Perturbed Decomposition Algorithm applied to the multi-objective Traveling Salesman Problem

Marek Cornu*, Tristan Cazenave*, Daniel Vanderpooten*

PSL, Universite Paris-Dauphine, LAMSADE - CNRS UMR 7243, 75775 Paris Cedex 16, France

Abstract

Dealing with multi-objective combinatorial optimization, this article proposes a new multi-objective set-based meta-heuristic named Perturbed Decomposition Algorithm (PDA). Combining ideas from decomposition methods, local search and data perturbation, PDA provides a 2-phase modular framework for finding an approximation of the Pareto front. The first phase decomposes the search into a number of linearly aggregated problems of the original multi-objective problem. The second phase conducts an iterative process: aggregated problems are first perturbed then selected and optimized by an efficient single-objective local search solver. Resulting solutions will serve as a starting point of a multi-objective local search procedure, called Pareto Local Search. After presenting a literature review of meta-heuristics on the multi-objective symmetric Traveling Salesman Problem (TSP), we conduct experiments on several instances of the bi-objective and tri-objective TSP. The experiments show that our proposed algorithm outperforms the best current methods on this problem.

Keywords: Multi-objective combinatorial optimization, Multi-objective Traveling Salesman Problem, Meta-heuristics, Pareto Local search, Decomposition algorithm, Data perturbation

1. Introduction

In multi-objective (MO) combinatorial optimization, several criteria are taken into account. When the preferences of the decision maker are not known, a far challenge is to generate the set of non-dominated points, so that no improvement on any objective is possible without sacrificing on at least another objective. Even for moderately-sized problems, it is usually computationally prohibitive to identify this set for two major reasons. First, the decision version of most MO combinatorial optimization (MOCO) problems is $\mathcal{NP}$-complete, even if the underlying single-objective version is in $\mathcal{P}$. Second, most MOCO problems are intractable in the sense that the number of non-dominated points can be exponential in the size of the instance (see [1] for more details on MO optimization).

To handle these difficulties, researchers have been interested in developing heuristic algorithms, such as meta-heuristics. In particular, MO local search (LS) algorithms are among the most successful meta-heuristics for tackling MOCO problems. The currently best performing LS meta-heuristics for MOCO problems typically involve different algorithmic components that are combined into an upper-level framework.

This article presents the Perturbed Decomposition Algorithm (PDA) algorithm. The framework of PDA combines single-objective LS and MO LS techniques, MO decomposition [2] and data perturbation [3]. To validate our approach, we conduct experiments on several instances of the bi-objective and tri-objective symmetric Traveling Salesman Problem (MOTSP) of different types and sizes.

*Corresponding author
Email address: marek.cornu@lamsade.dauphine.fr (Marek Cornu)

The paper is organized as follows. Section 2 first recalls the formal definition of a MOCO problem and fundamental definitions. Then, we introduce basic techniques of local search algorithms both in the single-objective and MO cases, and propose a literature review of meta-heuristics on the MO Traveling Salesman Problem (MOTSP). Section 3 describes the proposed algorithm, PDA and its three algorithmic components. Section 4 is devoted to the presentation of the MOTSP, the benchmark instances, a sensitivity analysis of PDA on the data perturbation, and the computational experiments with a comparison between PDA and the best current methods on MOTSP. Section 5 concludes on the contributions and perspectives of PDA.

2. Preliminaries

We first recall basic definitions of multi-objective optimization (Section 2.1), then the main concepts of single-objective local search (Section 2.2) and its extension to the MO case (Section 2.3), and finally present a review of meta-heuristics applied to MOTSP (Section 2.4).

2.1. Multi-objective combinatorial optimization

Let E be a finite set of q elements $E := \{e_1, ...e_q\}$, defining a combinatorial structure. Let $c_j : E \to \mathbb{R}\ j = 1, ..., p$ be the p cost functions that map each element of E with a vector of p costs, and $c = (c_1, ...c_p)$ be the *MO cost function*.

Considering the *minimization sum* version, a MOCO problem is defined as:

$$\begin{cases} \min f(x) = (f_1(x), ..., f_p(x)) \\ \text{subject to } x \in X \end{cases} \tag{1}$$

where $X \subset 2^E = \{0,1\}^q$ is the *feasible set* and the p potentially conflicting *objective functions* $f_j : X \to \mathbb{R}$ are such that $f_j(x) = \sum_{e \in x} c_j(e)$ for each $j = 1, ..., p$.

Let $Z \subset \mathbb{R}^p$ be the *objective space* and $Z_X := f(X) = \{z \in Z : z_j = f_j(x)$ for $j = 1,...,p$ and $x \in X\}$ the *outcome set,* mapping each feasible solution *x* to a point $z = f(x)$ of the objective space Z.

Let $z, z' \in Z$ be two points in the objective space. We say that z *dominates* z', denoted by $z \leq z'$, if $z_j \leq z'_j$ for each *j*=1,...,*p* and there exists $i \in \{1, ..., p\}$ such that $z_i < z'_i$.

We say that z weakly dominates z', denoted by $z \leqq z'$, if $z_j \leq z'_j$ for each *j*=1,...,*p*.

A point $z \in Z_X$ is called *non-dominated* if and only if there is no other point $\widehat{z} \in Z_X$ such that $\widehat{z} \leq z$. A feasible solution $x \in X$ is called *efficient* if its image in the objective space is non-dominated.

The set of all non-dominated points Z_{nd} is called *non-dominated set* or *Pareto front*. The set of all efficient solutions X_e is called the *efficient set*.

The *ideal point* is the point $z^* = (z_1^*, ..., z_p^*)$ which has the best values for each objective, i.e. such that $z_j^* := \min_{x \in X} f_j(x)\ j = 1, ..., p$.

Two common aggregation procedures are the weighted sum and the augmented weighted Tchebychev aggregations.

Let $\lambda = (\lambda_1, ..., \lambda_p)$ be a weighting vector (called *weight*). The *weighted sum problem* is given by:

$$\begin{cases} \min \text{ ws}(x, \lambda) = \sum_{j=1}^{p} \lambda_j f_j(x) \\ \text{subject to } x \in X \end{cases} \tag{2}$$

The resulting aggregated problem is a single-objective instance of the original MO problem.

Supported efficient solutions are optimal solutions of a weighted sum problem for some vector $\lambda > 0$. The images in objective space of the supported efficient solutions correspond to the non-dominated points which are located on the convex hull of Z_X. *Non-supported efficient solutions* are efficient solutions that are not optimal solutions for any weighted sum problem with $\lambda > 0$. Non-supported non-dominated points are located in the interior of the convex hull of Z_X.

Let $\lambda = (\lambda_1, ..., \lambda_p) \geq 0$ be a weight. The *weighted augmented Tchebychev problem* is given by:

$$\begin{cases} \min \text{ wat}(x, \lambda, z^*) = \max_{j=1,...,p} \lambda_j (f_j(x) - z_j^*) + \varepsilon \sum_{j=1}^{p} \lambda_j (f_j(x) - z_j^*) \\ \text{subject to } x \in X \end{cases} \tag{3}$$

where $\varepsilon > 0$ is a fixed small positive real. It is well known that any optimal solution of (3) is an efficient solution.

In order to sample the non-dominated set, it may be useful to generate a set of weights and solve a given aggregation procedure for each of these weights.

A technique to produce a number of equally dispersed weights is the Maximally Dispersed Set of weights (MDS) - also called *set of normalized weights* - of Steuer [4]. Given a parameter $h \in \mathbb{N}^*$, this technique provides the set of weights Λ given by:

$$\begin{cases} \Lambda = \{\lambda = (\lambda_1, ..., \lambda_p) : \sum_{j=1}^{p} \lambda_j = 1, \lambda_j \in \{\frac{0}{h}, \frac{1}{h} ..., \frac{h}{h}\}, j = 1, ..., p\} \\ \text{such that } \min_{\lambda^a, \lambda^b \in \Lambda : \lambda^a \neq \lambda^b} \sum_{j=1}^{p} \left|\lambda_j^a - \lambda_j^b\right| = \frac{2}{h} \end{cases} \tag{4}$$

where $|\Lambda| = \binom{h+p-1}{h}$. We will refer several times to the MDS later in the paper.

2.2. Single-objective local search

Let $d : (X, X) \rightarrow \mathbb{N}^+$ be a distance measure between two feasible solutions. For any $k \geq 1$, we define the *k-neighborhood structure* $\mathcal{N}_k : X \rightarrow 2^X$ as $\mathcal{N}_k(x) = \{y \in X : d(x, y) \leq k\}$. $\mathcal{N}_k(x)$ assigns to $x \in X$ the set of its *neighbors*. Changing from $x \in X$ to $y \in \mathcal{N}_k(x)$ is called a *neighborhood move*.

For some optimization problems, all feasible solutions contain the same number of elementary components from E, and $\mathcal{N}_k$ is called a *k-exchange* neighborhood structure. This is true for the TSP, where a solution $x \in X$, which represents a Hamiltonian cycle in a complete graph of *n* cities, is composed of *n* edges. In this context, let $y \in X$ be another feasible solution, then $y \in \mathcal{N}_k(x)$ if y is an Hamiltonian cycle obtained from x by exchanging at most k edges. The most elementary neighborhood structure for the TSP is the so-called *2-edge-exchange neighborhood*.

Given a single-objective minimization problem with objective function $g : X \rightarrow \mathbb{R}$ and a *k*-neighborhood structure $\mathcal{N}_k$ ($k \geq 1$), a *local search routine* explores, at each step, the neighborhood of the current solution $x \in X$ so as to find a neighbor $y \in \mathcal{N}_k(x)$ such that $g(y) < g(x)$. It stops in a *local optimum*, for which no improving neighbor can be found.

S*tochastic local search* (SLS) is a general concept of local search algorithm restarting the local search routine by use of a stochastic process.

A *perturbation move*, called *kick* in this article, is a technique with the aim of escaping from a local optimum. Let $x \in X$ be a local optimum according to $\mathcal{N}_k$ ($k \geq 1$). A kick consists in applying a random move from x in a larger size neighborhood $\mathcal{N}_l$ ($l > k$). The perturbation neighborhood size l has to be sufficiently large to lead to a different attraction basin than the one generated by $\mathcal{N}_k$ from x.

An *iterated local search* (ILS) algorithm [5, 6] is an SLS. It builds a *sequence* of locally optimal solutions by iteratively applying a kick to the current locally optimal solution and restarting a local search routine from this modified solution.

2.3. Multi-objective local search

In MOCO, we search for a set of solutions rather than a single one. Therefore, a LS adapted to solve MOCO problems should manage sets of solutions, instead of a unique solution. Single-objective local search techniques have been adapted to MO spaces. We define the *neighborhood of a set* of feasible solutions as the union of the neighborhoods of each solution.

Pareto Local Search (PLS) [7, 8, 9] is the MO extension of the local search routine. Given a neighborhood structure, PLS starts from a set of solutions and iteratively improves this set by memorizing neighbors whose images are not weakly dominated by any points found so far. PLS stops when all neighbors are non-efficient, and is stuck in a *locally efficient set*.

To our knowledge, two different versions of PLS have been published. The only difference between the two versions is that the neighborhood is explored either from the whole set of solutions (first version), or from a single solution (second version) [10].

The first version, called PLS1 in this article, has been introduced by Talbi et al. [7]. Starting from a set of solutions, PLS1 explores the neighborhood of each solution, and retains in an auxiliary set all the neighbors whose images are not weakly dominated by any points found so far. A new iteration starts from this auxiliary set and PLS1 continues this process until no more new non-weakly dominated neighbors have been identified. The use of the auxiliary set prevents the exploration of the neighborhood of an already visited solution. Initially, PLS1 has been applied to the MO Flow Shop Scheduling Problem [7] and later to the bi-objective TSP [8].

The second version, called PLS2 in this article, has been introduced by Paquete et al. [9]. Starting from a current set of solutions S, PLS2 first selects at random a non-visited solution, explores its neighborhood and add in S all the neighbors whose images are not weakly dominated by any points in S. PLS2 stops when all the solutions have been visited.

A comparison study between PLS1 and PLS2 on bi-objective TSP instances has been conducted in [11]. When PLS is launched from a randomly generated solution, then PLS2 obtains better results than PLS1. On the contrary, when PLS is launched from a good initial set of solutions, then PLS1 leads to an approximation of slightly better quality than PLS2 with a comparable computational time.

The method we propose, PDA, will conduct PLS from a good initial set of solutions. Therefore, PLS1 will be used as PLS in this article. An improved version of the original PLS1 is used in PDA and detailed in Section 3.4.

SLS for MO spaces are called Stochastic Pareto Local Search (SPLS) [12, 13].

Iterated PLS (IPLS) [14, 15] is a SPLS starting from an initial set of solutions, which iteratively performs a PLS. At the end of each PLS, some of the current potentially efficient solutions are selected and perturbed with a kick, so as

to form a new starting set. This technique can escape from a locally efficient set.

2.4. Literature review of meta-heuristics on the MOTSP

This section reports a non-exhaustive literature review of meta-heuristics approaches on MOTSP. Some methods are more detailed than others because they will be used to compare with our approach, PDA.

To our knowledge, Jaszkiewicz is the first author having published approximations of Pareto sets of bi-objective TSP instances. In [16], he proposed an improved version of the MO Genetic local search (MOGLS) initially proposed by Ishibushi and Murata [17, 18], who applied it on MO Flow Shop Scheduling Problems.

Then Paquete and Stützle present in [19] their Two-Phase Local Search (TPLS) and Pareto Double TPLS (PD-TPLS) methods. TPLS and PD-TPLS first produce a set of weights using the MDS method (see Section 2.1) then solve the related weighted sum problems using a domain-specific ILS method. PD-TPLS adds an additional step by searching for the potentially efficient solutions in the neighborhood of the solutions previously found. In [20], Dubois-Lacoste et al. analyze and improve the anytime behavior of TPLS. To our knowledge, TPLS and PD-TPLS have been tested only on bi-objective instances and it is shown in [19] that the PD-TPLS method gives better results than the MOGLS method.

