS, for the three benchmarks. The expected probabilities of winning the GS* in each of the benchmark are 0.953, 1 and 0.547, respectively.

6 CONCLUSION & FUTURE WORK

Using probability distributions has been a common practice to implement stochastic optimization algorithms, however, a great number of works in the literature have shown that using them frequently implies significant computational overheads. In this paper, we propose using the model-based gradient search framework to iteratively adapt the parameters of a probability distribution to optimize permutation-coded problems. We investigated its performance when using plain and natural gradients. Not limited to that, a parameter-free version of the algorithm has been presented. Conducted experiments demonstrated that the gradient search algorithm, under the plain gradient, performs better than natural gradient and also an analogous EDA (using the same probability distribution) in terms of execution results. Numerical experiments also point out that the advantage of GS is not limited to better results, but it is much faster.

Results definitively validate the present research line and encourage future results. In that sense, a possible research line to explore has to do with the probability model implemented in the framework. In this case, we proposed using the multi-stage model called Plackett-Luce model. However, it has been extensively acknowledged, especially in the research on EDAs, that there is no model that is the best for any optimization problem. In fact, previous works have shown that choosing the right model for each problem is the best way to approach it. In that sense, in the context of permutation-coded problems, we identify the Babington-Smith model, and its special cases, Bradley-Terry [Bradley and Terry 1952], Mallows-ϕ and Mallows-θ [Mallows 1957] models, as alternatives for future research [Marden 1996].

From the possible models, distance-based exponential models such as Mallows and Generalized Mallows cannot be directly applied, as it is one of the assumptions of the current work that the parameters that are adjusted via gradient search need to be continuous. In fact, Mallows and its generalization have a parameter that is the mode permutation σ_0 of the distribution. However, a possible research line could develop mixed learning strategies where σ_0 is estimated as usual, but the spread parameters are estimated via gradient search. This strategy would actually replace the numerical methods that are usually used for the estimation of the spread parameters. Another interesting alternative is to relax the mode permutation to a doubly stochastic matrix and modify accordingly both the distance and probability mass function. Such a line of research could allow the presented framework to be extended to distance-based models on the domain of permutations.

REFERENCES

J. Alza, J. Ceberio, and B. Calvo. 2018. Balancing the Diversification-Intensification Trade-off Using Mixtures of Probability Models. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. 1–8.

S.I. Amari. 1998. Natural gradient works efficiently in learning. *Neural computation* 10, 2 (1998), 251–276.

T. Bäck, C. Foussette, and P. Krause. 2013. *Contemporary evolution strategies*. Vol. 86. Springer.

A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon. 2017. Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis. *Journal of Machine Learning Research* 18, 77 (2017), 1–36. http://jmlr.org/papers/v18/16-305.html

A. Berny. 2000. Selection and Reinforcement Learning for Combinatorial Optimization. In *Parallel Problem Solving from Nature PPSN VI*. Springer Berlin Heidelberg, 601–610.

A. Berny. 2001. *Statistical Machine Learning and Combinatorial Optimization*. Springer Berlin Heidelberg, 287–306.

A. Berny. 2002. Boltzmann Machine for Population-Based Incremental Learning. In *Proceedings of the 15th European Conference on Artificial Intelligence*. IOS Press, 198–202.

H.G. Beyer. 1995. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation* 3, 3 (1995), 311–347.

H.G. Beyer and H.P Schwefel. 2002. Evolution strategies–a comprehensive introduction. *Natural computing* 1, 1 (2002), 3–52.

R. A. Bradley and M. E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika* 39, 3/4 (1952).

E. Brochu, V.M. Cora, and N. de Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. https://doi.org/10.48550/ARXIV.1012.2599

B. Calvo and G. Santafe. 2016. scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems. *The R Journal* 8, 1 (2016), 248–256. https://doi.org/10.32614/RJ-2016-017

J. Ceberio. 2014. *Solving Permutation Problems with Estimation of Distribution Algorithms and Extensions Thereof.*

J. Ceberio, E. Irurozki, A. Mendiburu, and J.A. Lozano. 2014. A Distance-based Ranking Model Estimation of Distribution Algorithm for the Flowshop Scheduling Problem. *IEEE Transactions on Evolutionary Computation* 18, 2 (April 2014), 286 – 300.

