Limited Budget Optimization of the Asteroid Routing Problem

Valentino Santucci[1*]

[1*]WARREDOC Center, University for Foreigners of Perugia, Piazza Fortebraccio 4, Perugia, 06123, Italy.

Corresponding author(s). E-mail(s): valentino.santucci@unistrapg.it;

Abstract

Limited budget optimization scenarios typically arise in two situations: when evaluating the objective function is computationally expensive, or in mission-critical contexts where a solution must be obtained within a short timeframe. In this work, we focus on the limited budget optimization of permutation problems, a class of combinatorial optimization problems where the objective function is defined over the space of permutations. As a use case, we examine the Asteroid Routing Problem (ARP), which aims to determine the optimal sequence of asteroids to be visited by a spacecraft departing from Earth's orbit, while minimizing both energy consumption and visit time. The ARP is a recently proposed computationally expensive benchmark permutation problem that may also be relevant in time-critical mission planning scenarios. In particular, we propose the application of two iterative optimization algorithms: FAT-RLS and FAT-EA. These algorithms differ in their search strategies, with FAT-RLS using randomized local search and FAT-EA following a trajectory based evolutionary approach. However, both incorporate a tabu mechanism and a perturbation strength regulation strategy to enhance search effectiveness while ensuring low computational overhead. To evaluate their performance, we conducted extensive experiments on well-established ARP instances, comparing FAT-RLS and FAT-EA with two recognized metaheuristics for limited budget optimization of permutation problems. The results clearly show that our approaches outperform the competitors.

Keywords: Limited Budget Optimization, Permutation Problems, Asteroid Routing Problem, Randomized Local Search, Evolutionary Algorithm

1 Introduction and Related Work

Limited budget optimization is crucial in scenarios where computational resources –such as time, memory, or even money– are restricted. These situations typically arise in two contexts: when evaluating the objective function to optimize is computationally expensive, or when a solution is required within a strict timeframe, as in mission-critical applications. Furthermore, these problems are often treated as black-box optimization tasks because their objective functions generally lack a closed-form mathematical expression. This occurs, for example, when evaluating a solution necessitates running a complex numerical engineering simulation, conducting a real physical experiment, or relying on social feedback as in tasks related to stock market prediction. In such cases, traditional optimization methods that do not treat the problem as a black-box are excluded and, even metaheuristic approaches that demand significant computational resources or long runtimes become not feasible. As a result, developing black-box optimization algorithms that can effectively explore the solution space within a limited budget of objective function evaluations is of paramount importance.

The most investigated scenario is that of continuous problems, where Bayesian Optimization (BO) techniques [1] are widely applied. These methods iteratively refine a surrogate model –typically a Gaussian process or a random forest– of the underlying objective function. At each iteration, using an acquisition function or a series of relatively inexpensive surrogate evaluations, a BO algorithm identifies one or more promising candidate solutions, which are then evaluated with the true objective function. The true evaluations are then used to update the surrogate model for the next iteration. This process significantly reduces the need for costly function evaluations, allowing the application of classical black-box strategies guided primarily by the surrogate function. Therefore, BO facilitates efficient exploration of the solution space by balancing exploration and exploitation, making it particularly effective for problems where evaluations are time-consuming and/or resource-intensive.

A prominent application of BO is hyperparameter tuning for machine learning algorithms [2, 3]. However, the effectiveness of BO methods depends on the accuracy of the surrogate model they learn. As the dimensionality of the search space increases, the number of objective function evaluations required to learn an effective surrogate model typically grows exponentially. In practice, standard BO techniques based on Gaussian processes often become impractical for problems with more than 15/20 decision variables [4, 5]. For example, [6] and [7] have shown that basic evolution strategies outperformed BO by a significant margin in estimating the 60 to 120 continuous parameters of a hydraulic model within a limited budget of evaluations. However, recent approaches have been proposed to alleviate the impact of the curse of dimensionality on continuous BO [8, 9].

In this work, we focus on permutation problems, a specific class of combinatorial optimization problems where the objective function is defined over the search space of permutations. Permutation problems are quite common, as they can model tasks of practical interest, such as finding the optimal ordering of jobs to be processed on one or more machines [10], determining the best possible matching of workers to tasks [11], or identifying the shortest route through a set of locations [12].

In this work, we consider the Asteroid Routing Problem (ARP) as a paradigmatic benchmark for expensive permutation problems. Originally introduced in [13], the ARP was inspired by the *11th Global Trajectory Optimisation Competition* organized by the European Space Agency[1]. Given a set of asteroids and their orbital information, the ARP aims to determine an optimal trajectory for a spacecraft that, after departing from Earth, sequentially visits each asteroid while minimizing both energy consumption and total travel time. The motivation for this problem stems from the rapid depletion of Earth's mineral resources, which are essential for electronic devices (e.g., quartz, lithium, and others), and the potential for future asteroid mining as an alternative source. However, our interest is not on the astrophysical details but rather on the computational aspects of the problem. From this point of view, the ARP is a permutation problem with an objective function that requires to run a costly numerical procedure.

Since the ARP is a combinatorial problem, classical BO approaches based on Gaussian processes are not applicable. In [13], two algorithms were used: CEGO, a distance-based Bayesian method tailored for combinatorial problems, and UMM, an estimation of distribution algorithm for permutation problems that was specifically redesigned for limited budget scenarios. More recently, Santucci addressed the ARP in [14] using FAT-RLS, a simple randomized local search scheme which incorporates both a tabu mechanism and a perturbation strength regulation strategy, making the search more effective without introducing significant computational overhead. This work [14] also serves as the seed for the present study. In particular, we introduce a novel algorithm, FAT-EA, which differs from FAT-RLS by using the trajectory-based evolutionary search typical of $(1 + 1)$-EAs (see e.g. [15, 16]) instead of the randomized local search scheme. The rationale behind this modification is to sacrifice a bit of exploitation in order to provide the search with the ability, at least theoretically, to escape some stagnation situations caused by FAT-RLS's inability to move beyond certain local optima of the ARP instance at hand. An extensive set of experiments have been carried out on commonly adopted ARP benchmark instances to compare FAT-EA with FAT-RLS, as well as with the two seminal ARP algorithms, CEGO and UMM.

The rest of the paper is structured as follows. Section 2 describes permutation problems and other preliminary concepts. The ARP is presented in Section 3. Then, the two algorithms FAT-EA and FAT-RLS are introduced in Section 4, while Section 5 shortly recalls the competitor algorithms used in the experiments. Section 6 shows the experimental results obtained, while Section 7 concludes the article.

2 Permutation Problems

Permutations are versatile algebraic objects that find applications in many different fields because of their ability to model a variety of concepts, including orderings and rankings of items, one-to-one mappings between two sets, as well as tours and sets of cycles within a collection of locations.

[1]For more details, we refer the interested reader to the competition webpage at https://sophia.estec.esa.int/gtoc_portal/?page_id=782.

Permutations, which are usually denoted by Greek symbols such as π or σ, can be mathematically defined as bijective functions onto the set $[n] = \{1, 2, \ldots, n\}$. Several notations are available, though the simplest and most common one is the so-called *single line notation*, where a permutation π is represented as a list of all different items, i.e.,

$$\pi = \langle \pi(1), \pi(2), \ldots, \pi(n) \rangle, \tag{1}$$

where $\pi(i) \in [n]$ indicates the item in position $i \in [n]$ in the list, ensuring that $\pi(i) \neq \pi(j)$ for any pair $i \neq j$.