Jaszkiewicz and Zielniewicz have experimented on bi-objective instances the Pareto Memetic Algorithm [21] (PMA) and found better results than MOGLS.

The next year, Kumar and Singh [22] present a memetic algorithm and find comparable results to MOGLS and PD-TPLS.

Paquete and Stützle generalize the PD-TPLS method in [23] by describing a generic class of MO SLS algorithms, composed of different algorithmic components. They investigate the importance and behavior of those components by experiments on bi-objective and tri-objective TSP instances. They propose an algorithm, called PD-TPLS-l in this paper, obtained by an experimental optimization of the components configuration. Using PD-TPLS-l, the authors find better results than MOGLS in the tri-objective case.

Lust and Teghem designed a bi-objective SLS method, the Two phase Pareto Local Search (2PPLS) [11] and its variants 2PPLS+P [11], 2PPLS-SpeedP1 [24]. The main idea of 2PPLS is that PLS is a powerful tool to generate potentially efficient solutions. However, instead of starting the method with randomly generated solutions (thus of poor quality) as done in previous works (see [25] for example), the first phase of 2PPLS generates a set of high quality solutions covering well the Pareto front by approximating the supported efficient solution set using the standard dichotomic scheme [26, 27]. PLS is used in the second phase to generate a more accurate approximation of the Pareto front. The authors experimentally show that this first phase drastically increases the final quality and convergence speed of PLS. Experiments show that 2PPLS and its variants outperform MOGLS and PMA on tested bi-objective instances.

The Evolutionary MO Simulated Annealing Algorithm (EMOSA) [28] of Li and Landa-Silva is compared to other MOSA-like algorithms and obtains better results on all bi-objective and tri-objective tested instances. However, contrary to the most efficient methods presented in this section, EMOSA does not use essential TSP-specific LS speed-up techniques, such as fixed radius candidate lists [29], don't look bits [30] and a neighborhood greater than a 2-exchange one for ILS. See [19, 23] for a comparison between the use of a 2-exchange and a 3-exchange neighborhoods in a MO SLS, and [24] for SPLS speed-up techniques. Therefore, EMOSA is not a competitive algorithm compared to the best current methods on MOTSP, both in bi-objective and tri-objective instances, making the comparison not relevant with our proposed approach, PDA.

Various MO Ant Colony Optimization (MOACO) algorithms have been proposed in recent years (see [31, 32] among others). López-Ibáñez and Stützle [33] propose a framework that suffices to describe most MOACO algorithms proposed so far. The authors tested different optimized configurations of this framework on Euclidean bi-objective TSP, and found better MOACO algorithms than those available in the literature. However, as EMOSA, results of MOACO are not comparable to the best current methods on MOTSP.

Liefooghe et al. [10] present a comparison between different strategies of neighborhood exploration for Dominance-based Multiobjective Local Search algorithms (DMLS), which is a generalization of PLS. Experiments have been conducted on bi-objective and tri-objective TSP instances.

Murata et al. [34] introduced the concept of *cellular structure* for MO genetic algorithms. This concept has been generalized and renamed *decomposition* by Zhang and Li [2] with their MO Evolutionary Algorithm based on Decomposition (MOEA/D) method. MOEA/D decomposes a MOCO problem into a fixed number of equally dispersed aggregated problems and optimizes them simultaneously in a collaborative way. Each of these problems is defined by a weight λ giving a unique search direction, and maintains a best-so-far solution (*incumbent*) according to the corresponding chosen aggregated function (e.g. $\text{ws}(\cdot, \lambda)$, $\text{wat}(\cdot, \lambda, z^*)$, etc.) for the entire duration of the run.

The Multi-objective Memetic algorithm based on decomposition (MoMad) [35] of Ke et al. is a recent method combining decomposition and PLS. As a decomposition method, MoMad is an iterative algorithm which first decomposes the MOCO problem into several aggregated problems (called *decomposition phase* in this article). For MOTSP, the chosen aggregated function is the weighted sum ws. As in 2PPLS, an ILS is applied on each single-objective problem in order to initialize the incumbents with high quality potentially efficient solutions covering well the Pareto front. Then the *iterative phase* begins. At each iteration, MoMad conducts a PLS. When the PLS stops, it applies again an ILS from each incumbent. The potentially efficient solutions generated compose a starting set for the next PLS. Note that incumbents are regularly updated in a collaborative way, by comparison with the other ones.

This iterative mechanism of restarting PLS is not new and has already been used in [14, 15] with the notion of IPLS. However, MoMad hybridizes IPLS with the decomposition methodology. Recent experiments (see [35]) on several bi-objective instances of different types and sizes have shown that MoMad outperforms the 2PPLS method on tested instances.

MOEA/D-ACO [36] combines the MOEA/D method with a MOACO algorithm. In [35], the authors show that MOEA/D-ACO is outperformed by both 2PPLS and MoMad on bi-objective instances.

To summarize MoMad is the best known method on bi-objective TSP instances. Besides, few works have been proposed to efficiently tackle tri-objective TSP instances. Among these, PD-TPLS-l is the best known algorithm for tri-objective instances. Because MoMad shows very good results on the bi-objective case, we will compare:

- PDA and MoMad [35] on bi-objective instances.
- PDA, MoMad [35] and PD-TPLS-l [23] on tri-objective instances.

3. The Perturbed Decomposition Algorithm

This section describes the method presented in the present work, the Perturbed Decomposition Algorithm (PDA). Section 3.1 presents the general framework of PDA, while the following subsections describe the main components. Algorithms related to each component of PDA are depicted in pseudo-code. The symbols $\downarrow$, $\uparrow$, and $\updownarrow$ specify the parameter transmission, respectively in, out and in-out.

3.1. General framework

PDA combines ideas from decomposition (see [2] for more details), SPLS algorithms and data perturbation (see [37, 38, 3] for more details). Basically, PDA first decomposes the search into a number of single-objective problems, called *sub-problems*. Then it iteratively runs ILS on perturbed sub-problems to provide a starting solution set for PLS. Algorithm 1 reports the main steps of PDA. A general description follows.

The initialization of PDA (**Step 1**) begins by the *decomposition step* (**Step 1.1**). It produces a set of K sub-problems $\Pi = (\pi^1, ..., \pi^K)$ where K depends on the value of the decomposition parameter $h \in \mathbb{N}^*$, chosen by the user (see Section 3.2).

For each $k = 1, ..., K$, a sub-problem $\pi^k \in \Pi$ is a tuple $(\lambda^k, c^k, x^k, r^k)$ composed of four elements:

- A unique weight $\lambda^k = (\lambda^k_1, ..., \lambda^k_j)$.
- A single-objective cost function $c^k : E \to \mathbb{R}$ such that $c^k(e) = \sum_{j=1}^{p} \lambda^k_j c_j(e)$ for all $e \in E$, where $c : E \longrightarrow \mathbb{R}^p$ is the MO cost function related to the addressed MOCO problem. So c^k defines the costs of the weighted-sum problem $\text{ws}(\cdot, \lambda^k)$.
- An *incumbent* $x^k := \arg\min\{\text{ws}\left(x, \lambda^k\right) : x \in A\}$ where A is the set of potentially efficient solutions found so far, called *archive*. x^k is initialized by a domain-dependent heuristic. ILS will start from x^k and optimize the single-objective problem related to c^k.
- A cumulative number r^k, counting the number of potentially efficient solutions found by ILS from π^k.

Each sub-problem provides a unique search direction and we claim that focusing optimization on the same directions during the entire duration of the run, as Decomposition algorithms [2] like MoMad usually do, may neglect other attractive areas of the search space.

The idea of PDA to prevent this issue, is to slightly modify the search direction of all sub-problems with data perturbation [37, 3]. Data perturbation adds a random noise to a single-objective cost function related to a weighted sum problem (see Section 3.3). By adding such a noise in a cost function, we produce a stochastic change to the sub-problem search direction, which remains a single-objective version of the MOCO problem. Thus, ILS still can optimize such problem.

This leads to the second part of the initialization (**Step 1.2**) which consists of perturbing the cost function c^k of each sub-problem $\pi^k \in \Pi$ $k = 1, ..., K$. The data perturbation uses a parameter $d \geq 0$ chosen by the user, which controls the maximum variation of the random noise applied to a cost function.

The main loop of PDA (**Step 2**) corresponds to an IPLS. PDA maintains the starting set of PLS P_{pls}, initialized with the solutions of the archive A. It stops when a stopping criterion given by the user (maximum computational time, number of iterations,...) is met.

For each iteration of the main loop (**Steps 2.1-2.3**):

- First, we have to generate a new starting set of solutions for PLS (**Step 2.1**). To do so, we run ILS from the incumbent x^k of the sub-problem $\pi^k \in \Pi$ using the perturbed cost function c^k, for each $k = 1, ..., K$ (**Step 2.1.a**).

  Note that the `ILS` procedure used by PDA returns *all* the local optima found during the search, instead of the best one like previous works did [23, 11, 24, 35]. Indeed, ILS is capable of finding a number of potentially efficient solutions during its optimization process. **Step 2.1.b** adds these solutions to both the starting set of

PLS P_{pls} and the archive A, using the `UpdateSolutionSet` procedure; and updates the incumbents with the `UpdateIncumbents` procedure. Given a solution $x \in X$ and the set of sub-problems $\Pi = (\pi^1, ..., \pi^K)$, the `UpdateIncumbents` procedure replaces the incumbent x^k of π^k by x if ws $\left(x, \lambda^k\right) <$ ws $\left(x^k, \lambda^k\right)$ for all $k = 1, ..., K$.

Besides, the cumulative number r^k is updated with the number of potentially efficient solutions found by ILS, for each $k = 1, ..., K$.

- Because data perturbation is a stochastic procedure, it can produce single-objective problems whose optimal solutions are not efficient for the addressed MOCO problem, or very difficult to optimize with an ILS. To overcome this issue, we identify the sub-problem for which ILS is the least efficient for finding potentially efficient solutions, i.e. the sub-problem with the lowest r^k value, $k = 1, ..., K$; and perturb again its single-objective cost function (**Step 2.2**). We call this technique a *sub-problem reset*.

- Finally, at the end of an iteration, PDA conducts a PLS (**Step 2.3**, see Section 3.4 for more details) starting from the set P_{pls} provided in Step 2.1. P_{pls} is emptied in order not to run PLS from the same solutions at the next iteration.

3.2. Decomposition

The purpose of the decomposition (Algorithm 2) is to decompose a MOCO problem into a number of equally dispersed weighted sum problems and optimizes each once with ILS.

First, it generates the set of weights $\Lambda = (\lambda^1, ..., \lambda^K)$ such that $|\Lambda| = K = \binom{h+p-1}{h}$ following the MDS methodology (see Section 2.1) where h is a parameter fixed by the user. Let $c : E \longrightarrow \mathbb{R}^p$ be the MO cost function related to the addressed MOCO. For each weight $\lambda^k \in \Lambda$, we build the single-objective cost function $c^k : E \longrightarrow \mathbb{R}$ such that $c^k(e) = \sum_{j=1}^{p} \lambda_j^k c_j(e)$ for all $e \in E$. The function c^k defines the costs of the weighted-sum problem ws$(\cdot, \lambda^k)$.

Then, ILS optimizes the problem ws $\left(\cdot, \lambda^k\right)$ and returns a set of solutions P^k. The starting solution for ILS is generated by a domain-dependent heuristic. We put as incumbent of the new sub-problem $\pi^k = (\lambda^k, c^k, x^k, r^k)$, the solution x^k minimizing ws(x, λ^k) for all $x \in P^k$. The archive A is updated with the solutions of P^k and the attribute r^k is initialized to 0. Finally, sub-problem π^k is added to the set Π of sub-problems.

3.3. Data perturbation

Data perturbation (also called *noising method*) [37, 3, 38] has been introduced in MO optimization by Lust and Teghem [11]. In our article, data perturbation needs only one parameter, whereas the one of Lust and Teghem [11] needs three parameters.

The principle of data perturbation used in PDA is to add a random noise into the output of the single-objective cost function of a sub-problem. Given a sub-problem $\pi^k = (\lambda^k, c^k, x^k, r^k)$, the MO cost function c, and the data perturbation parameter $d \geq 0$ controlling the maximum variation of the noise; Algorithm 3 computes for each $e \in E$ its perturbed cost value $c^k(e) = \nu \times \sum_{j=1}^{p} \lambda_j^k c_j(e)$, where ν is a real number taken from a uniform distribution in the range $[1-d, 1+d]$. The higher d, the larger the perturbation is. Finally, attribute r^k counting the number of potentially efficient solutions found by ILS from π^k is reset, i.e. its value is set to 0.