J. Ceberio, E. Irurozki, A. Mendiburu, and J.A. Lozano. 2015. A Review of Distances for the Mallows and Generalized Mallows Estimation of Distribution Algorithms. *Computational Optimization and Applications* 62, 2 (March 2015), 545–564.

J. Ceberio, A. Mendiburu, and J.A. Lozano. 2013. The Plackett-Luce ranking model on permutation-based optimization problems. In *2013 IEEE Congress on Evolutionary Computation (CEC), Cancun, Mexico*. IEEE, 494–501.

A. Chotard, A. Auger, and N. Hansen. 2012. Cumulative step-size adaptation on linear functions. In *International Conference on Parallel Problem Solving from Nature*. Springer, 72–81.

D.E. Critchlow, M.A. Fligner, and J.S. Verducci. 1991. Probability models on rankings. *Journal of Mathematical Psychology* 35 (1991), 294–318.

P. Diaconis. 1988. Group representations in probability and statistics. *Lecture notes-monograph series* 11 (1988), i–192.

M. Dorigo, M. Birattari, and T. Stutzle. 2006. Ant colony optimization. *IEEE computational intelligence magazine* 1, 4 (2006), 28–39.

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

M.A. Fligner and J.S. Verducci. 1988. Multistage ranking models. *Journal of american statistical association* 83, 403 (1988).

M.R. Garey and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA.

N. Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

N. Hansen and A. Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*. IEEE, 312–317.

A. Hauswirth, S. Bolognani, G. Hug, and F. Dörfler. 2016. Projected gradient descent on Riemannian manifolds with applications to online power system optimization. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 225–232.

E. Irurozki, B. Calvo, and J.A. Lozano. 2019. Mallows and generalized Mallows model for matchings. *Bernoulli* 25, 2 (2019), 1160 – 1188.

H. Joe and J. S. Verducci. 1993. On the Babington Smith Class of Models for Rankings. In *Probability Models and Statistical Analyses for Ranking Data*, M. A. Fligner and J. S. Verducci (Eds.). Springer New York, New York, NY, 37–52.

J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, Vol. 4. IEEE, 1942–1948.

D. P. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

B. Kleinberg, Y. Li, and Y. Yuan. 2018. An alternative view: When does SGD escape local minima?. In *International Conference on Machine Learning*. PMLR, 2698–2707.

W. Kool, H. Van Hoof, and M. Welling. 2019. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*. PMLR, 3499–3508.

P. Larrañaga and J.A. Lozano. 2001. *Estimation of distribution algorithms: A new tool for evolutionary computation.* Vol. 2. Springer Science & Business Media.

Z. Li, X. Lin, Q. Zhang, and H. Liu. 2020. Evolution strategies for continuous optimization: A survey of the state-of-the-art. *Swarm and Evolutionary Computation* 56 (2020), 100694.

R. D. Luce. 1959. *Individual Choice Behavior.* Wiley, New York.

S. Ma and J. Huang. 2007. Clustering threshold gradient descent regularization: with applications to microarray studies. *Bioinformatics* 23, 4 (2007), 466–472.

L. Malagò, M. Matteucci, and G. Pistone. 2011. Towards the Geometry of Estimation of Distribution Algorithms Based on the Exponential Family. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms*. Association for Computing Machinery, 230–242.

C.L. Mallows. 1957. Non-null ranking models. *Biometrika* 44, 1-2 (1957), 114–130.

J.I. Marden. 1996. *Analyzing and modeling rank data.* CRC Press.

R. Martí and G. Reinelt. 2011. *The linear ordering problem: exact and heuristic methods in combinatorial optimization*. Vol. 175. Springer.