The set of all the permutations of degree n is denoted by $\mathbb{S}_n$, which has cardinality $n!$ and is also known as the *symmetric group*. In fact, the standard function composition operation allows to compose two permutations into a third permutation. Given two permutations π and σ, their composition is denoted by $\pi\sigma$ and its elements are $\pi\sigma(i) = \pi(\sigma(i))$, for all $i \in [n]$. The composition operation is associative, admits a unique identity permutation $\iota = \langle 1, 2, \ldots, n \rangle$, and each permutation π has a unique inverse permutation π^{-1} such that $\pi\pi^{-1} = \pi^{-1}\pi = \iota$. These three properties prove that $\mathbb{S}_n$ is a group, a characteristic that has been exploited to design both swarm and evolutionary algorithms based on strong mathematical foundations [17].

In permutation optimization problems, the goal is to minimize or maximize a given objective function of the form $f : \mathbb{S}_n \to \mathbb{R}$, i.e., a real-valued function whose domain is the set of permutations $\mathbb{S}_n$. Permutation problems are combinatorial in nature, so f is not differentiable and classical gradient-based algorithms cannot be adopted[2]. Moreover, if f has no analytical definition – for example, because it requires to run an experiment –, the problem is a black-box problem, so an algorithm can only gather information about f by testing multiple permutation solutions and observing the returned objective values. Often, black-box optimization problems are also characterized by an objective function that is expensive to evaluate in terms of time – for example, because it requires to run a computationally intensive simulation –, but also in terms of other resources such as memory or money. Therefore, a suitable algorithm for expensive black-box permutation optimization problems is required to navigate the search space of permutations and should be able to reach a good enough solution within a low budget of objective function evaluations.

Even in a black-box scenario, it is often possible to intuitively identify which features of a permutation are important for a given problem simply by examining the problem description. Indeed, two distinct families of permutation problems can be identified:

- *ordering problems*, where the objective is to determine the optimal sequence of items within a given set A, and
- *assignment problems*, where the aim is to find the best possible one-to-one correspondence between two given sets A and B of equal size.

While these classifications may not be exhaustive, they cover many permutation problems frequently encountered in the scientific literature. For example, the Linear Ordering Problem (LOP) [19] and the Permutation Flowshop Scheduling Problem (PFSP) [20] are two typical examples of ordering problems, while the polynomially

[2] Though model-based gradient search algorithms are possible [18].

solvable Assignment Problem [21] and its NP-Hard quadratic variant – the Quadratic Assignment Problem (QAP) [22] – are common examples of assignment problems. Moreover, also the well known Traveling Salesman Problem (TSP) [12], though requiring to determine a circular tour among a given set of cities, can be seen as an ordering problem by (arbitrarily) designating a start/end city for all the tours, so that they can be represented as orderings over the remaining cities.

It is important to note that the distinction between ordering and assignment problems is not always clear-cut. In fact, it is known that both TSP and LOP instances can be seen as special cases of QAP instances [23], thus suggesting that the boundary between the ordering or assignment nature of a permutation problem is not yet well understood.

Two of the most commonly used genotypic representations for permutation problems are the classical linear representation and the permutation matrix representation. The linear representation is essentially the transposition in memory of the single line notation described in Equation (1). In contrast, the permutation matrix representation encodes a permutation as a binary matrix with exactly one 1-entry in each row and each column. While the linear representation is suitable for both ordering and assignment problems, the matrix representation does not directly encode any ordering information, but only the pairings between rows and columns indices. For this reason, in this work we adopt the linear genotypic representation of permutation solutions.

In the context of ordering problems, there are two equivalent linear representations: *ordering representation* and *ranking representation*, as termed in [24]. As previously discussed, an ordering problem aims to optimally sort a set A of n elements based on a problem-specific objective function. Using the ordering representations, positions from $[n]$ are mapped to elements of A, whereas the reverse mapping occurs in the ranking representation. Since, the elements of A, like the positions, are identified with numbers in $[n]$, it is easy to confuse these two representations. However, they both encode the same information and can be translated into one another through simple inversion of the permutation. Nevertheless, it is essential to clearly specify which is the used representation when defining the objective function and the algorithms/operators for permutation problems. In fact, using an incorrect representation without the necessary conversion, when required by the objective function or the algorithm/operator at hand, can lead to errors that are difficult to detect but detrimental for the effectiveness. In this work we adopt the ordering representation.

Finally, it is worth noting that in the permutation space, different movement operators are available to build local search schemes or genetic operators such as mutation. The three most commonly used classes of movement operators are: swaps of two adjacent items (also called *adjacent swaps*), swaps of two generic items (also called *exchanges*), and shifts of an item to another position (also called *insertions*). While exchanges are known to be suitable for assignment problems, adjacent swaps and insertions are appropriate for ordering problems [25]. Moreover, it is easy to see that insertion moves include adjacent swaps as special cases, and a single insertion move is equivalent to a chain of multiple adjacent swaps. Therefore, insertion moves are usually adopted in the design of algorithms for ordering problems.

3 The Asteroid Routing Problem

The Asteroid Routing Problem (ARP) was introduced in [13] as a benchmark for expensive and mission-critical black-box permutation optimization. It involves planning a route for a spacecraft that, once launched from Earth, must visit all the asteroids in a given set of n asteroids $A = \{a_1, a_2, \ldots, a_n\}$ in such a way that it minimizes both its fuel consumption and the total time required to complete the journey.

Computationally, the ARP is a bilevel optimization problem consisting of two nested tasks. The outer task involves determining an ordering of the asteroids in A, while the inner task requires calculating the parking and transit times for reaching each asteroid in the order given by the outer task.

A solution for the bilevel ARP is a pair $(\pi, \mathbf{t})$, where: $\pi \in \mathbb{S}_n$ encodes the ordering of A, while the vector $\mathbf{t} \in \mathbb{R}^{2n}_{\geq 0}$ encodes the parking and transit times for each asteroid. More specifically, assuming that a_0 represents the Earth, for each step $i \in [n]$:

- $a_{\pi(i)}$ is the i-th asteroid visited by the spacecraft,
- t_{2i-1} is the parking time spent by the spacecraft in the orbit of asteroid $a_{\pi(i-1)}$, and
- t_{2i} is the transit time required for the spacecraft to travel from the orbit of asteroid $a_{\pi(i-1)}$ to the orbit of asteroid $a_{\pi(i)}$.

To formalize the objective function of the ARP, we also consider the following auxiliary variables:

- τ_i, for $i \in \{0, 1, \ldots, n-1\}$, is the launch epoch from the orbit of asteroid $a_{\pi(i)}$;
- Δv_{2i-1} and Δv_{2i}, for $i \in [n]$, are the impulses required for the maneuvers to insert the spacecraft into the transit orbit between asteroids $a_{\pi(i-1)}$ and $a_{\pi(i)}$, and then into the parking orbit of asteroid $a_{\pi(i)}$, respectively.

The spacecraft is on Earth at start epoch τ_0, which is given by the ARP instance, while the other launch epochs are computed as follows:

$$\tau_i = \tau_0 + \sum_{j=1}^{2i} t_j. \tag{2}$$

For each step $i \in [n]$, the transit and parking impulses Δv_{2i-1} and Δv_{2i} allow the spacecraft to rendezvous with asteroid $a_{\pi(i)}$. These velocity impulses are computed as solutions of the so-called Lambert's problem, i.e.,

$$(\Delta v_{2i-1}, \Delta v_{2i}) = \text{Lambert}(a_{\pi(i-1)}, a_{\pi(i)}, \tau_{i-1}, t_{2i}), \tag{3}$$

for which we refer the interested reader to [13] and [26].