Algorithm 1: Perturbed Decomposition Algorithm

Input : a *MOCO problem,* a *stopping criterion*, decomposition parameter h, data perturbation parameter d

Output: set A of potentially efficient solutions

Step 1) Initialization :

Step 1.1) Decomposition :

Let $c : E \longrightarrow \mathbb{R}^p$ be the MO cost function related to the *MOCO problem*

$\Pi \leftarrow$`Decomposition(`$c\downarrow$, $h\downarrow$, $A\updownarrow$`)` s.t. $\Pi = \left(\pi^1, ..., \pi^K\right)$

Let P_{pls} be the starting set of PLS

$P_{pls} \leftarrow A$

Step 1.2) Initial perturbations :

For each $\pi^k \in \Pi$:

`DataPerturbation(`$\pi^k\updownarrow$, $c\downarrow$, $d\downarrow$`)`

End of For

Step 2) Main loop :

While *stopping criterion* is not satisfied do :

Step 2.1) Search for a new solutions for PLS :

For each $\pi^k = (\lambda^k, c^k, x^k, r^k) \in \Pi$:

Step 2.1.a) Iterated Local Search :

$P \leftarrow$`ILS(`$c^k\downarrow$, $x^k\downarrow$`)`

Step 2.1.b) Upgrades :

For each $x \in P$:

`UpdateIncumbents(`$x\downarrow$, $\Pi\updownarrow$`)`

If `UpdateSolutionSet(`$x\downarrow$, A$\updownarrow$`)`

Then `UpdateSolutionSet(`$x\downarrow$, $P_{pls}\updownarrow$`)`

End of For

$r^k \leftarrow r^k + |P \cap A|$

End of For

Step 2.2) Reset the least efficient sub-problem :

$\pi^w \leftarrow \arg\min\{r^i : \pi^i = (\lambda^i, c^i, x^i, r^i) \in \Pi\}$

`DataPerturbation(`$\pi^w\updownarrow$, $c\downarrow$, $d\downarrow$`)`

Step 2.3) Pareto local search :

`PLS(`$P_{pls}\downarrow$, $A\updownarrow$, $\Pi\updownarrow$`)`

$P_{pls} \leftarrow \emptyset$

End of While

Algorithm 2: `Decomposition`

Input : multi-objective cost function c, decomposition parameter h, archive A

Output: set of sub-problems Π

$\Lambda \leftarrow$`MDS` $(h\downarrow)$

foreach $\lambda^k \in \Lambda$ **do**

  foreach $e \in E$ **do**

    $c^k(e) \leftarrow \sum_{j=1}^{p} \lambda_j^k c_j(e)$

  $P^k \leftarrow$`ILS` $(c^k\downarrow)$

  $x^k \leftarrow \min_{x \in P^k} \{\text{ws}(x, \lambda^k)\}$

  foreach $x \in P^k$ **do**

    `UpdateSolutionSet` $(x\downarrow, \text{A}\updownarrow)$

  $r^k \leftarrow 0$

  $\pi^k \leftarrow (\lambda^k, c^k, x^k, r^k)$

  $\Pi \leftarrow \Pi \cup \{\pi^k\}$

Algorithm 3: `DataPerturbation`

Input : sub-problem π^k, multi-objective cost function c, data perturbation parameter d

Output: $\emptyset$

foreach $e \in E$ **do**

  $\nu \leftarrow \mathcal{U}(1-d, 1+d)$

  $c^k(e) \leftarrow \nu \times \sum_{j=1}^{p} \lambda_j^k c_j(e)$

$r^k \leftarrow 0$

3.4. Pareto Local Search

Algorithm 4 (**Step 2.3**) presents an improved version of the original PLS1 [7]. Given a neighborhood function $\mathcal{N}_{pls}$, PLS explores the neighborhood of a starting set P of potentially efficient solutions. Each potentially efficient neighbor found (Line 4) is inserted into a temporary set P_{next} and serves to update sub-problem incumbents (Line 5).

When the neighborhood of all solutions of the starting set P has been explored, a new PLS is conducted on P_{next} (Line 10). PLS stops when P_{next} is empty. During the process, P is static and A can be updated with potentially efficient neighbors. Thus it may happen that the image of a newly generated neighbor y entering A dominates the image of a solution x in P. So x is proven to be non-efficient and we stop the exploration of its neighborhood. Two cases appear:

(i) Just before the exploration of the neighborhood of x, if x is no longer in A (this can be verified in $\mathcal{O}(1)$), it means that a point in A dominates the image of x. Then we skip the exploration of the neighborhood of x (Line 2).

(ii) We stop the exploration of the neighborhood of x if the image of a neighbor y dominates the image of x (Line 7).

Therefore, contrary to previous works [35, 11], we never explore the neighborhood of a solution proven to be non-efficient. This simple proposed speed-up technique limits the computational resources allocated to PLS and transfers them to ILS.

Algorithm 4: `PLS`

Input : starting set of solutions P, archive A, set of sub-problems $\Pi = \left(\pi^1, ..., \pi^K\right)$

Output: $\emptyset$

1 $P_{next} \leftarrow \emptyset$
2 **foreach** $x \in P \cap A$ **do**
3 **foreach** $y \in \mathcal{N}_{pls}(x) : f(x) \nleqq f(y)$ **do**
4 **if** `UpdateSolutionSet` ($y\downarrow$, A$\updownarrow$) **then**
5 `UpdateIncumbents` ($y\downarrow$, $\Pi\updownarrow$)
6 `UpdateSolutionSet` ($y\downarrow$, $P_{next}\updownarrow$)
7 **if** $f(y) \leq f(x)$ **then**
8 **break**
9 **if** $P_{next} \neq \emptyset$ **then**
10 `PLS` ($P_{next}\downarrow$, $A\updownarrow$, $\Pi\updownarrow$)

4. Computational experiments

Computational experiments are conducted on bi-objective and tri-objective instances of the MOTSP. We first recall the formal definition of the MOTSP, present the benchmark, introduce the quality indicators used to evaluate the quality of our results, specify the parameter settings, and analyze the sensitivity of PDA on the data perturbation. Then, we compare the results of PDA and MoMad on bi-objective instances; and the results of PDA, MoMad and PD-TPLS-l on tri-objective instances.

4.1. Definition of the MOTSP

In the single-objective version of the TSP, a traveling salesman has to visit a set of cities without passing more than once through each city and returns to the starting city. The goal is to find a tour such that the total cost is minimized. In MOTSP, the traveling salesman has to minimize several (potentially conflicting) costs. More formally, we define the MOTSP as follows. Given a complete graph $G = (V, E)$ with $V = \{v_1, ..., v_n\}$ the set of n nodes and $E := \{e_1, ...e_q\}$ corresponding to the set of edges such that $q = \frac{n(n-1)}{2}$, the MOTSP is defined by (1), where X represents the set of Hamiltonian cycles on G. We are interested here in the bi-objective and tri-objective TSP, that is $p = 2, 3$. The MOTSP is intractable and its decision version is $\mathcal{NP}$-hard, even if $p = 2$ [1].

4.2. Benchmark instances

The benchmark is composed of 46 instances of different types and sizes: 25 from the literature (19 bi-objective, 6 tri-objective), and 21 additional tri-objective ones we have generated with the same construction processes as in the literature. Four types of instances are considered:

- **Euclidean instances**: an instance is composed of *p* single-objective Euclidean instances. For each objective, the costs between the edges correspond to the Euclidean distance between two cities in a plane.

  In the *bi-objective case*, the ten following Euclidean instances are used: three instances published in [25]: euclidAB100, euclidAB300 and euclidAB500. Two instances generated on the basis of TSPLIB instances [39]: kroAB100, kroAB200. And finally five other instances generated in [11]: kroAB300, kroAB400, kroAB500, kroAB750, kroAB1000.

  In the *tri-objective case*, we have generated ten Euclidean instances of different sizes (30, 40, 50 and 200). The instances are: euclidA-3-30, euclidB-3-30, euclidC-3-30, euclidA-3-40, euclidB-3-40, euclidC-3-40, euclidA-3-50, euclidB-3-50, euclidC-3-50 and euclidG-3-200. Like the bi-objective "kro" instances presented above, the coordinates of each city are integers that are uniformly and independently generated in the range [0,3163]. Two additional instances of size 100 and 300 published in [25], are considered: euclidABC100 and euclidABC300.

- **Clustered instances**: an instance is composed of *p* single-objective clustered instances. For each objective, the cities are randomly clustered in a plane, and the costs between the edges correspond to the Euclidean distance.

  In the *bi-objective case*, we consider three clustered instances generated in [11]: ClusteredAB100, ClusteredAB300 and ClusteredAB500.

  In the *tri-objective case*, the three following clustered instances are used: two instances of sizes 100 and 300 provided by Lust[1] (ClusterAB100, ClusterAB300), and one instance of size 200 (clusterD-3-200), we have generated ourselves with the DIMACS TSP instance generator[2]. We do not consider clustered instances of smaller size because clusters of cities are not well defined when $n < 100$.

[1] http://www-desir.lip6.fr/~lustt/Research.html#Main

[2] http://dimacs.rutgers.edu/Challenges/TSP/

- **Random instances**: the costs between the edges are randomly generated from a uniform distribution.

In the *bi-objective case*, we use three random instances published in [25]: rdAB100, rdAB300 and rdAB500.

In the *tri-objective case*, we have generated ten random instances of different sizes (30, 40 50 and 200): rdA-3-30, rdB-3-30, rdC-3-30, rdA-3-40, rdB-3-40, rdC-3-40, rdA-3-50, rdB-3-50, rdC-3-50 and rdE-3-200. Like the random bi-objective instances presented above, each component of the cost vector assigned to an edge (between two cities) is chosen as an integer value taken from a uniform distribution in the range [0,4473]. Two additional instances of size 100 and 300 published in [25], are considered: rdABC100 and rdABC300.

- **Mixed instances** (only for the bi-objective case): the first cost corresponds to the Euclidean distance between two cities in a plane and the second cost is randomly generated from a uniform distribution. Three mixed instances also published in [25] are used: mixedAB100, mixedAB300 and mixedAB500.

4.3. Multi-objective quality indicators and statistical test

In single-objective optimization, it is quite easy to measure the quality of a solution. It is a more difficult task in the MO case, because MO outputs are represented by sets of trade-off solutions, potentially incomparable in term of Pareto dominance. Consequently, we use several indicators, called *quality indicators*, to measure the quality of an approximation of the Pareto front.

This section presents three of the most used quality indicators to compare approximation sets in MOCO: the *hypervolume difference indicator* I_{H^-} [40], the *ϵ-indicator* I_ϵ [41] and the *R2 indicator* I_{R2} [42].

The computation of these indicators implies to know the exact Pareto front Z_{nd}, which is generally unknown for a given instance. Thus we approximate it by merging all the approximations generated during the experimental phase and keeping only the non-dominated ones, forming the approximation of the Pareto front $\widetilde{Z}_{nd}$. Let $\widetilde{z}^* \in Z$ be the approximation of z^*, based on $\widetilde{Z}_{nd}$.

Hypervolume difference indicator $\mathbf{I_H^-}$ *[40] (to minimize).* Given an approximation set A and a reference point $\bar{z} \in Z$ which is weakly dominated by every point of A, the hypervolume value of A with regard to $\bar{z}$ measures the hypervolume of the region of the objective space which is weakly dominated by A and weakly dominates $\bar{z}$. More formally, the hypervolume indicator I_H is such that $I_H(A, \bar{z}) = \int_Z dom(A, \bar{z})dz$ where $dom(A, \bar{z}) = \{z \in Z : \exists z' \in A : z' \leqq z \leqq \bar{z}\}$.

In the present work we use the hypervolume difference indicator I_H^-. Given an approximation set A and the reference point $\bar{z}$, the indicator value is defined as:

$$I_H^-(A, \widetilde{Z}_{nd}, \bar{z}) = I_H(\widetilde{Z}_{nd}, \bar{z}) - I_H(A, \bar{z})$$

$I_H^-(A, \widetilde{Z}_{nd}, \bar{z})$ defines the hypervolume of the subspace that is weakly dominated by $\widetilde{Z}_{nd}$ but not by A. In contrast to the original hypervolume indicator, the lower $I_H^-(A)$, the better the quality of A is. We use the algorithm of Fonseca and al. [43, 44] to compute the hypervolume. Source code is available online [3].

[3] http://iridia.ulb.ac.be/∼manuel/hypervolume

Before using I_H^-, a normalization is necessary in order to allow the different objectives to contribute equally to indicator value. A standard linear normalization procedure will apply the following transformation:

$$z_j^{'} \leftarrow \frac{z_j - z_j^{min}}{z_j^{max} - z_j^{min}} + 1 \quad \text{for } j = 1, ..., p \tag{5}$$

where $z^{min} = (z_1^{min}, ..., z_p^{min}) \in Z$ and $z^{max} = (z_1^{max}, ..., z_p^{max}) \in Z$, such that z_j^{min} and z_j^{max} are respectively the estimated minimum and maximum values that the j^{th} objective can take, for each $j = 1, ..., p$. The computation of z_j^{min} and z_j^{max} is based on the values of the points of all provided approximation sets. Note that without the $+1$ in (5), extreme points will not contribute to the hypervolume value. After normalization, the coordinates of points fall in the range $[1, 2]$.

As advised by Fonseca et al. [44, 43] in their hypervolume computation algorithm, we use $\bar{z} = z^{max} + 0.1 \times (z^{max} - z^{min})$ as the reference point for computing the hypervolume. After the normalization step, $\bar{z} = (2.1, 2.1)$.

ϵ indicator $\mathbf{I}_\epsilon$ [41] (to minimize). Given an approximation set A and the approximation of the Pareto front $\widetilde{Z}_{nd}$, the unary ϵ indicator I_ϵ gives the factor by which A is worse than $\widetilde{Z}_{nd}$ with respect to all objectives, defined as:

$$I_\epsilon(A, \widetilde{Z}_{nd}) = \inf_{\epsilon \in \mathbb{R}} \{\forall z' \in \widetilde{Z}_{nd}, \exists z \in A : z_j \leq (1 + \epsilon) z'_j\}$$

The lower $I_\epsilon(A, \widetilde{Z}_{nd})$, the better the approximation set A is comparing to $\widetilde{Z}_{nd}$.

R2 indicator $\mathbf{I_{R2}}$ [42] (to minimize). Given an approximation set A and a set of weights Λ, the unary R2 indicator I_{R2} value of A is defined as:

$$I_{R2}(A, \Lambda, \widetilde{Z}_{nd}, \widetilde{z}^*) = \frac{\sum_{\lambda \in \Lambda} \left(\min_{x \in A} \text{wat}(x, \lambda, \widetilde{z}^*) - \min_{x \in \widetilde{Z}_{nd}} \text{wat}(x, \lambda, \widetilde{z}^*) \right)}{|\Lambda|}$$

The lower $I_{R2}(A, \Lambda, \widetilde{Z}_{nd}, \widetilde{z}^*)$, the better the approximation set A is comparing to $\widetilde{Z}_{nd}$. As indicated in [45], the set Λ is made using the MDS method (see Section 2 for details) with a parameter h which should be sufficiently large to cover well $\widetilde{Z}_{nd}$. In order to have a number of weights proportional to the size of $\widetilde{Z}_{nd}$, we set $h := \arg\inf\{\binom{h'+p-1}{h'} \geq \frac{1}{10}|\widetilde{Z}_{nd}| : h' \in \mathbb{N}\}$. As suggested by Fonseca et al. [45], normalization is not mandatory for this indicator.