R. Martí, G. Reinelt, and A. Duarte. 2012. A benchmark library and a comparison of heuristic methods for the linear ordering problem. *Computational optimization and applications* 51, 3 (2012), 1297–1317.

M. Meila, K. Phadnis, A. Patterson, and J.A. Bilmes. 2012. Consensus ranking under the exponential model. https://doi.org/10.48550/ARXIV.1206.5265

F. Mosteller. 1951. Remarks on the method of paired comparisons. I. The least squares solutions assuming equal standard deviations and equal correlations. *Pyschometrika* 16 (1951), 3–9.

Y. Ollivier, L. Arnold, A. Auger, and N. Hansen. 2017. Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles. *Journal of Machine Learning Research* 18, 1 (2017), 564–628.

A. Ostermeier, A. Gawelczyk, and N. Hansen. 1994a. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation* 2, 4 (1994), 369–380.

A. Ostermeier, A. Gawelczyk, and N. Hansen. 1994b. Step-size adaptation based on non-local use of selection information. In *International Conference on Parallel Problem Solving from Nature*. Springer, 189–198.

R. L. Plackett. 1975. The Analysis of Permutations. *Journal of the Royal Statistical Society* 24, 10 (1975), 193–202.

J. Rojas-Delgado, J. Ceberio, B. Calvo, and J.A. Lozano. 2022. Bayesian Performance Analysis for Algorithm Ranking Comparison. *IEEE Transactions on Evolutionary Computation* (2022), 1–1. https://doi.org/10.1109/TEVC.2022.3208110

S. Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

V. Santucci, M. Baioletti, and A. Milani. 2019. Tackling permutation-based optimization problems with an algebraic particle swarm optimization algorithm. *Fundamenta Informaticae* 167, 1-2 (2019), 133–158.

V. Santucci and J. Ceberio. 2020. Using pairwise precedences for solving the linear ordering problem. *Applied Soft Computing* 87 (2020), 105998.

V. Santucci, J. Ceberio, and M. Baioletti. 2020. Gradient search in the space of permutations: an application for the linear ordering problem. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 1704–1711.

T. Schiavinotto and T. Stützle. 2003. Search Space Analysis of the Linear Ordering Problem. In *Applications of Evolutionary Computing*. Lecture Notes in Computer Science, Vol. 2611. Springer Berlin Heidelberg, 322–333.

L. L. Thurstone. 1927. A law of comparative judgment. *Psychological Review* 34, 4 (1927), 273–286.

X. Wang. 2008. Method of steepest descent and its applications. *IEEE Microwave and Wireless Components Letters* 12 (2008), 24–26.

D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. 2014. Natural Evolution Strategies. *Journal of Machine Learning Research* 15 (2014), 949–980.

Xin Yao and Yong Liu. 1997. Fast evolution strategies. In *International Conference on Evolutionary Programming*. Springer, 149–161.

A.A. Zhigljavsky. 1991. *Theory of Global Random Search*. Springer Netherlands. https://books.google.es/books?id=gsntCAAAQBAJ

M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. 2004. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research* 131, 1-4 (2004), 373–395.

A ADDITIONAL PERFORMANCE RESULTS

Table 2. Results of the GS ($\lambda = 100, \eta = 0.05$ and $\lambda = 1000, \eta = 0.1$), EDA and NES ($\lambda = 10, \eta = 0.5$) for the LOP instances in the xLOLIB benchmark ($n = 150$ and $n = 250$). The Median Relative Deviations (MRD) measures of the values found across the 20 repetitions of the best known results are reported. Results in bold highlight the algorithm that obtained the lowest MRD. A maximum number of $1000n^2$ evaluations were performed by each of the algorithms.