Once the upper-level permutation π is available, Equations (2) and (3) define the inner task, which consists of n continuous optimization problems that, after being sequentially solved, yield the vector of times $\mathbf{t}$ and the vector of the velocity impulses $\Delta\mathbf{v}$.

In [13], the inner continuous problems are solved using the Sequential Least Squares Programming (SLSQP) algorithm [27], a deterministic method which allows to focus

on the outer task of the ARP problem by treating the computations of the values for $\mathbf{t}$ and $\Delta\mathbf{v}$ as an internal deterministic procedure depending only on the provided asteroid ordering π. Therefore, the two levels of the problem are collapsed into one, and the ARP becomes a black-box permutation problem of ordering nature, whose goal is to minimize the objective function defined on the domain of permutations $\mathbb{S}_n$ as

$$f(\pi) = \sum_{i=1}^{2n} |\Delta v_i| + \frac{2 \text{ km/s}}{30 \text{ days}} \cdot \sum_{i=1}^{2n} t_i, \tag{4}$$

where: the first summation is proportional to the energy consumed by the spacecraft to perform all the maneuvers, the second summation represents the total time taken by the spacecraft to complete its journey, while the constant in front of the second summation has been experimentally derived in [13].

Computing Equation (4) is computationally expensive, and since it relies on numerical algorithms, its complexity in terms of Big-O notation is difficult to characterize precisely. However, we measured the computational time required to run an evaluation of a permutation, which we observed it takes approximately 3 seconds for an instance of size $n = 10$ and around 9 seconds for $n = 30$.

Finally, note that an ARP instance is completely defined by: the orbital parameters of the Earth and all asteroids, the starting epoch τ_0, and a gravitational parameter. An instance generator is provided in [13] which takes as input a seed parameter and the number of asteroids n, i.e., the size of the generated instance. The implementations of the ARP objective function and the instance generator are available from https://doi.org/10.5281/zenodo.5725837.

4 Fast Adaptive Tabu-based Algorithms

The two algorithms under consideration, FAT-RLS and FAT-EA, are built upon three key components: (i) a simple and efficient trajectory search scheme –randomized local search for FAT-RLS and $(1 + 1)$ evolutionary search for FAT-EA, (ii) an adaptive perturbation strategy that gradually reduces mutation strength over iterations, and (iii) a tabu-based mechanism that prevents the same item from being moved too frequently within incumbent solution.

In the following subsections we describe the three components and we provide the pseudo-code for both FAT-RLS and FAT-EA.

4.1 Search Schemes: RLS vs. $(1 + 1)$-EA

Since our focus is on the ARP, which, as described in Section 3, can be categorized as a permutation ordering problem, we employ the insertion neighborhood for the search of both FAT-RLS and FAT-EA. Given an ARP instance of size n and a solution $\pi \in \mathbb{S}_n$, an insertion move (i, j), where $i, j \in [n]$, consists of moving the item $\pi(i)$ to position j in π. Denoting by σ the permutation resulting from applying the insertion (i, j) to

π, we have that, in a forward insertion where $i < j$:

$$\sigma(k) = \begin{cases} \pi(k) & \text{if } k < i \text{ or } k > j, \\ \pi(k+1) & \text{if } i \leq k < j, \\ \pi(i) & \text{if } k = j, \end{cases} \tag{5}$$

while, in a backward insertion where $i > j$:

$$\sigma(k) = \begin{cases} \pi(k) & \text{if } k < j \text{ or } k > i, \\ \pi(k-1) & \text{if } j < k \leq i, \\ \pi(i) & \text{if } k = j. \end{cases} \tag{6}$$

FAT-RLS employs a randomized local search (RLS) scheme, which evolves a single solution by randomly selecting a neighboring solution at each iteration. If the new solution is fitter, it replaces the current one in the next iteration. Due to its simplicity, RLS has been widely studied in theoretical evolutionary computation [28], though it is less common in practical applications. However, we selected RLS as the search engine for FAT-RLS because of its strong exploitative nature and its efficiency (requiring only a single objective function evaluation per iteration). This characteristics make it particularly suitable for optimizing within a limited evaluation budget in expensive black-box scenarios. Furthermore, since each iteration only involves applying a simple movement operator to the current solution, the computational overhead remains minimal. This is especially advantageous when compared to the computationally intensive model learning and update procedures required in Bayesian approaches such as [29, 30].

However, by checking just one neighbor per iteration, RLS cannot escape local optima and it is practically trapped in the basin of attraction where it starts. While this may not be a significant issue under a limited evaluation budget, to enhance exploratory capabilities, we also consider the $(1 + 1)$ evolutionary search scheme in FAT-EA. Originally designed for binary search spaces, the $(1 + 1)$ scheme is widely used in theoretical studies of evolutionary algorithms [31]. It mutates a binary string by flipping each bit with a probability of $1/n$ (where n is the length of the string). On average, this flips one bit per iteration, similar to RLS, but unlike RLS, it allows multiple bits to be flipped with a positive probability. Scharnow et al. [16] observed that the number of bits flipped in this mechanism follows a Poisson distribution (with parameter $\lambda = 1$). Based on this insight, they extended the $(1 + 1)$ scheme to permutation-based representations by performing a Poisson-distributed number of perturbations at each iteration. Moreover, as also suggested in [16], we ensure that at least one perturbation is always applied. Specifically, we sample a random number r from a Poisson distribution and then apply $r + 1$ insertions to the current solution.

The key difference between the $(1 + 1)$ scheme and RLS lies in the number of insertions per iteration. In FAT-RLS, exactly one insertion is performed, whereas in FAT-EA, at least one insertion is applied. Theoretically, this adjustment introduces in FAT-EA the (potential) ability to escape local optima, while slightly shifting the search balance from exploitation toward exploration compared to FAT-RLS.

4.2 Adaptive Perturbation Strategy

From Equations (5) and (6) it is easy to see that a generic insertion (i, j) rearranges $|i - j| + 1$ items in the permutation, thus making the insertion move equivalent to a series of $|i - j|$ adjacent swap moves. Limiting our attention to forward insertions (the case of backward insertions is analogous), as depicted in Equation (5), the insertion (i, j) is equivalent to the sequence of $|i - j|$ adjacent swaps

$$(i, i + 1); \quad (i + 1, i + 2); \quad \ldots; \quad (j - 1, j).$$

Since the Kendall's-τ distance between two permutations is the number of adjacent swaps needed to transform one permutation into the other, applying an insertion (i, j) to a permutation π results in a new permutation σ with a Kendall's-τ distance of $|i-j|$ from π. Thus, we define the perturbation strength of an insertion (i, j) as $d = |i - j|$.

Therefore, rather than randomly selecting i and j as in standard approaches, we control the perturbation strength d of the insertion move(s) applied at each iteration. Indeed, both FAT-RLS and FAT-EA employ an adaptive perturbation strength strategy, which requires two hyperparameters: the *initial perturbation strength* d_{ini}, and the *steepness factor* β.