Note that for all indicators used in the present work, the value of the approximation of the non-dominated set $\widetilde{Z}_{nd}$ is 0.

Mann-Whitney statistical test. In order to statistically compare the results of the different algorithms, the Mann-Whitney non-parametric statistical test [46] has been applied. For a specific indicator on a given instance, this test assesses whether two algorithms are comparable. If the Mann-Whitney test is satisfied, it means there is no statistical difference between the values of the quality indicator obtained by the two algorithms. Otherwise mean values are simply compared.

As three hypotheses are tested simultaneously (one for each indicator, given an instance), the levels of risk of the tests have been adjusted with the Holm sequential rejective method (see [47] for more details). The starting level of risk of the Mann-Whitney test has been fixed to 1%.

4.4. Parameter settings of MoMad and PD-TPLS-l

The MoMad parameters have been fixed as follows, as indicated in [35]:

- During the decomposition phase, MoMad uses the MDS method (see Section 2.1) to generate its set of weights. For bi-objective instances, the number of sub-problems is set to $\min(n, 600)$. MoMad has not been tested for tri-objective instances, so we have experimentally fixed the number of sub-problems to $\binom{h+3-1}{h}$ where $h = \min(n, 60)$ is the parameter of the MDS method, for all tri-objective instances. So for instances of size 30, 40, 50, the respective number of sub-problems is 496, 861, 1326. For larger instances, the number of sub-problems is 1891.

  For each generated weight, MoMad optimizes the corresponding weighted sum problem by running an improved version of the Lin-Kernighan (LK) heuristic [48]: the chained LK of Applegate et al [49]. It uses a variable *k*-edge-neighborhood and is one of the best ILS for the single-objective TSP. The source code of the chained LK is available trough the Concorde package[4].

- The maximum number of iterations for PLS is set to 10.

- The number of iterations is fixed to 500 for all bi-objective instances and for tri-objective instances of size $n \leq 50$. For larger-sized tri-objective instances ($n \geq 100$), the number of iterations is fixed to 1000.

PD-TPLS-l also uses MDS to generate its set of weights, and we set the number of weights to $\binom{150+3-1}{150} = 11,476$ for all instances, in order to avoid the clustering effect described in [23] and briefly discussed in Section 4.7.2. For each generated weight, PD-TPLS-l optimizes the corresponding weighted sum problem by calling the chained LK as suggested in [23], instead of the 3-opt first improvement used in the original method. Internal tests have shown that PD-TPLS-l using the chained LK gives better results.

4.5. Parameter settings of PDA

For the decomposition phase, we choose as ILS the chained LK. We use the default parameters of the chained LK, but modify the algorithm to integrate it into our own implementation and to memorize all the generated local optima with distinct *fitness* values, where the *fitness function* is the function optimized by the chained LK.

The neighborhood function of PLS $\mathcal{N}_{pls}$ chosen is the *2-edge-exchange* neighborhood, as suggested in [50]. A candidate edge list is associated to $\mathcal{N}_{pls}$: it consists of all edges composing at least one solution of the set of potentially efficient solutions A.

During the main loop of PDA, we choose as ILS a 3-opt first improvement with biased random double-bridge kicks and same candidate edge list as in PLS. We use the implementation of Paquete[5].

To have approximately the same amount of computational resources between the compared methods, and thus making a comparison as fair as possible:

- The number K of sub-problems in PDA is the same as in MoMad.

- The stopping criterion used by PDA for a given instance corresponds to the minimum between the average computational time spent by MoMad and the one of PD-TPLS-l on this instance.

The value of the data perturbation parameter d is fixed in the next subsection.

[4] http://www.tsp.gatech.edu/concorde

[5] http://www.sls-book.net/implementations.html

4.5.1. Sensitivity analysis of PDA on the data perturbation

PDA introduces data perturbation to the framework of Decomposition algorithms [2]. The aim of this section is to analyze the influence of data perturbation on PDA's results. To this aim, we have generated 12 new Euclidean and random, bi-objective and tri-objective TSP test instances of different sizes (see Section 4.2 for instance generation):

- In the *bi-objective case*:
  - 3 Euclidean instances: euclidT-2-100, euclidT-2-300, euclidT-2-500;
  - 3 random instances: rdT-2-100, rdT-2-300, rdT-2-500.
- In the *tri-objective case*:
  - 3 Euclidean instances: euclidT-3-30, euclidT-3-50, euclidT-3-100;
  - 3 random instances: rdT-3-30, rdT-3-50, rdT-3-100.

For each instance, we compare 5 different alternative values of the data perturbation parameter d: 0% (no data perturbation), 2.5%, 5%, 7.5% and 10%. For a given instance, PDA has been run 10 times for each value of d. The average I_H^- values related to each alternative have been compared in function of the running time, controlled by the number of iterations varying from 0 to 1500. I_H^- values have been computed every 300 iterations.

Results are shown in Figure 1 for bi-objective instances and Figure 2 for tri-objective instances. Each curve corresponds to a unique value of data perturbation. First, one can note that the running time of all alternatives are quite similar. We remark that the increase of the number of iterations brings a significant improvement of the quality indicator value.

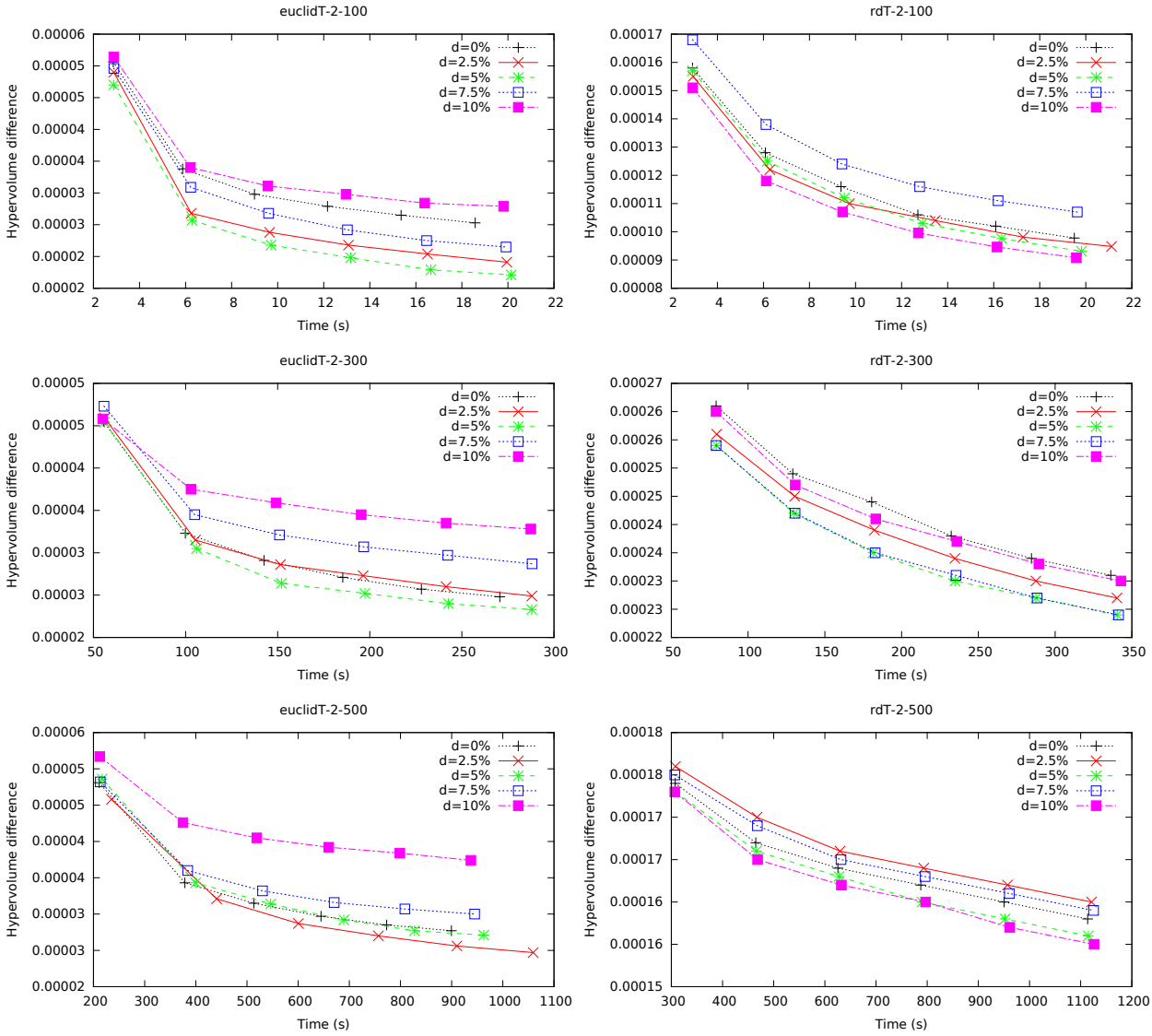

Figure 1: Influence of the data perturbation d on I_H^- (to min.) in function of the running time, for bi-objective Euclidean (left) and random (right) instances.

In the bi-objective case, it appears that PDA with small values of d performs better on Euclidean instances, particularly for $d = 2.5\%, 5\%$ (the red and green curves). When the size of the instance increases, the differences of performance without data perturbation ($d = 0\%$) and with the best values of perturbation ($d = 2.5\%, 5\%$) decreases. However, for euclid-T-500, $d = 2.5\%$ is still a bit better than $d = 0\%$. For random instances, larger values of d seem to be more efficient, and differences of performances with and without data perturbation are more pronounced. PDA with $d = 5\%$ obtains good and stable results on the three random instances.

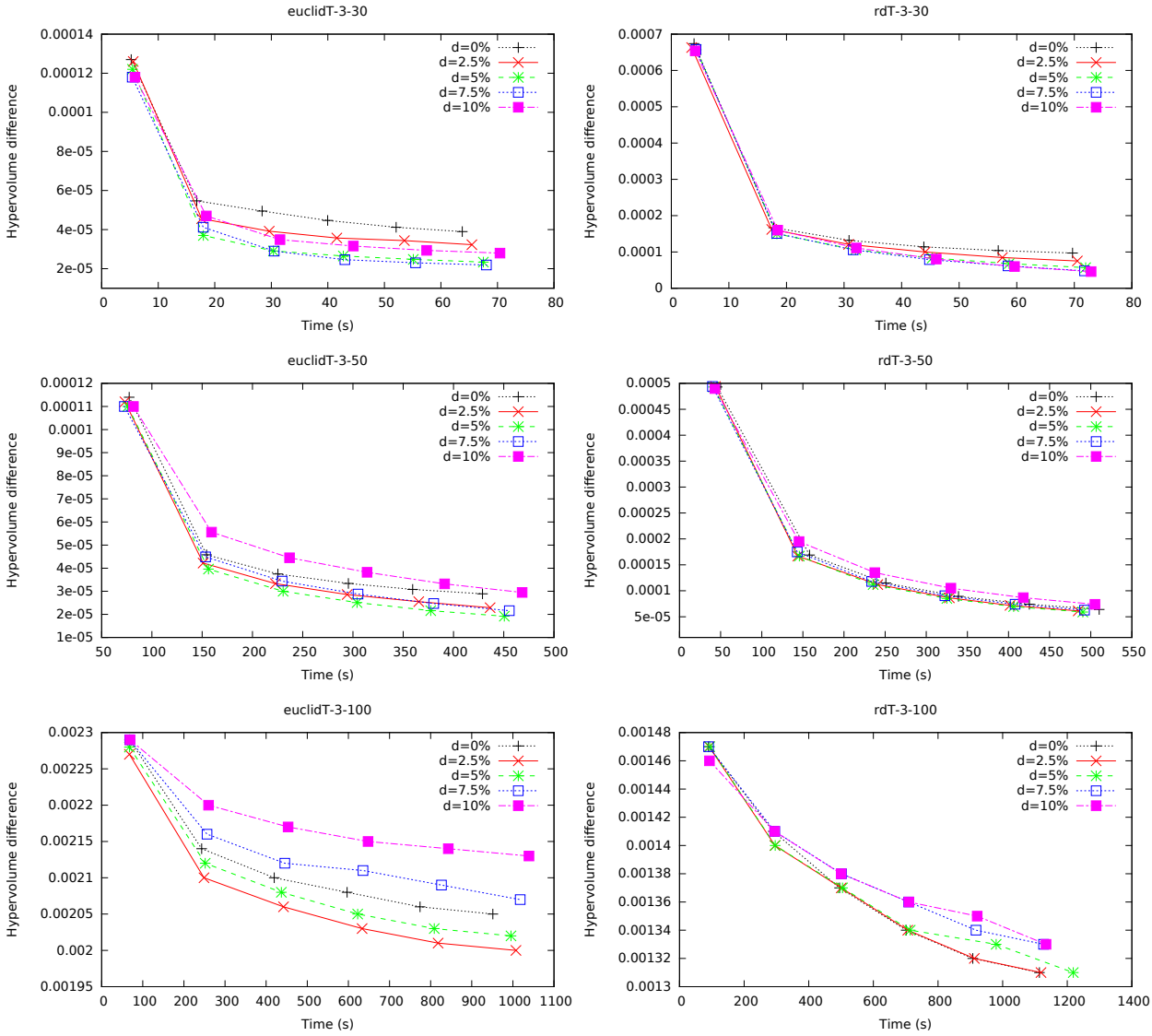

Figure 2: Influence of the data perturbation d on I_H^- (to min.) in function of the running time, for tri-objective Euclidean (left) and random (right) instances.

In the tri-objective case, large values of d perform better for instances of size less than 50 for both Euclidean and random instances. When $n \geq 50$, smaller values of d are more efficient for both types of instances, while $d = 2.5\%, 5\%$ bring better results than $d = 0\%$ for Euclidean instances. For random instances of size $n \geq 50$, $d = 2.5\%, 5\%$ give similar results than $d = 0\%$.