Instance	$n = 150$				$n = 250$			
	Best Known	GS	EDA	NES	Best Known	GS	EDA	NES
N-be75eec	3479547	**0.03297**	0.04101	0.11908	8863792	**0.03884**	0.04045	0.14260
N-be75np	7162017	**0.03554**	0.04119	0.12771	17787439	0.04158	**0.04130**	0.14643
N-be75oi	2243536	**0.02343**	0.02978	0.07808	5902311	**0.03662**	0.04415	0.11717
N-be75tot	12270431	**0.04631**	0.04928	0.12934	30877286	**0.03658**	0.03934	0.14646
N-stabu1	2870590	**0.03536**	0.03592	0.11272	7729985	0.03711	**0.03677**	0.14144
N-stabu2	4326193	**0.03277**	0.03355	0.11358	11468284	0.03798	**0.03737**	0.14305
N-stabu3	4507290	**0.03200**	0.03276	0.10602	11869084	0.04113	**0.04104**	0.13982
N-t59b11xx	3238100	**0.03653**	0.04434	0.13836	8384455	**0.04300**	0.04528	0.15789
N-t59d11xx	1461623	**0.03529**	0.03624	0.12420	3831921	**0.04059**	0.04098	0.13491
N-t59f11xx	1539948	**0.03762**	0.04028	0.12493	3978521	0.04347	**0.04027**	0.15033
N-t59n11xx	318453	**0.03802**	0.04262	0.11830	822422	**0.04536**	0.04645	0.15330
N-t65b11xx	6433186	**0.03389**	0.03911	0.12060	17238947	**0.04365**	0.04423	0.14754
N-t65d11xx	3556730	**0.04029**	0.04104	0.12665	9320283	**0.03986**	0.04129	0.15145
N-t65f11xx	3152817	0.03927	**0.03731**	0.12707	8391707	**0.04075**	0.04142	0.15310
N-t65l11xx	253206	**0.02610**	0.03273	0.10061	665843	**0.03594**	0.04183	0.13853
N-t65n11xx	550079	**0.03532**	0.03770	0.13775	1426219	**0.04022**	0.04338	0.15693
N-t69r11xx	11843437	**0.04289**	0.04995	0.13272	31675692	**0.04018**	0.04141	0.15451
N-t70b11xx	9628132	**0.03667**	0.04196	0.13085	25356014	0.04629	**0.04544**	0.15998
N-t70d11xn	5816430	**0.03824**	0.04356	0.13446	15166124	0.04153	**0.04004**	0.14184
N-t70d11xx	6151361	**0.04235**	0.04435	0.13244	15988331	**0.03910**	0.04167	0.14434
N-t70f11xx	5138418	**0.04156**	0.04291	0.13178	13541532	0.04368	**0.04322**	0.14979
N-t70l11xx	436862	**0.04344**	0.04964	0.12764	1108964	**0.04397**	0.04711	0.16259
N-t70n11xx	948326	0.04440	**0.04391**	0.13763	2438817	**0.04296**	0.04374	0.16176
N-t74d11xx	9365784	**0.03727**	0.03897	0.11845	24359820	0.04197	**0.03850**	0.14519
N-t75d11xx	9621376	**0.03808**	0.04088	0.12915	24987677	0.03957	**0.03916**	0.14274
N-t75e11xx	41567726	**0.03573**	0.03794	0.12977	106474931	**0.04122**	0.04438	0.15484
N-t75k11xx	1539206	**0.03282**	0.03605	0.12682	4080733	0.04216	**0.04112**	0.15228
N-t75n11xx	1740380	**0.03989**	0.04587	0.13269	4515161	0.04513	**0.04406**	0.15650
N-tiw56n54	835995	**0.03424**	0.03962	0.11599	2092186	**0.03889**	0.03996	0.14202
N-tiw56n58	1152698	**0.03773**	0.04313	0.12165	2896743	0.04154	**0.04035**	0.14069
N-tiw56n62	1625001	**0.03818**	0.04313	0.13155	4129750	**0.03772**	0.03832	0.14354
N-tiw56n66	2103450	**0.03731**	0.04499	0.13102	5351638	**0.03663**	0.03716	0.14720
N-tiw56n67	2368279	**0.03767**	0.04161	0.11486	6304037	**0.03502**	0.03972	0.12983
N-tiw56n72	4132640	**0.03838**	0.04350	0.12039	11138394	**0.03688**	0.04022	0.12923
N-tiw56r54	956715	**0.03409**	0.04015	0.12106	2382752	**0.04151**	0.04267	0.14912
N-tiw56r58	1217611	**0.03816**	0.04367	0.12777	3052235	0.04271	**0.04166**	0.14351
N-tiw56r66	1935339	**0.03831**	0.04490	0.13295	4931979	**0.03681**	0.03784	0.14276
N-tiw56r67	2054112	**0.03472**	0.03770	0.12334	5278228	**0.03921**	0.04304	0.14116
N-tiw56r72	2821100	**0.04127**	0.04457	0.12878	7439243	**0.04082**	0.04136	0.14290