The perturbation strength d is initialized to d_{ini} and each insertion (i, j) is randomly chosen among the insertions such that $|i - j| = d$. Moreover, the perturbation strength d monotonically decreases using a "skewed S-shaped" function, whose steepness is regulated by β. This strategy fosters the transition from explorative to exploitative behaviour over iterations, a well known best practice in designing meta-heuristic algorithms [32].

The "skewed S-shaped" function has both domain and codomain in $[0, 1]$ and, given $\beta \geq 1$, is defined as

$$s_\beta(p) = 1 - \frac{1}{1 + \left(\frac{1-p}{p}\right)^\beta}. \tag{7}$$

Hence, the perturbation strength d at each iteration is computed by rounding the product of the initial strength d_{ini} and the s function applied to the percentage of evolution made (i.e., the ratio of the current iteration number to the allowed iteration budget). Figure 1 depicts the function $s_\beta(p)$ across various β values. Specifically, when $\beta = 1$, $s_1(p) = 1 - p$, leading to a linear decrease of the perturbation strength, while as β increases the curve steepens in the central part, thus extending the initial exploration phase and making more abrupt and quick the transition to the final exploitative phase.

4.3 Tabu Mechanism

A tabu-based mechanism is introduced to serve two main purposes: (i) preventing the search trajectory from revisiting previously encountered solutions, and (ii) ensuring that the same items are not moved too frequently by the applied perturbations.

The tabu data structure used is a queue, TQ, with a maximum size k –a hyperparameter of the algorithm–, that stores the k most recently moved items. Therefore, when an insertion (i, j) has to be selected, the move is forbidden if the item $\sigma(i)$ to be shifted is marked as tabu, i.e., if $\sigma(i) \in TQ$.

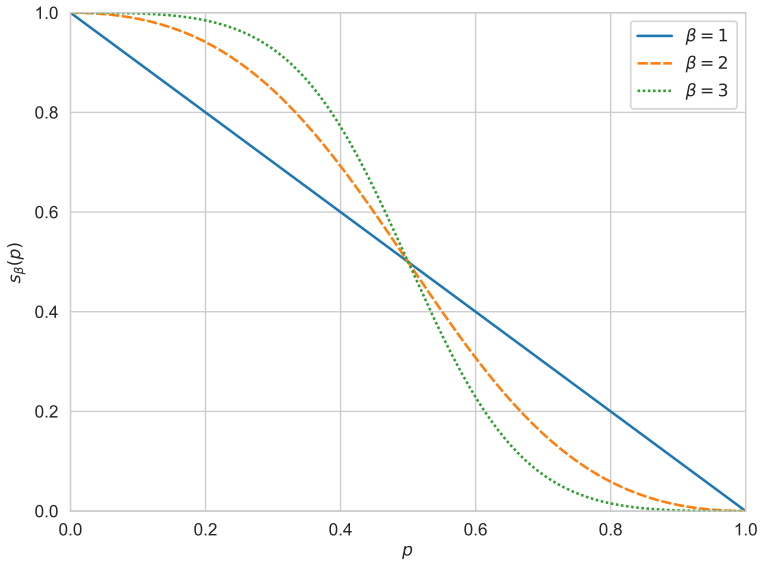

Fig. 1: The "skewed S"-shaped function for $\beta = 1, 2, 3$. Figure taken from [14].

Once a valid insertion (i, j), i.e., one where $\sigma(i) \notin TQ$, is selected and applied, the item $\sigma(i)$ is added to TQ. If the queue reaches its maximum capacity k, the oldest item in the queue is removed. Clearly, since the tabu queue is updated after every insertion, this occurs once per iteration in FAT-RLS and potentially multiple times in FAT-EA, depending on the number of insertions performed.

In summary, both FAT-RLS and FAT-EA adopt simple trajectory search schemes which only performs one objective function evaluation per iteration and, to improve their effectiveness in the limited budget scenario typical of expensive black-box permutation problems, they utilize an adaptive perturbation strength approach along with a tabu-based mechanism that limits the number of perturbations allowed in each iteration, with the goal of speeding up the process towards promising solutions.

4.4 Pseudocode of FAT-RLS and FAT-EA

For the sake of completeness, we provide the pseudocode for FAT-RLS and FAT-EA in Algorithm 1.

Both algorithms aim to minimize (without loss of generality) an objective function of the form $f : \mathbb{S}_n \to \mathbb{R}$, by taking in input the three hyperparameters $d_{ini} \in [1, n)$, $\beta \geq 1$, and $k \leq n$, as previously discussed.

After initializing variables (lines 2–6), each iteration of the main loop (lines 8–25) generates a new trial solution (lines 9–20), which then competes with the current solution and replaces it if fitter (lines 21–25). More specifically, lines 10–11 determine the perturbation strength according to the strategy described in Section 4.2. The key

difference between FAT-RLS and FAT-EA lies in line 14, where the number of insertions to apply to the incumbent solution is selected (see Section 4.1). The insertions are then randomly selected and applied in lines 15–20, following the tabu mechanism outlined in Section 4.3. Finally, if the trial solution is fitter, it replaces the current one (lines 21–25). Once the evaluation budget is exhausted, the best solution found is returned (line 26).

Interestingly, both schemes are elitist, as the current solution at each iteration is also the best solution found so far.

Algorithm 1 Pseudocode of FAT-RLS and FAT-EA

1: **Input:** Objective function: $f : \mathbb{S}_n \to \mathbb{R}$. Hyperparameters: d_{ini}, β, k.
2: */* Initialization */*
3: $\pi \leftarrow$ random permutation from $\mathbb{S}_n$
4: Evaluate $f(\pi)$
5: $nfev \leftarrow 1$
6: $TQ \leftarrow \emptyset$
7: */* Main loop */*
8: **while** $nfev < budget$ **do**
9: */* Determine the perturbation strength */*
10: $p \leftarrow nfev/budget$
11: $d \leftarrow \lceil\ 1 + s_\beta(p) \cdot (d_{\text{ini}} - 1)\ \rceil$
12: */* Perturbation(s) and tabu queue management */*
13: $\sigma \leftarrow \pi$
14: $r \leftarrow 0$ for FAT-RLS **else** sample from a Poisson distribution for FAT-EA
15: **for** $r + 1$ times **do**
16: Sample i, j s.t. $|i - j| = d$ and $\sigma(i) \notin TQ$
17: Apply the insertion (i, j) to σ
18: Push item $\sigma(j)$ into TQ
19: **if** $|TQ| = k$ **then**
20: Remove oldest item from TQ
21: */* Update current solution */*
22: Evaluate $f(\sigma)$
23: **if** $f(\sigma) < f(\pi)$ **then**
24: $\pi \leftarrow \sigma$
25: $nfev \leftarrow nfev + 1$
26: **return** $\pi, f(\pi)$

5 Other Optimization Algorithms

The goal of this work is to investigate the performance of the FAT-RLS and FAT-EA algorithms, described in Section 4, on the ARP, i.e., the permutation ordering problem detailed in Section 3. To this end, we experimentally compare FAT-RLS and FAT-EA against four competitor algorithms: a simple Random Search (RS) scheme used as a

baseline competitor, the constructive heuristic called Greedy Nearest Neighbor (GNN) specifically introduced in [13] for the ARP, the Bayesian algorithm known as Combinatorial Efficient Global Optimization (CEGO), introduced in [29], and the estimation of distribution algorithm termed Unbalanced Mallows Model (UMM), introduced in [33].