We can notice that the efficiency of data perturbation is strongly dependent on the instance characteristics (size, number of objectives, cost functions). Globally, data perturbation has a positive impact on the optimization process over the tested instances and meets our expectations, described in Section 3.1. Indeed, in a great majority of instances, when PLS is stuck in a locally efficient set, ILS is more efficient in proposing new starting solutions for PLS, by optimizing perturbed weighted sum problems instead of non perturbed ones. In fact, searching for optima of perturbed weighted sum problems enables the exploration of regions of the search space which were neglected without data perturbation, and thus escape more easily from locally efficient set.

To summarize this sensitivity analysis, small but strictly positive values of data perturbation are recommended for MOTSP, whereas greater values seem to lead to unstable results from an instance to another.

4.5.2. Final parameter settings

The value of the data perturbation parameter d has been fixed to 5% for both bi-objective and tri-objective instances. Indeed, choosing $d = 5\%$ represents a good compromise between Euclidean and random instances in the bi-objective case, and performs well in the tri-objective case when the size of the instance increases. Table 1 reports the chosen parameter values for PDA.

Parameter	Value
Number of sub-problems K	Same values as MoMad
ILS - decomposition phase	Chained LK
ILS - iterative phase	3-opt first improvement
Data perturbation d	5%
PLS's neighborhood function $\mathcal{N}_{pls}$	2-edge-exchange
Stopping criterion	Min(MoMad time, PD-TPLS-l time)

Table 1: Final parameter settings of PDA.

4.6. Experimental design

For all bi-objective instances and small-sized ($n \leq 50$) tri-objective instances, the sets of points of the three algorithms (PDA, MoMad and PD-TPLS-l) are managed as regular unbounded archives.

For tri-objective instances with $n \geq 100$, the sets of points of the three algorithms are managed by the well-known ϵ-*archive* concept introduced by Laumanns et al. [51]. They propose to place a hyper-grid that discretizes the objective space into regions called *boxes*. A box can contain at most one point. Given a tolerance $\epsilon > 0$, the principle of the ϵ-archive is to maintain a set of well-distributed points in the objective space, and to bound the size of this set. The tolerance parameter ϵ controls the dispersion of the points and implicitly determines the maximal size of the approximation. The larger ϵ is, the larger the dispersion of the points is and the smaller the size of the approximation is. See [51] for a more detailed description of an ϵ-archive, the guarantees on the good distribution of points in the ϵ-archive and the guarantees on the bound on its size. For all concerned instances, ϵ is fixed to 1%.

The three algorithms have been run 20 times on the instances presented in Section 4.2 and compared using the I_ϵ, I_H^- and I_{R2} indicators. All experiments presented were performed on a 3.4 GHz computer with 16Gb RAM on a Linux OS. All algorithms are written in C/C++.

4.7. Experimental results

4.7.1. Results on bi-objective instances

Tables 2 and 3 compare PDA and MoMad, by reporting for the 19 bi-objective TSP benchmark instances the average values and standard deviation of I_ϵ, I_H^- and I_{R2}, the average size $|A|$ of the approximation sets returned for each algorithm, the size $|\widetilde{Z}_{nd}|$ of the approximation of the non-dominated set, the number of examined solutions, and the average time spent by the two algorithms. Numbers in bold indicate a better average value.

PDA has better average values on I_ϵ, I_H^- and I_{R2} than MoMad on all tested bi-objective instances. Furthermore, the Mann-Whitney test indicates that PDA is better than MoMad for each instance and quality indicator used. While

the computational time of PDA and MoMad are similar, the number of examined solutions is generally larger for PDA than for MoMad.

The I_ϵ, I_H^- and I_{R2} values of the approximation sets produced by the two algorithms are plotted in Figure 3 for three Euclidean "kro" instances, and in Figure 4 for three random instances.

Instance	Algorithm	I_ϵ (10^{-3})	I_H^- (10^{-4})	I_{R2}	$\|A\|$	$\|\widetilde{Z}_{nd}\|$	Nb. of exam. sol. (10^6)	Time (s)
ClusterAB100	MoMad	4.06 $\pm$ 0.50	1.60 $\pm$ 0.09	5.46 $\pm$ 0.65	2,454	3,036	2.85	14
	PDA	**1.64** $\pm$ 0.28	**0.20** $\pm$ 0.04	**0.72** $\pm$ 0.15	**2,762**		2.36	14
ClusterAB300	MoMad	6.19 $\pm$ 0.48	3.42 $\pm$ 0.30	21.0 $\pm$ 0.95	15,931	21,616	113	209
	PDA	**1.99** $\pm$ 0.54	**0.39** $\pm$ 0.06	**3.62** $\pm$ 0.64	**18,488**		203	209
ClusterAB500	MoMad	7.05 $\pm$ 1.02	0.86 $\pm$ 0.03	15.6 $\pm$ 0.26	40,936	54,239	440	677
	PDA	**1.30** $\pm$ 0.47	**0.22** $\pm$ 0.04	**4.94** $\pm$ 0.73	**45,706**		577	678
euclidAB100	MoMad	3.64 $\pm$ 0.40	2.01 $\pm$ 0.14	8.56 $\pm$ 0.76	1,400	1,812	1.46	10
	PDA	**2.05** $\pm$ 0.04	**0.30** $\pm$ 0.03	**1.23** $\pm$ 0.18	**1,612**		1.29	10
euclidAB300	MoMad	1.28 $\pm$ <0.01	0.72 $\pm$ 0.01	10.4 $\pm$ 0.20	14,436	18,519	62.9	164
	PDA	**0.83** $\pm$ 0.13	**0.20** $\pm$ 0.02	**3.31** $\pm$ 0.35	**16,334**		66.7	164
euclidAB500	MoMad	1.17 $\pm$ 0.03	0.78 $\pm$ <0.01	19.3 $\pm$ 0.23	34,148	44,878	338	605
	PDA	**0.90** $\pm$ 0.11	**0.38** $\pm$ 0.04	**9.82** $\pm$ 0.73	**37,984**		392	606
kroAB100	MoMad	3.94 $\pm$ 0.29	1.74 $\pm$ 0.12	9.79 $\pm$ 0.66	2,569	3,332	2.78	11
	PDA	**2.03** $\pm$ 0.50	**0.21** $\pm$ 0.02	**1.15** $\pm$ 0.13	**3,015**		2.44	11
kroAB200	MoMad	3.54 $\pm$ 0.03	0.98 $\pm$ 0.02	9.70 $\pm$ 0.30	6,649	8,913	16.8	59
	PDA	**1.17** $\pm$ 0.20	**0.15** $\pm$ 0.02	**1.51** $\pm$ 0.20	**7,880**		15.8	59
kroAB300	MoMad	1.82 $\pm$ 0.15	0.63 $\pm$ 0.01	7.48 $\pm$ 0.14	14,880	19,027	69.1	169
	PDA	**0.87** $\pm$ 0.13	**0.15** $\pm$ 0.02	**2.04** $\pm$ 0.25	**16,963**		69.6	169
kroAB400	MoMad	2.07 $\pm$ <0.01	0.82 $\pm$ 0.01	12.0 $\pm$ 0.17	21,793	30,388	152	346
	PDA	**1.11** $\pm$ 0.225	**0.27** $\pm$ 0.04	**5.09** $\pm$ 0.42	**25,027**		178	346
kroAB500	MoMad	2.18 $\pm$ <0.01	0.84 $\pm$ <0.01	18.5 $\pm$ 0.19	33,436	46,095	339	632
	PDA	**1.07** $\pm$ 0.21	**0.39** $\pm$ 0.03	**11.0** $\pm$ 0.75	**37,863**		430	632
kroAB750	MoMad	1.64 $\pm$ 0.92	0.86 $\pm$ <0.01	30.2 $\pm$ 0.24	60,415	84,042	1,151	1,417
	PDA	**1.17** $\pm$ 0.28	**0.55** $\pm$ 0.04	**21.0** $\pm$ 1.06	**66,365**		1,462	1,417
kroAB1000	MoMad	2.61 $\pm$ <0.01	1.05 $\pm$ <0.01	44.6 $\pm$ 0.21	98,476	127,073	2,663	2,465
	PDA	**1.63** $\pm$ 0.40	**0.78** $\pm$ 0.05	**34.8** $\pm$ 1.55	**105,174**		3,264	2,466

Table 2: Comparison between PDA and MoMad results on Euclidean and clustered bi-objective instances.

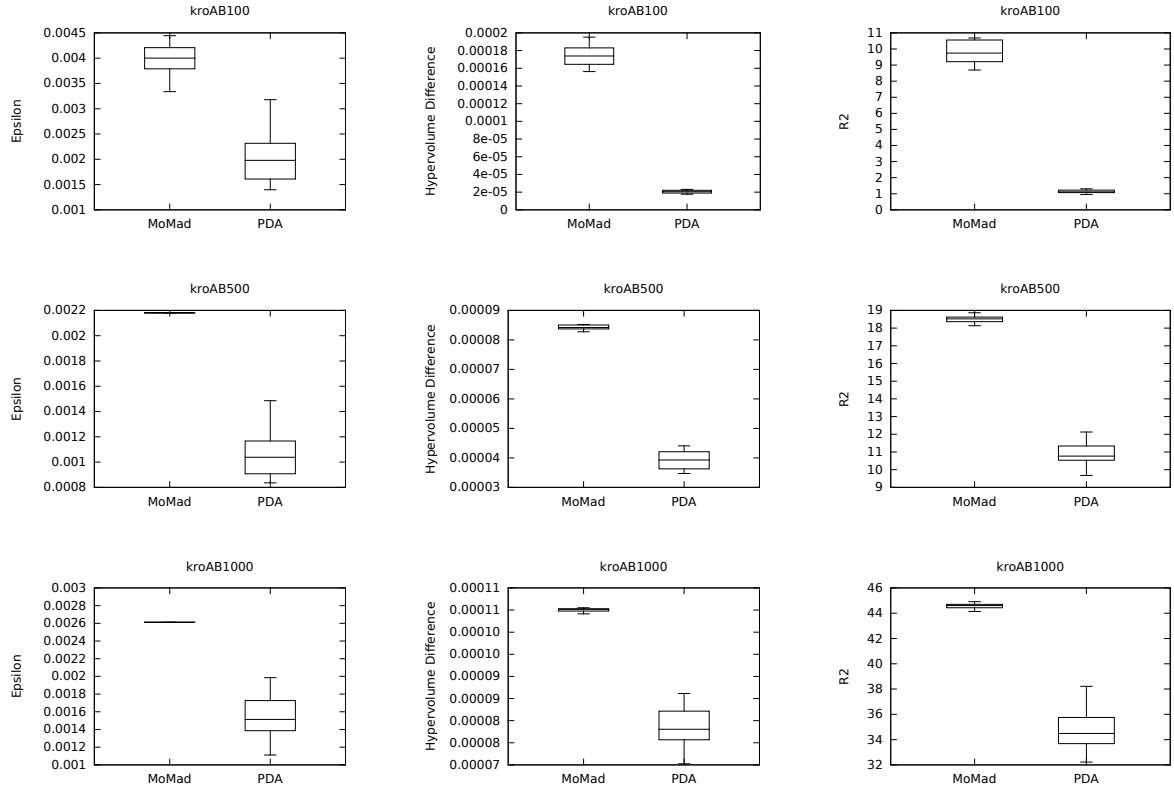

Figure 3: I_ϵ (left), I_H^- (middle) and I_{R2} (right) comparison between MoMad and PDA on kroAB100 (1st line), kroAB500 (2nd line), and kroAB1000 (3rd line) instances

Instance	Algorithm	I_ϵ (10^{-3})	I_H^- (10^{-4})	I_{R2}	$\lvert A\rvert$	$\lvert\widetilde{Z}_{nd}\rvert$	Nb. of exam. sol. (10^6)	Time (s)
mixedAB100	MoMad	15.2 $\pm$ 1.29	5.14 $\pm$ 0.19	21.8 $\pm$ 1.51	983	1,846	1.20	11
	PDA	**4.64** $\pm$ 0.68	**0.64** $\pm$ 0.06	**1.38** $\pm$ 0.27	**1,461**		1.23	11
mixedAB300	MoMad	25.0 $\pm$ 0.39	2.56 $\pm$ 0.02	37.5 $\pm$ 0.51	5,786	12,093	28.0	185
	PDA	**9.93** $\pm$ 1.06	**1.29** $\pm$ 0.06	**16.7** $\pm$ 0.98	**7,477**		32.6	185
mixedAB500	MoMad	14.9 $\pm$ 0.27	1.97 $\pm$ <0.01	55.8 $\pm$ 0.38	14,008	26,028	145	721
	PDA	**11.6** $\pm$ 1.18	**1.42** $\pm$ 0.05	**38.4** $\pm$ 1.25	**16,692**		182	721
rdAB100	MoMad	22.0 $\pm$ 3.14	8.62 $\pm$ 0.36	44.3 $\pm$ 1.89	641	1,707	0.80	12
	PDA	**6.34** $\pm$ 1.03	**1.14** $\pm$ 0.12	**4.90** $\pm$ 0.60	**1,161**		1.04	12
rdAB300	MoMad	34.5 $\pm$ 1.00	5.52 $\pm$ 0.04	82.6 $\pm$ 0.66	2,079	8,617	9.37	205
	PDA	**13.0** $\pm$ 1.12	**3.18** $\pm$ 0.10	**50.5** $\pm$ 1.56	**3,231**		12.7	205
rdAB500	MoMad	29.0 $\pm$ 0.97	4.02 $\pm$ 0.01	13.3 $\pm$ 0.60	3,518	11,188	29.9	750
	PDA	**18.0** $\pm$ 1.95	**2.82** $\pm$ 0.06	**10.0** $\pm$ 1.80	**4,822**		38.4	750

Table 3: Comparison between PDA and MoMad results on random and mixed bi-objective instances.

Standard deviation of indicator values is globally larger for PDA compared to MoMad. The variations of indicator values for our method on a given instance, is due to the use of data perturbation which induces an additional level of stochasticity (in addition to the use of Chained LK and the 3-opt first improvement ILS) compared with MoMad, which is a Decomposition algorithm without data perturbation.