Table 3. Results of the GS* and NES* on the IO and xLOLIB benchmarks (n = 150 and n = 250). The Median Relative Deviations (MRD) measures of the values found across the 20 repetitions of the best known results are reported. Results in bold highlight the algorithm that obtained the lowest MRD. A maximum number of $1000n^2$ evaluations were performed by each of the algorithms.

	IO				xLOLIB n = 150			xLOLIB n = 250		
Instance	Best Known	GS*	NES*	Instance	Best Known	GS*	NES*	Best Known	GS*	NES*
N-be75eec	236464	0.00960	0.02254	N-be75eec	3479547	0.03211	0.16165	8863792	0.03920	0.15654
N-be75np	716994	0.00172	0.01611	N-be75np	7162017	0.03583	0.14783	17787439	0.04144	0.14827
N-be75oi	111171	0.00258	0.01561	N-be75oi	2243536	0.02200	0.11577	5902311	0.03459	0.13337
N-be75tot	980516	0.00172	0.03033	N-be75tot	12270431	0.04624	0.15649	30877286	0.03639	0.15638
N-stabu70	362512	0.00799	0.04491	N-stabu1	2870590	0.03397	0.14730	7729985	0.03709	0.14044
N-stabu74	541393	0.00647	0.04180	N-stabu2	4326193	0.03269	0.14234	11468284	0.03765	0.14622
N-stabu75	553303	0.00661	0.04640	N-stabu3	4507290	0.03123	0.14431	11869084	0.03887	0.15151
N-t59b11xx	209320	0.00473	0.03104	N-t59b11xx	3238100	0.03781	0.15944	8384455	0.04393	0.14929
N-t59d11xx	147354	0.00478	0.02693	N-t59d11xx	1461623	0.03527	0.15214	3831921	0.04079	0.13150
N-t59f11xx	122520	0.00040	0.02166	N-t59f11xx	1539948	0.03818	0.15441	3978521	0.04366	0.14849
N-t59i11xx	8261545	0.00021	0.01179	N-t59n11xx	318453	0.03824	0.16913	822422	0.04498	0.15188
N-t59n11xx	20928	0.00272	0.02203	N-t65b11xx	6433186	0.03325	0.14328	17238947	0.04483	0.14575
N-t65b11xx	356758	0.00558	0.04572	N-t65d11xx	3556730	0.03968	0.15510	9320283	0.04025	0.13463
N-t65d11xx	237739	0.00298	0.03415	N-t65f11xx	3152817	0.03875	0.14885	8391707	0.04115	0.14738
N-t65f11xx	217295	0.00501	0.03416	N-t65l11xx	253206	0.02485	0.14124	665843	0.03539	0.16224
N-t65i11xx	14469163	0.00110	0.02811	N-t65n11xx	550079	0.03354	0.16728	1426219	0.04205	0.15699
N-t65l11xx	16719	0.00144	0.00446	N-t69r11xx	11843437	0.04209	0.16165	31675692	0.04023	0.15958
N-t65n11xx	32157	0.00239	0.02942	N-t70b11xx	9628132	0.03623	0.15653	25356014	0.04457	0.15881
N-t65w11xx	138181029	0.00361	0.03591	N-t70d11xn	5816430	0.03879	0.14099	15166124	0.04142	0.14473