In [13], both UMM and CEGO were executed under two settings: the *uninformed* setting, where the standalone versions of the algorithms were run, and the *informed* setting, where their initial solutions were created using GNN instead of being generated randomly.

The four competitor algorithms are briefly described in the following subsections.

5.1 Random Search (RS)

The trivial random search procedure, denoted by RS, is considered as a baseline method in this work. RS generates a given number of permutation solutions uniformly at random using the well known Fisher Yates algorithm [34], then it evaluates the objective function on all the generated solutions and returns the best one. Although trivial, RS is an interesting baseline method because all solutions can be evaluated in parallel (at least in principle).

5.2 Greedy Nearest Neighbor (GNN)

The Greedy Nearest Neighbor heuristic, from now on referred to as GNN[3], is inspired from the well known nearest neighbor heuristic for the TSP [35].

The main idea behind GNN is that a reasonably effective permutation of the asteroids can be formed by repeatedly visiting the asteroid which is closest, in terms of Euclidean distance, to the last visited one, after determining the positions of all unvisited asteroids at the time the spacecraft is arrived in the last visited one.

The pseudocode of GNN is provided in Algorithm 2, while for further details we refer the interested reader to [13].

Algorithm 2 Pseudocode of the GNN heuristic

function GNN
  $s \leftarrow a_0$
  $\tau \leftarrow \tau_0$
  $U \leftarrow [n]$
  for $i \leftarrow 1$ **to** $n-1$ **do**
    $\pi(i) \leftarrow \arg\min_{j \in U} d_{Eucl}(s, a_j, \tau)$
    $(t_{2i-1}, t_{2i}) \leftarrow \text{SLSQP}(s, a_{\pi(i)})$
    $\tau \leftarrow \tau + t_{2i-1} + t_{2i}$
    $U \leftarrow U \setminus \{\pi(i)\}$
    $s \leftarrow a_{\pi(i)}$
  return $\pi, f(\pi)$

[3] GNN implementation is available from https://doi.org/10.5281/zenodo.5725837.

5.3 Combinatorial Efficient Global Optimization (CEGO)

The CEGO algorithm [29] builds upon the well-known EGO method [36] and adapts it for combinatorial optimization problems. EGO was initially developed for continuous domains and uses a Bayesian framework to progressively build a surrogate Gaussian process model of the objective function. In contrast, CEGO addresses combinatorial spaces by using a distance-based combinatorial surrogate model, where the classical Euclidean distance of continuous spaces is replaced with a discrete distance function which is suitable for the search space at hand.

In [30], various distance functions for the permutation space are examined. The available implementation of CEGO[4] begins by generating a few initial solutions using a max-min-distance procedure, which are then used to construct an initial surrogate model. Subsequently, a genetic algorithm utilizing operators suitable for the permutation encoding is employed to search for optimal solutions of the surrogate function. The best solution found is then evaluated using the true objective function. This evaluation allows to gain information about the objective function and to update the surrogate model maintained by CEGO. The process is repeated until a specified termination criterion is met.

Therefore, after the initial warm-up, each round of CEGO consists of: (i) optimizing the surrogate function, (ii) performing one evaluation of the true objective function, and (iii) updating the surrogate model. It is important to note that, as experimentally shown in [24], the update of the surrogate model in the permutation space is a hard problem in itself, requiring a significant amount of computational time, especially when the amount of training permutations gets large.

For additional details on the genetic algorithm settings used in CEGO, we refer interested readers to [30].

5.4 Unbalanced Mallows Model (UMM)

The UMM algorithm [33] belongs to the well known family of estimation of distribution algorithms. It iteratively alternates between learning and sampling from a Mallows model, a well-known probability distribution model for permutations [37]. At the beginning, a few solutions are randomly generated to construct the initial Mallows model. Then, in each subsequent iteration, UMM samples a permutation, evaluates it, and updates the model accordingly.

The Mallows model is characterized by a mode permutation $\pi_0 \in \mathbb{S}_n$ and a dispersion parameter $\theta \in \mathbb{R}$. The mode permutation is determined using the so-called *Unbalanced Borda* procedure, which weighs previously sampled solutions based on their fitness, ensuring that the top 10% of samples contribute 90% of the weight. The dispersion parameter θ is correlated to the expected Kendall's-τ distance $E[D]$ between a sample and the mode π_0. This expected distance is adjusted by linearly decreasing it from $\binom{n}{2}/2$ to 1 over the iterations.

For more detailed information about UMM, interested readers can refer to [33][5].

[4] CEGO implementation is available at https://cran.r-project.org/web/packages/CEGO.
[5] UMM implementation is available at https://zenodo.org/record/4500974.

6 Experiments

6.1 Experimental Settings

Experiments were conducted using the same settings as described in [13]. Ten ARP benchmark instances were generated using the instance generator and the seeds provided by the authors of [13], allowing us to reuse the results made available by them for the competitor and baseline algorithms. Specifically, two instances were considered for each size $n \in \{10, 15, 20, 25, 30\}$ using the seeds 42 and 73. It is worth noting that the seed is used by the instance generator solely to initialize its internal random number generator. The naming scheme *n_seed* is used in the rest of the article to refer to each instance.

Two different experimetnal settings are considered as follows.

- *Black-box setting*, where the competing algorithms FAT-RLS, FAT-EA, CEGO and UMM are randomly initialized, while RS is considered as baseline method.
- *Informed setting*, where the GNN heuristic is used both as baseline method and to produce reasonably good initial solutions for FAT-RLS, FAT-EA, CEGO and UMM.

In both settings, all algorithms were run 30 times per instance, with each run having a budget of 400 objective function evaluations[6]. The number of allowed function evaluations was specifically chosen to align with the primary goal of this work, i.e., studying optimization algorithms in a limited budget scenario.

Additionally, it is worth mentioning that in [13], two variants of both CEGO and UMM were examined: one that evaluates the evolved permutation directly, and another that first inverts the permutation and then evaluates it. However, as discussed in Section 2, only one of these variants is sound[7]. Therefore, in this work, we focus on a single CEGO and a single UMM: the ones referred to as CEGO-order and UMM-rank in [13] (which clearly outperformed their counterparts, CEGO-rank and UMM-order).

Regarding hyperparameter settings, since the primary goal of this experimentation is to test algorithms for an expensive and mission critical problem –the ARP–, it would not be practical to perform extensive parameter tuning on ARP instances. Indeed, in a hypothetical real-world scenario, we could have used the same time to address the target instance directly. Therefore, we selected the FAT-RLS parameters based on the lightweight tuning performed in a previous study [24] which was conducted on standard, less expensive and diverse permutation problems (LOP, PFSP, and QAP). Additionally, since FAT-EA share the same hyperparameters of FAT-RLS, we applied the same setting also to FAT-EA without further experimentation. Specifically, the hyperparameters of FAT-RLS and FAT-EA are set as follows: $d_{\text{ini}} = 0.5n$, $\beta = 1.2$, and $k = n$. Finally, the competitor algorithms UMM and CEGO were configured as described in [13].

[6] It is worth noting that the paper [14], which this work extends, contains a typo incorrectly stating that the budget was 100 objective function evaluations.

[7] Because, as stated in Section 2, necessarily one of the variant evolve the inverse of the permutation which is evaluated.

6.2 Effectiveness analysis in the black-box setting

The results collected in the black-box setting were analyzed from two different perspectives: median performances and peak performances.