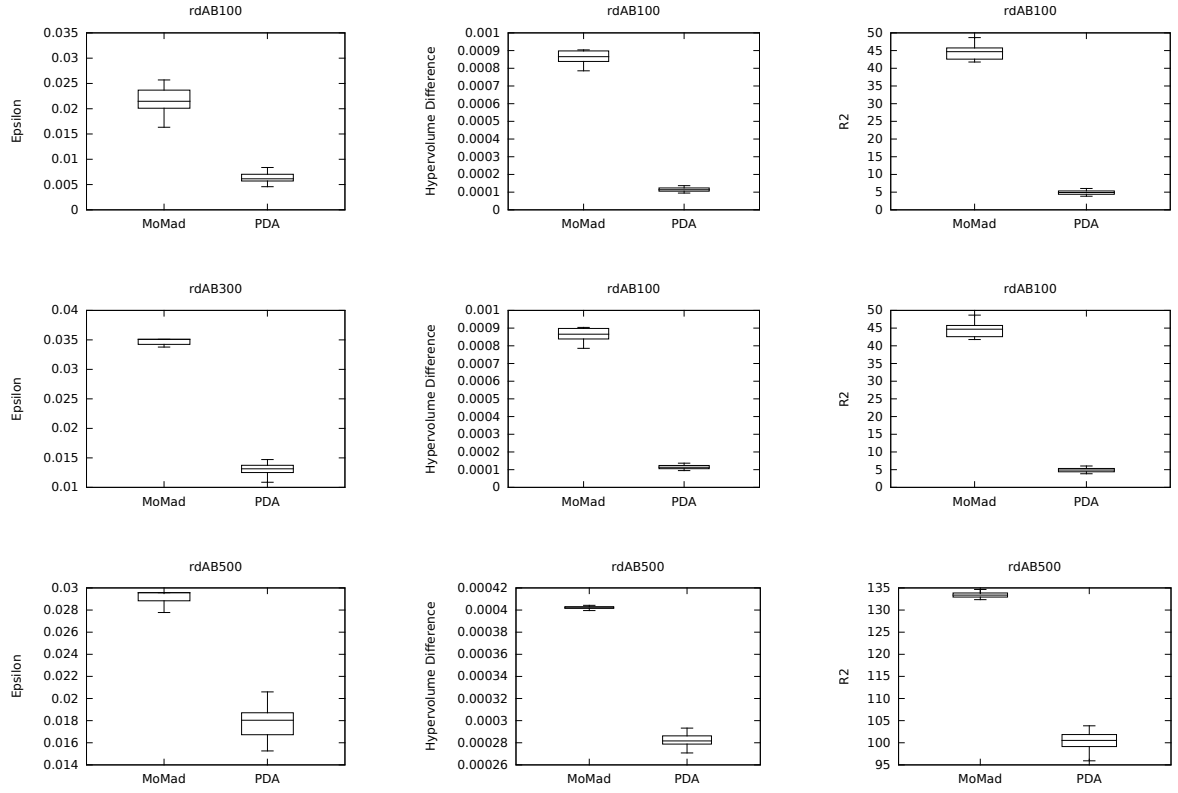

Figure 4: I_ϵ (left), I_H^- (middle) and I_{R2} (right) comparison between MoMad and PDA on rdAB100(1st line), rdAB300(2nd line) and rdAB500(3rd line) instances.

However, it is important to note that for all I_H^-, I_ϵ and I_{R2} collected values, the best value found by MoMad never reaches the worst one found by our method for all the tested instances, except for I_ϵ on euclidAB750.

Finally, we can observe that PDA has better performance on random, mixed and clustered instances than on Euclidean "kro" and "euclid" instances. We will discuss this point in Section 4.7.3.

4.7.2. Results on tri-objective instances

Tables 4 and 5 compare PDA, MoMad and PD-TPLS-l results on the 27 tri-objective TSP benchmark instances.

PDA obtains better results than MoMad and PD-TPLS-l on all tested instances. As for the bi-objective case, the Mann-Whitney test has been applied for tri-objective instances and states that PDA is better than both MoMad and PD-TPLS-l, for each instance and quality indicator used.

The I_ϵ, I_H^-, and I_{R2} values of the approximation sets produced by PDA, MoMad and PD-TPLS-l are plotted in Figure 5 for three Euclidean instances, and in Figure 6 for three random instances.

Instance	Algorithm	I_ϵ (10^{-2})	I_H^- (10^{-4})	I_{R2}	$\|A\|$	$\|\widetilde{Z}_{nd}\|$	Nb. of exam. sol. (10^6)	Time (s)
ClusterABC100	MoMad	1.47 ± 0.08	18.5 ± 0.13	36.3 ± 0.34	15,933	80,360	78	905
	PDA	**1.20** ± 0.05	**12.8** ± 0.09	**25.8** ± 0.24	**16,343**		43	905
	PD-TPLS-l	2.34 ± 0.11	17.1 ± 0.09	38.5 ± 0.37	14,088		2,366	1,439
clusterD-3-200	MoMad	2.04 ± 0.02	27.2 ± 0.07	85.6 ± 0.40	24,170	251,917	322	2,041
	PDA	**1.20** ± 0.03	**14.6** ± 0.06	**50.2** ± 0.32	**25,387**		163	2,041
	PD-TPLS-l	2.59 ± 0.38	18.4 ± 0.14	61.7 ± 0.49	22,677		5,066	2,228
ClusterABC300	MoMad	3.76 ± 0.27	36.5 ± 0.15	172 ± 0.70	28,064	342,642	730	3,614
	PDA	**2.07** ± 0.08	**16.6** ± 0.27	**90.1** ± 0.58	**30,791**		367	3,614
	PD-TPLS-l	3.11 ± 0.11	19.9 ± 0.38	106 ± 0.63	26,297		7,788	3,633
euclidA-3-30	MoMad	0.99 ± 0.09	0.47 ± 0.05	0.37 ± 0.04	12,111	12,820	4.81	22
	PDA	**0.86** ± 0.03	**0.30** ± 0.03	**0.16** ± 0.05	**12,197**		4.32	22
	PD-TPLS-l	2.84 ± 0.03	26.8 ± 0.35	15.2 ± 0.15	5,060		251	178
euclidB-3-30	MoMad	0.86 ± 0.05	0.52 ± 0.07	0.45 ± 0.10	10,695	11,426	4.38	22
	PDA	**0.85** ± 0.05	**0.31** ± 0.06	**0.21** ± 0.06	**10,819**		3.91	22
	PD-TPLS-l	2.34 ± 0	26.3 ± 0.17	16.6 ± 0.25	4,666		256	179
euclidC-3-30	MoMad	0.82 ± 0.01	0.36 ± 0.05	0.34 ± 0.10	15,748	16,437	5.91	23
	PDA	**0.78** ± 0.14	**0.28** ± 0.06	**0.17** ± 0.04	**15,863**		5.26	23
	PD-TPLS-l	2.18 ± 0	26.0 ± 0.11	15.1 ± 0.05	5,627		244	168
euclidA-3-40	MoMad	1.07 ± 0.03	1.49 ± 0.09	1.61 ± 0.15	39,636	46,413	26.2	91
	PDA	**0.79** ± 0.07	**0.43** ± 0.04	**0.41** ± 0.08	**42,061**		23.8	91
	PD-TPLS-l	2.03 ± 0	21.3 ± 0.12	16.6 ± 0.13	14,234		619	408
euclidB-3-40	MoMad	0.92 ± 0.09	0.59 ± 0.07	0.66 ± 0.13	34,919	37,206	21.4	85
	PDA	**0.65** ± 0.09	**0.16** ± 0.02	**0.13** ± 0.01	**35,623**		18.8	85
	PD-TPLS-l	2.54 ± 0	21.5 ± 0.15	17.4 ± 0.17	12,352		628	406
euclidC-3-40	MoMad	0.94 ± 0.04	1.17 ± 0.04	1.06 ± 0.08	32,189	36,694	21.6	81
	PDA	**0.72** ± 0.06	**0.31** ± 0.02	**0.25** ± 0.03	**33,946**		19.2	81
	PD-TPLS-l	2.18 ± 0	21.3 ± 0.18	16.6 ± 0.13	11,942		615	380
euclidA-3-50	MoMad	0.79 ± 0.08	1.10 ± 0.07	1.37 ± 0.12	91,192	102,970	83.2	224
	PDA	**0.55** ± 0.04	**0.20** ± 0.02	**0.17** ± 0.03	**96,292**		72.5	224
	PD-TPLS-l	1.80 ± 0.03	20.1 ± 0.16	21.7 ± 0.20	25,290		1,058	738
euclidB-3-50	MoMad	0.69 ± 0.04	1.58 ± 0.03	2.16 ± 0.10	106,154	125,686	98.8	228
	PDA	**0.50** ± 0.01	**0.37** ± 0.02	**0.40** ± 0.03	**113,181**		86.8	228
	PD-TPLS-l	1.97 ± 0.08	20.8 ± 0.20	20.1 ± 0.39	27,493		1,047	751
euclidC-3-50	MoMad	0.64 ± 0.03	1.22 ± 0.05	1.43 ± 0.10	109,375	126,855	107	258
	PDA	**0.50** ± 0.04	**0.27** ± 0.02	**0.29** ± 0.02	**116,378**		88.5	258
	PD-TPLS-l	2.13 ± 0.13	18.4 ± 0.11	19.2 ± 0.24	31,783		1,023	721
euclidABC100	MoMad	1.33 ± 0.04	22.2 ± 0.15	41.5 ± 0.48	12,196	59,811	52.7	784
	PDA	**1.14** ± 0.03	**15.8** ± 0.14	**30.4** ± 0.45	**12,687**		29.9	784
	PD-TPLS-l	2.63 ± 0.24	22.2 ± 0.18	41.5 ± 0.49	11,124		2,455	982
euclidG-3-200	MoMad	1.81 ± 0.11	30.7 ± 0.12	113 ± 0.73	19,720	225,617	231	1,859
	PDA	**1.16** ± 0.04	**17.7** ± 0.06	**69.5** ± 0.68	**20,800**		121	1,859
	PD-TPLS-l	2.23 ± 0.13	21.7 ± 0.13	82.0 ± 0.98	18,923		4,927	2,005
euclidABC300	MoMad	2.13 ± 0.05	33.3 ± 0.09	175 ± 0.87	18,451	252,373	399	3,092
	PDA	**1.20** ± 0.04	**16.6** ± 0.13	**100** ± 1.44	**20,347**		197	2,668
	PD-TPLS-l	2.28 ± 0.18	19.9 ± 0.15	114 ± 1.26	18,146		7,052	2,667

Table 4: Comparison between PDA, MoMad and PD-TPLS-l results on clustered and Euclidean tri-objective instances.

One can remark that PD-TPLS-l is outperformed by both PDA and MoMad for instances of size $n \leq 50$. In spite of a number of weighted sum optimizations much larger than PDA and MoMad, PD-TPLS-l stagnates and never reaches the performances of its competitors on these instances. Besides, the computational time of PD-TPLS-l explodes with the size of the instance. These two points can be explained: first, PD-TPLS-l conducts only one iteration of PLS, contrary to PDA and MoMad which continue PLS after the first iteration, so the final quality of the latter will be better. Second, the neighborhood of the PLS used by PD-TPLS-l is a 3-edge-exchange, much larger than the 2-edge-exchange used by PDA and MoMad and thus consuming more computational resources when n increases. This is confirmed by the number of solutions examined by PD-TPLS-l, which is at least 10 times larger than the number of solutions examined by PDA or MoMad. Finally, PD-TPLS-l uses a big-sized candidate list containing useless candidates (see [19] for technical details about the candidate list used), much less efficient than the one used by PDA. These remarks point out the difficulty of designing an efficient PLS using a neighborhood larger than the 2-exchange in the MOTSP.

On the other hand, for instances of size $n \geq 100$, PD-TPLS-l performs similarly or better than MoMad, but is still outperformed by PDA. In fact, MoMad is impacted by the clustering effect described in [23]. MoMad solves a limited number of weighted sum problems. For each one, an optima is memorized. The regions near these solutions are explored by PLS, but the other regions are neglected. For bi-objective instances, or small tri-objective instances, the number of considered weighted sum problems is sufficiently large so that neglected regions are small. But for larger tri-objective instances, the number of considered weighted sum problems becomes insufficient so that neglected regions are much larger. PD-TPLS-l avoids this clustering effect by optimizing a lot of weighted sum problems and launching its MO LS from the corresponding local optima.

Although PDA solves the same number of sub-problems as MoMad, data perturbation circumvents this clustering effect and even brings much better results than PD-TPLS-l for large tri-objective instances. The bad results of PD-TPLS-l compared to PDA also indicate that multiplying the search directions by optimizing (non perturbed) weighted sum problems, is not a good alternative to approximate the non-dominated set of the MOTSP. Lust and Teghem [11] pointed out the same issue on bi-objective instances.

Contrary to the bi-objective case, PDA examines much less solutions than MoMad. Several differences between PDA and MoMad can explain these results:

1. During the decomposition phase, PDA memorizes all the local optima with distinct fitness values generated by the LK heuristic, whereas MoMad does not. Given that a LK run generates $n + 1$ local optima, and PDA and MoMad call the same number of LK runs, PDA will examine in the worst case $n + 1$ times more solutions than MoMad during the decomposition phase.
2. Contrary to the MoMad's PLS, the PDA's PLS does not explore the neighborhood of a solution proven to be inefficient. Therefore, given a starting set of solutions, the PDA's PLS will examine less solutions than the MoMad's PLS.
3. PDA finds more potentially efficient solutions than MoMad for all tested instances, as indicated in tables 2, 3, 4 and 5. Thus PLS may explore the neighborhood of more solutions than MoMad.