N-t69r11xx	771149	0.00444	0.02477	N-t70d11xx	6151361	0.04069	0.15274	15988331	0.04073	0.13965
N-t70b11xx	528419	0.00240	0.03927	N-t70f11xx	5138418	0.04072	0.14993	13541532	0.04473	0.14632
N-t70d11xx	376725	0.00280	0.04485	N-t70l11xx	436862	0.03884	0.17811	1108964	0.04439	0.16421
N-t70d11xxb	366469	0.00215	0.02810	N-t70n11xx	948326	0.04364	0.15908	2438817	0.04372	0.15651
N-t70f11xx	360336	0.00535	0.03612	N-t74d11xx	9365784	0.03674	0.14954	24359820	0.04128	0.14178
N-t70i11xx	24785782	0.00129	0.02055	N-t75d11xx	9621376	0.03782	0.14662	24987677	0.03984	0.13812
N-t70k11xx	60659200	0.00112	0.03134	N-t75e11xx	41567726	0.03396	0.16251	106474931	0.04256	0.15947
N-t70l11xx	25253	0.00000	0.00879	N-t75k11xx	1539206	0.03208	0.16138	4080733	0.04189	0.14251
N-t70n11xx	52704	0.00370	0.02849	N-t75n11xx	1740380	0.04012	0.16915	4515161	0.04342	0.14698
N-t70u11xx	21716400	0.00190	0.02306	N-tiw56n54	835995	0.03435	0.14514	2092186	0.03981	0.14304
N-t70w11xx	224319954	0.00295	0.04287	N-tiw56n58	1152698	0.03742	0.15056	2896743	0.04103	0.14407
N-t70x11xx	283808865	0.00237	0.03660	N-tiw56n62	1625001	0.03649	0.14909	4129750	0.03819	0.14226
N-t74d11xx	566089	0.00095	0.02897	N-tiw56n66	2103450	0.03663	0.14873	5351638	0.03702	0.13913
N-t75d11xx	578304	0.00199	0.03957	N-tiw56n67	2368279	0.03615	0.14365	6304037	0.03565	0.13262
N-t75e11xx	2739219	0.00298	0.03140	N-tiw56n72	4132640	0.03852	0.15013	11138394	0.03593	0.13715
N-t75i11xx	63567735	0.00074	0.03571	N-tiw56r54	956715	0.03305	0.14896	2382752	0.04028	0.14412
N-t75k11xx	108844	0.00106	0.02880	N-tiw56r58	1217611	0.03813	0.13904	3052235	0.04104	0.14198
N-t75n11xx	93777	0.00522	0.03148	N-tiw56r66	1935339	0.03770	154.00000	4931979	0.03764	0.14188
N-t75u11xx	52708323	0.00126	0.04119	N-tiw56r67	2054112	0.03342	0.14528	5278228	0.03711	0.14274
N-tiw56n54	91554	0.00386	0.04312	N-tiw56r72	2821100	0.04002	0.14301	7439243	0.03845	0.14460
N-tiw56n58	125224	0.00375	0.04258							
N-tiw56n62	176715	0.00344	0.05117							
N-tiw56n66	226547	0.00420	0.05163							
N-tiw56n67	226033	0.00312	0.05374							
N-tiw56n72	365146	0.00372	0.05039							
N-tiw56r54	102948	0.00283	0.05074							
N-tiw56r58	129568	0.00446	0.04963							
N-tiw56r66	209491	0.00505	0.04635							
N-tiw56r67	222810	0.00281	0.04304							
N-tiw56r72	270663	0.00385	0.04992							
N-usa79	1813986	0.01354	0.05622							