For median performances, we calculated the Median Relative Percentage Deviation (MRPD) for each algorithm $\mathcal{A}$ and instance $\mathcal{I}$ as follows

$$MRPD_{\mathcal{A},\mathcal{I}} = 100 \cdot \frac{median_{\mathcal{A},\mathcal{I}} - best_{\mathcal{I}}}{best_{\mathcal{I}}}, \tag{8}$$

where $median_{\mathcal{A},\mathcal{I}}$ is the median objective value obtained by algorithm $\mathcal{A}$ over 30 executions on instance $\mathcal{I}$, while $best_{\mathcal{I}}$ is the best objective value obtained by any algorithm in this setting for instance $\mathcal{I}$.

The MRPDs are presented in Table 1. For each algorithm-instance pair, the median value may be marked with the following symbols: ■ or □ indicate that FAT-EA significantly outperformed or was significantly outperformed by the considered algorithm, respectively, while ● or ○ indicate that FAT-RLS significantly outperformed or was significantly outperformed by the considered algorithm, respectively. The statistical analyses were conducted using the well known Mann Whitney U test [38], with a significance threshold of 0.05.

Table 1: Median Relative Percentage Deviations on Black-box Experiments. Algorithms marked with ■ or □ are significantly outperformed by or outperform FAT-EA, respectively. Algorithms marked with ● or ○ are significantly outperformed by or outperform FAT-RLS, respectively.

Instance	■ FAT-EA	● FAT-RLS	UMM	CEGO	RS
10_42	11.70	12.36	13.26	9.12 □○	21.03 ■●
10_73	14.56	16.39	11.30 ○	5.07 □○	16.19
15_42	13.53	13.18	17.48 ■●	14.01	25.61 ■●
15_73	7.48	10.16	15.58 ■●	12.56 ■	24.62 ■●
20_42	9.21	9.56	20.80 ■●	15.50 ■●	26.18 ■●
20_73	14.50	14.65	30.29 ■●	23.99 ■●	33.21 ■●
25_42	15.06	14.62	29.32 ■●	24.57 ■●	34.42 ■●
25_73	13.23 ●	8.62 □	26.30 ■●	18.80 ■●	28.58 ■●
30_42	11.55 ●	6.01 □	27.26 ■●	18.95 ■●	28.58 ■●
30_73	10.59	7.95	24.46 ■●	18.96 ■●	27.31 ■●

The median performance presented in Table 1 can be commented as follows.

- For the two smallest instances with $n = 10$, both FAT-EA and FAT-RLS do not appear particularly competitive, especially compared to CEGO.
- For sizes $n \geq 15$, both FAT-EA and FAT-RLS significantly outperform UMM in all the instances and CEGO in almost all the instances. This is the most important observation.
- In the comparison between FAT-EA and FAT-RLS, the evolutionary search scheme of FAT-EA seems beneficial on smaller instances, while the randomized local search of FAT-RLS appears more effective for larger instances.

For the peak performances, Table 2 presents the objective values returned by the best execution of each algorithm for each instance. Values in bold represent, for each instance, the best objective value ever achieved by a black-box algorithm.

Table 2: Best Objective Values on Black-box Experiments. Bolded results represent the best performance for each instance.

Instance	FAT-EA	FAT-RLS	UMM	CEGO	RS
10_42	**346.7**	**346.7**	**346.7**	**346.7**	389.6
10_73	338.3	329.4	329.9	**324.7**	343.6
15_42	519.1	**505.4**	516.9	515.7	583.1
15_73	524.3	**515.1**	530.7	523.7	573.0
20_42	701.2	**698.9**	729.1	726.9	777.8
20_73	711.8	**676.3**	810.1	730.8	813.2
25_42	865.5	**837.4**	966.8	945.9	1075.1
25_73	913.9	**889.0**	1028.7	988.7	1089.6
30_42	1074.3	**1062.3**	1260.6	1183.0	1271.0
30_73	1115.7	**1098.2**	1232.8	1212.1	1344.2

Table 2 shows that FAT-RLS obtained the best results in 9 out of 10 instances, while in the remaining instance (10_73) it obtained the second best peak performance after CEGO. The best results of FAT-EA are slightly worse than those of FAT-RLS, even in instances where FAT-EA has the best median result. Moreover, as also partially indicated by Table 1, where the MRPDs of both FAT-RLS and FAT-EA improve as n increases, it appears that both algorithms are capable of achieving competitive results across the entire benchmark spectrum, although they show a lack of performance when the instance size is smaller.

To further validate the provided observations, Figure 2 presents a boxplot graph illustrating the distributions of the relative percentage deviations obtained in each run (calculated as the difference between the objective value obtained in a run and the best objective value for the instance, divided by the latter and expressed as a percentage), grouped by instance size.

The graph confirms the previous discussion, also highlighting that FAT-EA is sometimes more robust than FAT-RLS, as observed in the cases $n = 10, 20, 25$.

6.3 Effectiveness analysis in the informed setting

The median and peak performance analyses conducted in Section 6.3 were also carried out in the informed scenario, where the algorithms are initialized using the solution obtained through the GNN heuristic.

Tables 3 and 4 present, respectively, the median and best results achieved by FAT-EA, FAT-RLS, UMM, CEGO, and GNN in the informed setting.

The results indicate that all four meta-heuristics improved upon the initial solution provided by GNN. However, the performance gains over the baseline algorithm are slightly less pronounced than in the black-box setting. Likely, this is because the initial solution is heuristically constructed rather than randomly generated.

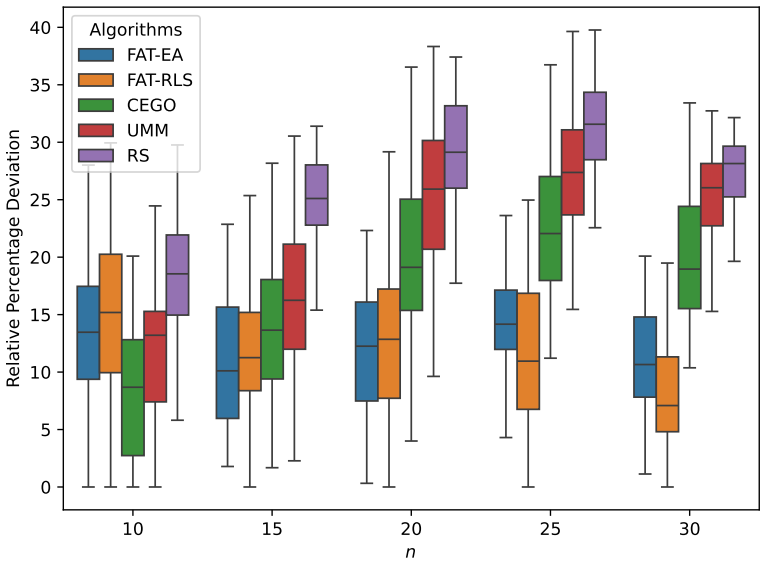

Fig. 2: Distributions of the relative percentage deviations recorded in the black-box experiments group by instance size n.

Table 3: Median Relative Percentage Deviations on Informed Experiments. Algorithms marked with ■ or □ are significantly outperformed by or outperform FAT-EA, respectively. Algorithms marked with ● or ○ are significantly outperformed by or outperform FAT-RLS, respectively.