Table 6 highlights, for two instances of different sizes and number of objectives, the number of solutions examined by the two main algorithmic components shared by MoMad, PDA and PD-TPLS-l: the single objective local search component (ILS) and the MO local search component (PLS). Not surprisingly, for the three methods, the largest proportion of solutions are examined by PLS. However, the proportion of solutions examined by PLS in PDA is lower than in MoMad. Therefore, as expected in Section 3.4, PDA saves computational resources thanks to its PLS and

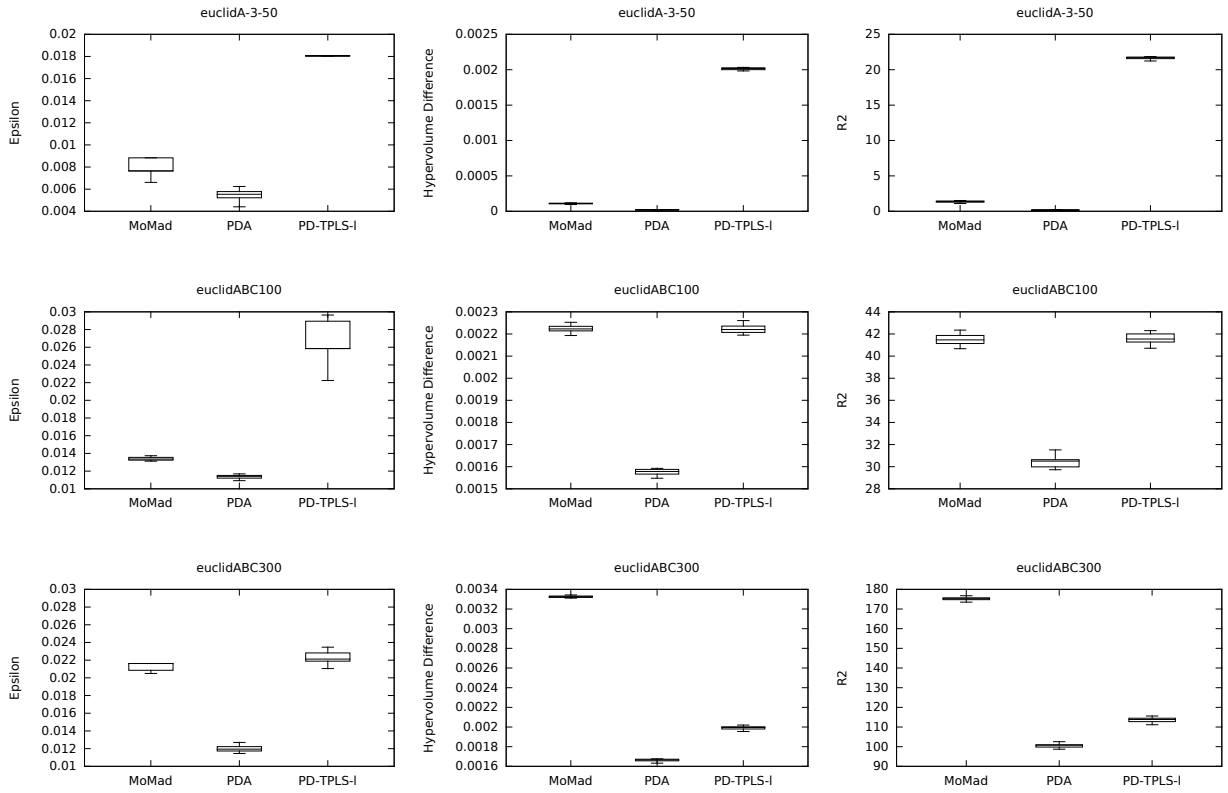

Figure 5: I_ϵ (left), I_H^- (middle) and I_{R2} (right) comparison between MoMad, PDA and PD-TPLS-l on euclidA-3-50 (1st line), euclidABC100 (2nd line), euclidABC300 (3rd line) instances.

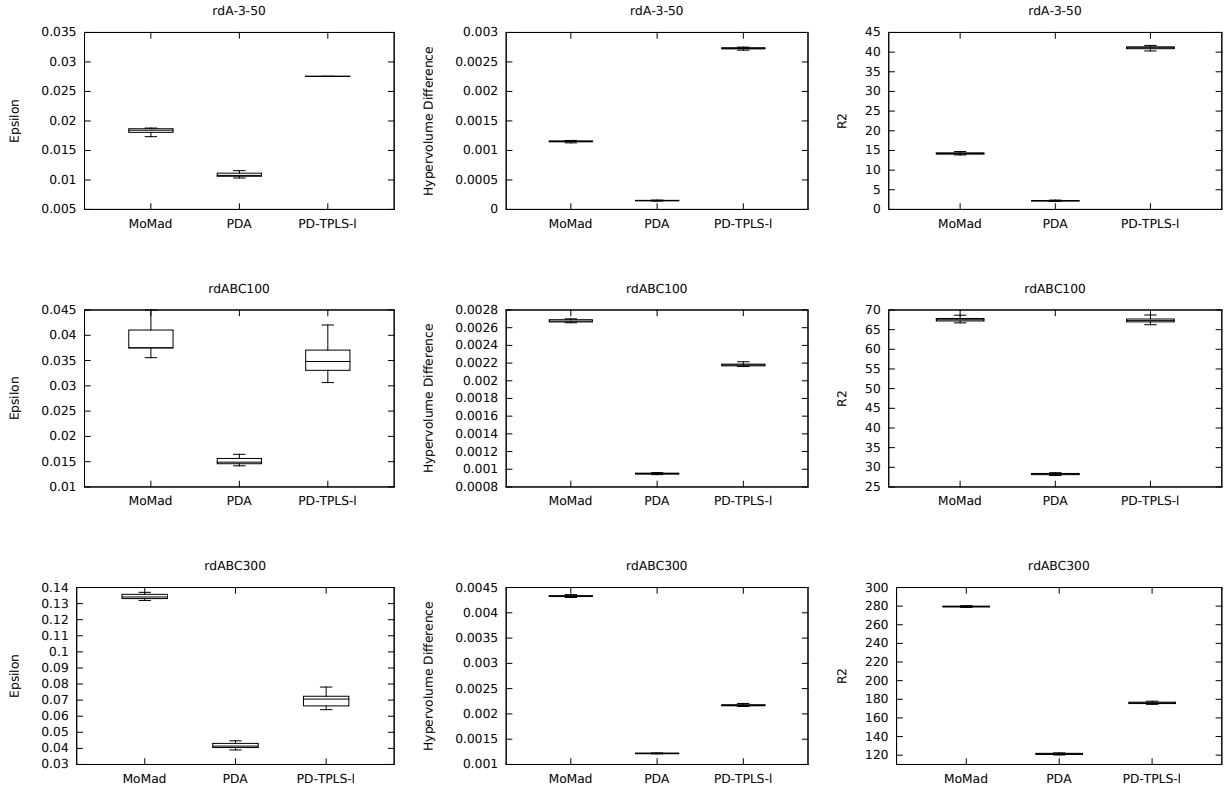

Figure 6: I_ϵ (left), I_H^- (middle) and I_{R2} (right) comparison between MoMad, PDA and PD-TPLS-l on rdA-3-50 (1st line), rdABC100 (2nd line), rdABC300 (3rd line) instances.

Instance	Algorithm	I_ϵ (10^{-2})	I_H^- (10^{-4})	I_{R2}	$\|A\|$	$\|\widetilde{Z}_{nd}\|$	Nb. of exam. sol. (10^6)	Time (s)
rdA-3-30	MoMad	2.39 $\pm$ 1.37	4.01 $\pm$ 0.26	3.02 $\pm$ 0.17	12,249	15,215	6.53	24
	PDA	**1.54** $\pm$ 1.43	**1.11** $\pm$ 0.06	**0.97** $\pm$ 0.11	**12,865**		5.49	24
	PD-TPLS-l	3.29 $\pm$ 0	33.9 $\pm$ 0.23	26.4 $\pm$ 0.31	5,690		264	168
rdB-3-30	MoMad	2.32 $\pm$ 0.23	3.77 $\pm$ 0.24	3.35 $\pm$ 0.32	12,374	15,651	6.42	25
	PDA	**1.59** $\pm$ 0.05	**0.80** $\pm$ 0.04	**0.69** $\pm$ 0.14	**13,298**		5.57	25
	PD-TPLS-l	3.69 $\pm$ 0.02	26.9 $\pm$ 0.17	31.1 $\pm$ 0.39	5,645		257	253
rdC-3-30	MoMad	1.85 $\pm$ 0.23	2.99 $\pm$ 0.10	3.33 $\pm$ 0.19	11,720	14,695	5.88	23
	PDA	**1.52** $\pm$ 0.16	**1.04** $\pm$ 0.05	**0.91** $\pm$ 0.13	**12,429**		4.95	23
	PD-TPLS-l	3.62 $\pm$ 0.03	27.0 $\pm$ 0.24	26.3 $\pm$ 0.17	5,318		245	267
rdA-3-40	MoMad	2.24 $\pm$ 0.25	9.16 $\pm$ 0.24	10.7 $\pm$ 0.28	23,804	39,101	21.6	85
	PDA	**1.60** $\pm$ 0.08	**1.05** $\pm$ 0.02	**1.44** $\pm$ 0.11	**30,378**		20.4	85
	PD-TPLS-l	3.79 $\pm$ 0.18	30.8 $\pm$ 0.17	36.1 $\pm$ 0.29	10,624		606	349
rdB-3-40	MoMad	2.16 $\pm$ 0.09	7.64 $\pm$ 0.14	7.91 $\pm$ 0.33	24,088	37,859	22.1	81
	PDA	**1.41** $\pm$ 0.12	**0.96** $\pm$ 0.02	**1.08** $\pm$ 0.09	**30,343**		20.4	81
	PD-TPLS-l	3.05 $\pm$ 0.06	28.4 $\pm$ 0.15	35.0 $\pm$ 0.17	10,236		615	575
rdC-3-40	MoMad	2.13 $\pm$ 0.07	8.75 $\pm$ 0.19	10.0 $\pm$ 0.38	29,586	47,796	27.0	86
	PDA	**1.23** $\pm$ 0.13	**1.15** $\pm$ 0.04	**1.36** $\pm$ 0.12	**37,098**		26.1	86
	PD-TPLS-l	3.31 $\pm$ 0	28.7 $\pm$ 0.14	36.2 $\pm$ 0.26	12,281		627	611
rdA-3-50	MoMad	1.85 $\pm$ 0.11	11.5 $\pm$ 0.11	14.2 $\pm$ 0.22	53,634	108,039	74.8	210
	PDA	**1.08** $\pm$ 0.06	**1.50** $\pm$ 0.05	**2.20** $\pm$ 0.09	**74,105**		67.0	210
	PD-TPLS-l	2.83 $\pm$ 0.20	27.3 $\pm$ 0.18	41.1 $\pm$ 0.38	22,293		897	784
rdB-3-50	MoMad	2.14 $\pm$ 0.12	12.2 $\pm$ 0.15	16.4 $\pm$ 0.38	39,589	81,522	54.1	197
	PDA	**1.20** $\pm$ 0.08	**1.29** $\pm$ 0.05	**1.78** $\pm$ 0.09	**58,846**		52.3	197
	PD-TPLS-l	3.01 $\pm$ 0.06	28.9 $\pm$ 0.13	47.1 $\pm$ 0.35	17,478		920	943
rdC-3-50	MoMad	1.96 $\pm$ 0.15	10.2 $\pm$ 0.09	15.9 $\pm$ 0.26	49,902	97,807	67.3	194
	PDA	**1.19** $\pm$ 0.08	**1.33** $\pm$ 0.04	**1.93** $\pm$ 0.08	**69,584**		60.5	194
	PD-TPLS-l	3.32 $\pm$ 0.27	25.6 $\pm$ 0.19	46.3 $\pm$ 0.40	20,651		916	889
rdABC100	MoMad	3.89 $\pm$ 0.25	26.8 $\pm$ 0.14	67.5 $\pm$ 0.47	26,218	179,330	119	979
	PDA	**1.51** $\pm$ 0.06	**9.51** $\pm$ 0.07	**28.3** $\pm$ 0.21	**34,334**		64.5	891
	PD-TPLS-l	3.53 $\pm$ 0.30	21.8 $\pm$ 0.16	67.4 $\pm$ 0.57	22,667		1,951	892
rdE-3-200	MoMad	8.34 $\pm$ 0.08	37.8 $\pm$ 0.15	180 $\pm$ 0.58	43,397	713,669	489	2,421
	PDA	**2.36** $\pm$ 0.22	**13.6** $\pm$ 0.10	**86.4** $\pm$ 0.47	**53,705**		261	2,007
	PD-TPLS-l	5.21 $\pm$ 0.37	21.7 $\pm$ 0.16	121 $\pm$ 0.81	42,747		3,846	2,007
rdABC300	MoMad	10.3 $\pm$ 0.17	43.3 $\pm$ 0.13	280 $\pm$ 0.60	50,365	970,441	958	4,083
	PDA	**4.18** $\pm$ 0.17	**12.2** $\pm$ 0.06	**121** $\pm$ 0.71	**68,526**		578	3,225
	PD-TPLS-l	7.06 $\pm$ 0.45	21.7 $\pm$ 0.17	176 $\pm$ 1.00	49,981		5,948	3,225

Table 5: Comparison between PDA, MoMad and PD-TPLS-l results on random tri-objective instances.

transfers them to ILS, especially in the tri-objective case.

Instance	p	n	Algorithm	ILS		PLS		Total
				Nb. of exam. sol. (10^6)	Proportion of exam. sol. (%)	Nb. of exam. sol. (10^6)	Proportion of exam. sol. (%)	(10^6)
euclidAB500	2	500	**MoMad**	**0.25**	0.07	**337.6**	99.93	**337.9**
			PDA	0.88	0.22	391.2	99.78	392.1
euclidABC100	3	100	MoMad	1.89	3.59	50.8	96.41	52.7
			PDA	3.82	12.78	**26.1**	87.22	**29.9**
			PD-TPLS-l	**0.01**	<0.01	2454.69	>99.99	2454.7

Table 6: Number of solutions examined by the different algorithmic components of MoMad, PDA and PD-TPLS-l on the bi-objective instance euclidAB500 and on the tri-objective instance euclidABC100.