Instance	■ FAT-EA	● FAT-RLS	UMM	CEGO	GNN
10_42	9.97 ●	9.97 □	10.22 ■●	8.68 □○	12.86 ■●
10_73	19.34	20.46	18.69 □○	10.73 □○	22.67 ■●
15_42	1.32 ●	0.96 □	2.21 ■●	1.32 ●	3.51 ■●
15_73	4.42	2.95	6.44 ■●	2.79 □	12.79 ■●
20_42	6.26	3.36	16.25 ■●	5.95	22.12 ■●
20_73	4.53 ●	3.20 □	7.33 ■●	3.93 □	8.27 ■●
25_42	9.18 ●	7.69 □	14.29 ■●	8.37 ●	17.50 ■●
25_73	3.84 ●	0.88 □	13.72 ■●	7.63 ■●	13.80 ■●
30_42	4.22	4.08	7.27 ■●	4.03	8.78 ■●
30_73	3.23 ●	2.06 □	6.85 ■●	1.18 □○	7.63 ■●

FAT-EA is significantly outperformed by FAT-RLS in 6 instances out of 10, though it obtained four new best known solutions, two more than FAT-RLS.

In the comparison with respect to the informed UMM, both FAT-EA and FAT-RLS significantly outperformed UMM in 9 out of 10 instances, mirroring the trend observed in the black-box setting. Conversely, the comparison with respect to informed CEGO is more balanced than in the black-box case for FAT-RLS. Indeed, FAT-RLS

Table 4: Best Objective Values on Informed Experiments. Bolded results represent the best performance for each instance.

Instance	FAT-EA	FAT-RLS	UMM	CEGO	GNN
10_42	381.3	381.3	381.3	**346.7**	391.3
10_73	370.2	385.4	367.6	**324.7**	398.3
15_42	493.1	493.1	**490.9**	491.4	508.1
15_73	**511.0**	512.3	532.0	519.9	576.4
20_42	709.7	**689.3**	729.7	707.2	841.7
20_73	**638.7**	659.1	672.8	652.5	691.5
25_42	833.3	**805.3**	895.8	865.7	946.3
25_73	**807.0**	807.5	885.7	863.7	918.3
30_42	**1040.4**	1045.3	1093.9	1065.2	1131.7
30_73	961.5	959.6	1002.0	**952.1**	1024.7

significantly outperformed CEGO in 3 instances and was significantly outperformed by CEGO in 3 other instances.

By comparing the results of Tables 2 and 4, it is worth noting that on the two instances of size $n = 10$, GNN and the informed FAT-EA and FAT-RLS did not match the results of RS and their black-box counterparts. This likely indicates that the $n = 10$ instances have a relatively "shallow landscape" where random search is sufficient to provide good results.

Furthermore, as in the black-box experiments, Figure 3 shows a boxplot graph illustrating the distributions of the relative percentage deviations obtained in each run grouped by instance size.

Interestingly, comparing Figures 3 and 2 reveals that FAT-EA is less robust than FAT-RLS when the two algorithms are seeded with a good initial solution rather than a random one.

6.4 Computational time analysis

Besides effectiveness, it is also important to analyze the efficiency of the competing algorithms in terms of the computational time required to complete an execution within the given budget of 400 function evaluations. In fact, algorithms with lower computational overhead are generally preferred in real-world mission-critical scenarios.

To this end, we present in Table 5 the average computational time (in seconds) observed for each algorithm and instance size, based on their executions on instances of that size. These measurements were taken on an machine equipped with an Intel-Core i9 11900F processor at 2.50GHz, 32GB of memory, and running Ubuntu 22.04.

Table 5 clearly shows that the computational time required for FAT-EA and FAT-RLS is almost identical to that of Random Search (RS), confirming that the algorithmic overhead of the two proposed methods is practically negligible.

A more interesting observation arises when comparing these results with UMM and CEGO. UMM requires approximately 300 seconds (5 minutes) more than FAT-EA and FAT-RLS when $n = 10$, and this gap increases to around 1000 seconds ($\sim$ 16 minutes) when $n = 30$. CEGO, on the other hand, is significantly slower, with an average runtime exceeding 38 hours.

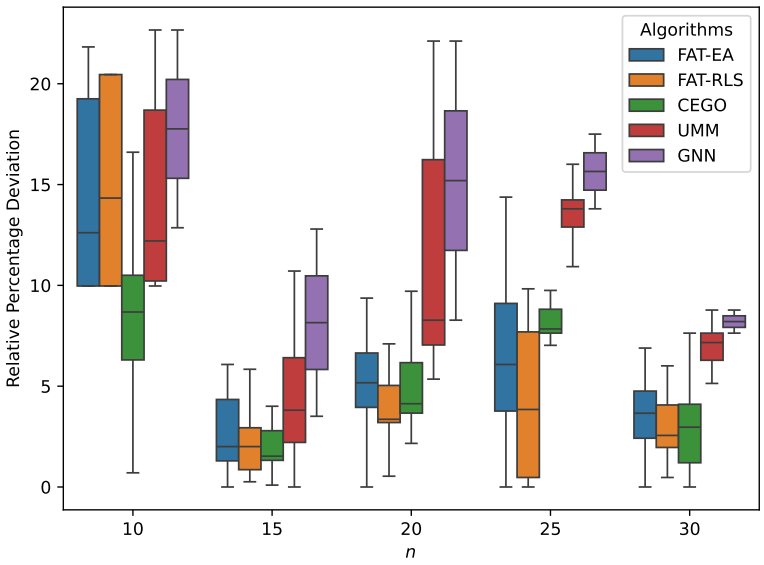

Fig. 3: Distributions of the relative percentage deviations recorded in the informed experiments group by instance size n.

Table 5: Average computational time in seconds required by the runs of the competing algorithms grouped by instance size.

n	FAT-EA	FAT-RLS	UMM	CEGO	RS
10	1 293	1 273	1 587	142 149	1 185
15	1 818	1 838	2 628	144 642	1 737
20	2 557	2 526	3 055	139 059	2 485
25	3 181	3 083	3 697	142 062	2 999
30	3 810	3 894	4 812	135 426	3 608

Notably, unlike the other algorithms, CEGO's runtime does not scale with the instance size. This is likely because its model-building phase involves solving a combinatorial problem, which is itself inherently complex and computationally demanding.

7 Conclusion and Future Work

In this study, we experimentally studied the performance of two algorithms, FAT-EA and FAT-RLS, in tackling the Asteroid Routing Problem (ARP).

Both the proposed algorithms are formed by three key components: an elitist and trajectory-based search scheme, an adaptive regulation of the perturbation strength, and a tabu-based mechanism. The difference between FAT-RLS and FAT-EA lies in the search scheme adopted: FAT-RLS uses a randomized local search which performs

exactly one insertion perturbation per iteration, while FAT-EA adopts the $(1 + 1)$ evolutionary search which applies at least one insertion perturbation per iteration. The rationale behind FAT-EA is to trade off some of the exploitation observed in the FAT-RLS scheme in exchange for the potential to escape local optima.

Experiments were conducted on widely used benchmark instances for the ARP, comparing the performance of FAT-EA and FAT-RLS with two established probabilistic approaches: UMM, an estimation of distribution algorithm designed for permutation problems, and CEGO, a Bayesian method for combinatorial optimization. The experiments were carried out in two scenarios: a black-box setting and an informed setting. In the black-box setting, all competing algorithms are seeded with randomly generated permutations, while in the informed setting, the algorithms are initialized with solutions provided by a heuristic method specifically designed for the ARP.

The results show that, while FAT-EA is of some help in smaller instances in the black-box scenario and in achieving peak performance on certain instances in the informed scenario, FAT-RLS, despite its simpler design, consistently outperforms the other algorithms. In fact, although a definitive conclusion cannot be drawn from the comparison between FAT-RLS and CEGO in the informed scenario, the significantly smaller computational overhead of FAT-RLS –several orders of magnitude less than that of CEGO– certainly gives it an advantage.

Therefore, these results lead to a key takeaway: simple algorithms, primarily based on very strong exploitation approaches, can be a viable alternative to more sophisticated techniques when addressing limited budget optimization of mission critical combinatorial problems.

The analyzed results also reveal some lack of robustness in certain cases for both FAT-RLS and FAT-EA, suggesting opportunities for improvement in this regard. Finally, another interesting avenue for future work would be to study the proposed algorithms also on other expensive and real world permutation problems.

Declarations

Competing Interests. The author declare there is no conflict of interest.

Funding Information. Not applicable.

Author Contributions. Conceptualization: Valentino Santucci; Methodology: Valentino Santucci; Formal analysis and investigation: Valentino Santucci; Writing: Valentino Santucci.

Data Availability Statement. Not applicable.

Research Involving Human and /or Animals. Not applicable.

Informed Consent. Not applicable.

References

[1] Frazier, P.I.: A tutorial on bayesian optimization. arXiv preprint arXiv:1807.02811 (2018)

[2] Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems **25** (2012)

[3] Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., Deng, S.-H.: Hyperparameter optimization for machine learning models based on bayesian optimization. Journal of Electronic Science and Technology **17**(1), 26–40 (2019)

[4] Eriksson, D., Pearce, M., Gardner, J., Turner, R.D., Poloczek, M.: Scalable global optimization via local bayesian optimization. Advances in Neural Information Processing Systems **32**, 5496–5507 (2019)

[5] Moriconi, R., Deisenroth, M.P., Kumar, K.S.: High-dimensional bayesian optimization using low-dimensional feature spaces. Machine Learning **109**(9), 1925–1943 (2020)

[6] Agresta, A., Baioletti, M., Biscarini, C., Milani, A., Santucci, V.: Evolutionary algorithms for roughness coefficient estimation in river flow analyses. In: Proceedings of the 25th International Conference on the Applications of Evolutionary Computation (EvoApps 2021), pp. 795–811. Springer, Cham (2021)

[7] Agresta, A., Baioletti, M., Biscarini, C., Caraffini, F., Milani, A., Santucci, V.: Using optimisation meta-heuristics for the roughness estimation problem in river flow analysis. Applied Sciences **11**(22) (2021) https://doi.org/10.3390/app112210575

[8] Raponi, E., Wang, H., Bujny, M., Boria, S., Doerr, C.: High dimensional bayesian optimization assisted by principal component analysis. In: Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I 16, pp. 169–183 (2020). Springer

[9] Santoni, M.L., Raponi, E., Leone, R.D., Doerr, C.: Comparison of high-dimensional bayesian optimization algorithms on bbob. ACM Transactions on Evolutionary Learning **4**(3), 1–33 (2024)

[10] Santucci, V., Baioletti, M., Milani, A.: Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. IEEE Transactions on Evolutionary Computation **20**(5), 682–694 (2015)

[11] Santucci, V., Ceberio, J.: On the use of the doubly stochastic matrix models for the quadratic assignment problem. Evolutionary Computation, 1–30 (2025)

[12] Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS Journal on Computing **25**(2), 346–363 (2013)

[13] López-Ibáñez, M., Chicano, F., Gil-Merino, R.: The asteroid routing problem: A benchmark for expensive black-box permutation optimization. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), pp. 124–140 (2022). Springer

[14] Santucci, V., *et al.*: A simple yet effective algorithm for the asteroid routing problem. In: Proceedings of the 16th International Joint Conference on Computational Intelligence, pp. 50–59 (2024). SCITEPRESS

[15] Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+ 1) evolutionary algorithm. Theoretical Computer Science **276**(1-2), 51–81 (2002)

[16] Scharnow, J., Tinnefeld, K., Wegener, I.: The analysis of evolutionary algorithms on sorting and shortest paths problems. Journal of Mathematical Modelling and Algorithms **3**, 349–366 (2005)

[17] Santucci, V., Baioletti, M., Milani, A.: An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization. Swarm and Evolutionary Computation **55**, 100673 (2020)

[18] Ceberio, J., Santucci, V.: Model-based gradient search for permutation problems. ACM Transactions on Evolutionary Learning and Optimization **3**(4), 1–35 (2023)

[19] Santucci, V., Ceberio, J.: Using pairwise precedences for solving the linear ordering problem. Applied Soft Computing **87** (2020) https://doi.org/10.1016/j.asoc.2019.105998

[20] Santucci, V., Baioletti, M., Milani, A.: Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. IEEE Transactions on Evolutionary Computation **20**(5), 682–694 (2016) https://doi.org/10.1109/TEVC.2015.2507785

[21] Kuhn, H.W.: The hungarian method for the assignment problem. Naval research logistics quarterly **2**(1-2), 83–97 (1955)

[22] Koopmans, T.C., Beckmann, M.J.: Assignment Problems and the Location of Economic Activities. Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University (1955)

[23] Loiola, E.M., De Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., Querido, T.: A survey for the quadratic assignment problem. European journal of operational research **176**(2), 657–690 (2007)

[24] Santucci, V., Baioletti, M.: A fast randomized local search for low budget optimization in black-box permutation problems. In: 2022 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2022). IEEE

[25] Baioletti, M., Milani, A., Santucci, V.: Variable neighborhood algebraic differential evolution: An application to the linear ordering problem with cumulative costs. Information Sciences **507**, 37–52 (2020)

[26] Izzo, D.: Revisiting lambert's problem. Celestial Mechanics and Dynamical Astronomy **121**, 1–15 (2015)

[27] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., *et al.*: Scipy 1.0: fundamental algorithms for scientific computing in python. Nature methods **17**(3), 261–272 (2020)

[28] Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theoretical Computer Science **378**(1), 32–40 (2007)

[29] Zaefferer, M., Stork, J., Friese, M., Fischbach, A., Naujoks, B., Bartz-Beielstein, T.: Efficient global optimization for combinatorial problems. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 871–878 (2014)

[30] Zaefferer, M., Stork, J., Bartz-Beielstein, T.: Distance measures for permutations in combinatorial efficient global optimization. In: International Conference on Parallel Problem Solving from Nature, pp. 373–383 (2014). Springer

[31] Doerr, B., Neumann, F.: A survey on recent progress in the theory of evolutionary algorithms for discrete optimization. ACM Transactions on Evolutionary Learning and Optimization **1**(4), 1–43 (2021)

[32] Chopard, B., Tomassini, M.: An Introduction to Metaheuristics for Optimization. Springer, ??? (2018)

[33] Irurozki, E., López-Ibáñez, M.: Unbalanced mallows models for optimizing expensive black-box permutation problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 225–233 (2021)

[34] Eberl, M.: Fisher-yates shuffle. Arch. Formal Proofs **2016**, 19 (2016)

[35] Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M. II: An analysis of several heuristics for the traveling salesman problem. SIAM journal on computing **6**(3), 563–581 (1977)

[36] 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)

[37] 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 (2013)

[38] Hollander, M., Wolfe, D.A., Chicken, E.: Nonparametric Statistical Methods vol. 751. John Wiley & Sons, ??? (2013)