Note that the size of $\widetilde{Z}_{nd}$ strongly grows with n, reaching approximately one million points for the instance rdABC300 (Table 5). For instances of size $n \geq 100$, the approximation sets found by the methods are much smaller than $\widetilde{Z}_{nd}$. This is due to the use of the ϵ-archives, which bounds the size of the approximation while preserving the diversity of the points. Indeed, results of PDA are still of good quality. One can consider the instance rdABC300, for which PDA has its worst I_ϵ results. On this instance, the approximation obtained by PDA is worst than $\widetilde{Z}_{nd}$ by only a factor of 4.18% (in average), indicating that PDA generates well dispersed approximation sets over $\widetilde{Z}_{nd}$.

4.7.3. Further comments on the results

In this section, we are wondering how PDA explores the feasible set X to obtain such good results in comparison to MoMad and PD-TPLS-l. For this purpose, let us define the notion of *cluster* for the MOTSP. A cluster is a set of solutions that are reachable from each other by applying an *elementary move* [52]. In this article, we consider as elementary move the 2-edge-exchange move. Intuitively, a local search routine like PLS starting from a solution of the cluster and using the exhaustive 2-edge-exchange neighborhood function, may reach any of the solutions inside the cluster.

As described in Section 2.3, the idea of IPLS is to run a PLS over some solutions to find the potentially efficient solutions sharing the same clusters. When PLS has explored all the reachable clusters, and thus is stuck in a locally efficient set, some solutions of the current set are perturbed in order to discover new clusters.

However, as pointed out by Paquete and Stützle in their study on clusters [52], the degree of clustering of solutions depends on the MOCO problem and the instance type. Therefore, algorithms using PLS may have more difficulties on instances with weakly clustered solutions, in the sense that they will be stuck faster in a locally efficient set.

In such instances, we claim that data perturbation can be useful to discover more quickly new clusters, and thus escaping more efficiently from locally efficient sets. To illustrate this, we have calculated the number of non unary clusters found by the different methods (PDA, MoMad and PD-TPLS-l) on several bi-objective and tri-objective instances. Results are reported in Tables 7 and 8. We can see that PDA finds every time a larger number of clusters than the two other methods, which do not use data perturbation. Furthermore, we have shown in the previous section that the solutions found by PDA, and thus the clusters found, are of better quality than its competitors for all instances.

This shows the usefulness of data perturbation: by modifying in a non linear way the search directions, PDA has access to clusters hardly attainable with the other methods.

We can notice that the performances of PDA shown in the previous section are strongly correlated with the additional number of clusters found by our method than its competitors. For example, given the ClusterAB500 instance, Table 2 reports that the average quality of the approximations found by PDA when compared with MoMad is at least 3 times better according to the indicators used; whereas the number of clusters found by PDA reported in Table 7 is 1.7 larger than MoMad. On the other hand, the results of PDA on the euclidA-3-30 instance (reported in Table 4) are slightly better than those of MoMad, whereas the number of clusters found by the former is just a little bit larger than the latter.

Instance	Algorithm	Nb. of clusters
ClusterAB100	MoMad	312
	PDA	**451**
ClusterAB300	MoMad	1,339
	PDA	**2,486**
ClusterAB500	MoMad	3,006
	PDA	**4,967**

Instance	Algorithm	Nb. of clusters
kroAB100	MoMad	291
	PDA	**463**
kroAB300	MoMad	1,192
	PDA	**2,078**
kroAB500	MoMad	2354
	PDA	**4,037**

Instance	Algorithm	Nb. of clusters
rdAB100	MoMad	147
	PDA	**174**
rdAB300	MoMad	431
	PDA	**696**
rdAB500	MoMad	663
	PDA	**1,089**

Table 7: Average number of clusters for clustered (left table), Euclidean (middle table) and random (right table) bi-objective instances.

Instance	Algorithm	Nb.of clusters
euclidA-3-30	MoMad	1,386
	PDA	**1,447**
	PD-TPLS-l	521
euclidA-3-40	MoMad	5,220
	PDA	**6,156**
	PD-TPLS-l	1,665
euclidA-3-50	MoMad	10,235
	PDA	**11,434**
	PD-TPLS-l	2,907

Instance	Algorithm	Nb. of clusters
rdA-3-30	MoMad	2,461
	PDA	**2,561**
	PD-TPLS-l	794
rdA-3-40	MoMad	5,017
	PDA	**6,487**
	PD-TPLS-l	1,333
rdA-3-50	MoMad	11,074
	PDA	**16,551**
	PD-TPLS-l	2,541

Table 8: Average number of clusters for Euclidean (left table) and random (right table) tri-objective instances.

5. Conclusion

In the present work, we introduced a new efficient component in Decomposition algorithms: the data perturbation [37, 3]. We showed that by using data perturbation combined with PLS and the Decomposition framework, we can obtain very good results on MOTSP, both on the bi-objective and tri-objective cases.

PDA has been compared to two other state-of-the-art algorithms: MoMad [35] and PD-TPLS-l [23]. On bi-objective instances, our method performs better than MoMad. On tri-objective instances, PDA obtains better results than both MoMad and PD-TPLS-l.

These good results are explained by the fact that PDA has a better exploration strategy of the feasible set X to find efficient solutions, by finding more clusters of solutions of better quality than methods not using data perturbation.

The main drawback of PDA is its input parameters: the maximum range of data perturbation and the number of sub-problems have to be fixed by the user. We analyzed the effect of different values of maximum range of data perturbation on the final quality of our method. Future researches will concern an adaptive way to find a good compromise on the number of sub-problems.

Considering the perspective of PDA, we first make a focus on the MOTSP. It would be interesting to use the Chained Lin-Kernighan heuristic [49] as single-objective ILS solver in the main loop, instead of the 3-opt first improvement currently used; and finding efficient speed-up techniques to run efficiently a 3-edge-exchange neighborhood for PLS instead of a 2-edge-exchange.

PDA could be adapted to other MOCO problems, such as the MO Quadratic Assignment Problem, where the solutions seems to be less clustered than for the MOTSP [52]; and for which data perturbation might be even more useful.

Acknowledgments The authors thank T. Lust, L. Paquete and H. Li for providing them helpful source codes.

References

[1] M. Ehrgott, Multicriteria Optimization, Springer, 2005.

[2] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731.

[3] B. Codenotti, G. Manzini, L. Margara, G. Resta, Perturbation: An efficient technique for the solution of very large instances of the euclidean TSP, INFORMS Journal on Computing 8 (2) (1996) 125–133.

[4] R. E. Steuer, Multiple criteria optimization: theory, computation, and applications, Wiley, 1986.

[5] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, Springer, 2003.

[6] T. Stützle, Iterated local search for the quadratic assignment problem, European Journal of Operational Research 174 (3) (2006) 1519–1539.

[7] E.-G. Talbi, M. Rahoual, M. H. Mabed, C. Dhaenens, A hybrid evolutionary approach for multicriteria optimization problems: Application to the flow shop, in: EMO, Vol. 1, Springer, 2001, pp. 416–428.

[8] E. Angel, E. Bampis, L. Gourves, A dynasearch neighborhood for the bicriteria traveling salesman problem, in: Metaheuristics for Multiobjective Optimisation, Springer, 2004, pp. 153–176.

[9] L. Paquete, M. Chiarandini, T. Stützle, Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study, in: Metaheuristics for Multiobjective Optimisation, Springer, 2004, pp. 177–199.

[10] A. Liefooghe, J. Humeau, S. Mesmoudi, L. Jourdan, E.-G. Talbi, On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems, Journal of Heuristics 18 (2) (2012) 317–352.

[11] T. Lust, J. Teghem, Two-phase pareto local search for the biobjective traveling salesman problem, Journal of Heuristics 16 (3) (2010) 475–510.

[12] M. Ehrgott, X. Gandibleux, Hybrid metaheuristics for multi-objective combinatorial optimization, in: Hybrid metaheuristics, Springer, 2008, pp. 221–259.

[13] M. M. Drugan, D. Thierens, Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies, Journal of Heuristics 18 (5) (2012) 727–766.

[14] M. Drugan, D. Thierens, Path-guided mutation for stochastic pareto local search algorithms, in: Parallel Problem Solving from Nature XI, Springer, 2010, pp. 485–495.

[15] P. Merz, Advanced fitness landscape analysis and the performance of memetic algorithms, Evolutionary Computation 12 (3) (2004) 303–325.

[16] A. Jaszkiewicz, Genetic local search for multi-objective combinatorial optimization, European Journal of Operational Research 137 (1) (2002) 50–71.

[17] H. Ishibuchi, T. Murata, Multi-objective genetic local search algorithm, in: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, IEEE, 1996, pp. 119–124.

[18] H. Ishibuchi, T. Murata, A multi-objective genetic local search algorithm and its application to flowshop scheduling, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 28 (3) (1998) 392–403.

[19] L. Paquete, T. Stützle, A two-phase local search for the biobjective traveling salesman problem, in: Evolutionary Multi-Criterion Optimization, Springer, 2003, pp. 479–493.

[20] J. Dubois-Lacoste, M. López-Ibáñez, T. Stützle, Improving the anytime behavior of two-phase local search, Annals of mathematics and artificial intelligence 61 (2) (2011) 125–154.

[21] A. Jaszkiewicz, P. Zielniewicz, Efficient adaptation of the pareto memetic algorithm to the multiple objective travelling salesperson problem, in: Proceedings of the 7th International Conference devoted to Multi-Objective Programming and Goal Programming, 2006.

[22] R. Kumar, P. Singh, Pareto evolutionary algorithm hybridized with local search for biobjective TSP (2007) 361–398.

[23] L. Paquete, T. Stützle, Design and analysis of stochastic local search for the multiobjective traveling salesman problem, Computers & Operations Research 36 (9) (2009) 2619–2631.

[24] T. Lust, A. Jaszkiewicz, Speed-up techniques for solving large-scale biobjective TSP, Computers & Operations Research 37 (3) (2010) 521–533.

[25] L. Paquete, Stochastic local search algorithms for multiobjective combinatorial optimization: methods and analysis, Vol. 295, IOS Press, 2005.

[26] Y. P. Aneja, K. P. K. Nair, Bicriteria transportation problem, Management Science 25 (1) (1979) 73–78.

[27] J. L. Cohon, Multiobjective programming and planning, Academic Press, New York, 1978.

[28] H. Li, D. Landa-Silva, An adaptive evolutionary multi-objective approach based on simulated annealing, Evolutionary Computation 19 (4) (2011) 561–595.

[29] D. S. Johnson, L. A. McGeoch, The traveling salesman problem: A case study in local optimization, Local search in combinatorial optimization 1 (1997) 215–310.

[30] J. J. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA Journal on computing 4 (4) (1992) 387–411.

[31] J. Cheng, G. Zhang, Z. Li, Y. Li, Multi-objective ant colony optimization based on decomposition for bi-objective traveling salesman problems, Soft Computing 16 (4) (2012) 597–614.

[32] C. García-Martínez, O. Cordón, F. Herrera, A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP, European Journal of Operational Research 180 (1) (2007) 116–148.

[33] M. López-Ibáñez, T. Stützle, The automatic design of multiobjective ant colony optimization algorithms, IEEE Trans. Evolutionary Computation 16 (6) (2012) 861–875.

[34] T. Murata, H. Ishibuchi, M. Gen, Specification of genetic search directions in cellular multi-objective genetic algorithms, in: Evolutionary multi-criterion optimization, Springer, 2001, pp. 82–95.

[35] L. Ke, Q. Zhang, R. Battiti, A simple yet efficient multiobjective combinatorial optimization method using decompostion and pareto local search, IEEE Trans on Cybernetics 44 (10) (2014) 1808–1820.

[36] L. Ke, Q. Zhang, R. Battiti, Moea/d-aco: A multiobjective evolutionary algorithm using decomposition and antcolony, Cybernetics, IEEE Transactions on 43 (6) (2013) 1845–1859.

[37] I. Charon, O. Hudry, The noising method: a new method for combinatorial optimization, Operations Research Letters 14 (3) (1993) 133–137.

[38] I. Charon, O. Hudry, The noising methods: a survey, in: Essays and surveys in metaheuristics, Springer, 2002, pp. 245–261.

[39] G. Reinelt, TSPLIB - a traveling salesman problem library, ORSA journal on computing 3 (4) (1991) 376–384.

[40] E. Zitzler, Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, Shaker Verlag, 1999.

[41] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, V. G. Da Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 117–132.

[42] M. P. Hansen, A. Jaszkiewicz, Evaluating the quality of approximations to the non-dominated set, IMM, Department of Mathematical Modelling, Technical University of Denmark, 1998.

[43] C. M. Fonseca, L. Paquete, M. López-Ibánez, An improved dimension-sweep algorithm for the hypervolume indicator, in: Congress on Evolutionary Computation, IEEE, 2006, pp. 1157–1163.

[44] N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, J. Vahrenhold, On the complexity of computing the hypervolume indicator, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 1075–1082.

[45] C. M. Fonseca, J. D. Knowles, L. Thiele, E. Zitzler, A tutorial on the performance assessment of stochastic multiobjective optimizers, in: Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005), Vol. 216, 2005, p. 240.

[46] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, The annals of mathematical statistics (1947) 50–60.

[47] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian journal of statistics (1979) 65–70.

[48] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations research 21 (2) (1973) 498–516.

[49] D. Applegate, W. Cook, A. Rohe, Chained Lin-Kernighan for large traveling salesman problems, INFORMS Journal on Computing 15 (1) (2003) 82–92.

[50] P. C. Borges, M. P. Hansen, A basis for future successes in multiobjective combinatorial optimization, Technical report, Department of Mathematical Modelling, Technical University of Denmark.

[51] M. Laumanns, L. Thiele, K. Deb, E. Zitzler, Combining convergence and diversity in evolutionary multiobjective optimization, Evolutionary computation 10 (3) (2002) 263–282.

[52] L. Paquete, T. Stützle, Clusters of non-dominated solutions in multiobjective combinatorial optimization: An experimental analysis, in: Multiobjective Programming and Goal Programming, Springer, 2009, pp. 69–77.