Pareto Adaptive Decomposition algorithm

Marek Cornu, Tristan Cazenave, Daniel Vanderpooten

PSL, Universite Paris-Dauphine, LAMSADE
Place du Marechal de Lattre de Tassigny, 75 016 Paris, France

Abstract

Dealing with multi-objective combinatorial optimization and local search, this article proposes a new multi-objective meta-heuristic named Pareto Adaptive Decomposition algorithm (PAD). Combining ideas from decomposition methods, two phase algorithms and multi-armed bandit, PAD provides a 2-phase modular framework for finding an approximation of the Pareto front. The first phase decomposes the search into a number of scalarized problems by linear aggregation of the original multi-objective problem. Following a data perturbation step, the second phase conducts an iterative process: a number of scalarized problems are selected by a multi-armed bandit policy and optimized by a single-objective local search solver. Resulting solutions will serve as a starting point of a multi-objective local search procedure, called Pareto Local Search. Based on this framework, we conduct experiments on several instances of the bi-objective symmetric Traveling Salesman Problem. The experiments show that our proposed algorithm outperforms the best current method on this problem.

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.

This article presents the Pareto Adaptive Decomposition algorithm (PAD), a new MO local search meta-heuristic. PAD provides a modular framework by combining ideas from the Multi-Objective Memetic Algorithm based on Decomposition [9], the Two Phase Pareto Local Search [11] and the mortal multi-armed bandit paradigm [6]. The paper is organized as follows: Sect 1 first recalls the formal definition of a MO combinatorial optimization (MOCO) problem and fundamental definitions from MO optimization. Then, we introduce basic concepts of local search algorithms, followed by a presentation of related work. Sect 2 describes the proposed algorithm, PAD. Finally Sect 3 is devoted first to the adaptation of PAD to the bi-objective Traveling Salesman Problem (bTSP) and then the computational experiments. Sect 4 concludes on the contributions and perspectives of PAD.

1.1 Multi-objective optimization

Let ξ be a finite set of q elements $\xi := \{e_1, ...e_q\}$, defining a combinatorial structure. Let $c_1 : \xi \to \mathbb{N}, ..., c_p : \xi \to \mathbb{N}$ be the $p \geq 2$ *cost functions* on the elements of ξ. Considering the *minimization sum* version, a MOCO problem is defined as:

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

given the *feasible set* $X \subset 2^{\xi}$ and the p potentially conflicting *objective functions* $f_i : X \to \mathbb{N}$ such that $f_j(x) = \sum_{e \in x} c_j(e)$ for a given feasible solution $x \in X$, for each $j = 1, ..., p$.

Let $Z \subset \mathbb{R}^p$ be the objective space and $Z_X := f(X) = \{z \in Z : z_j = f_j(x) \text{ for } j = 1,...,p \text{ and } x \in X\}$ the *attainable objective set,* mapping a 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 $z \prec z'$, 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 $z \preceq z'$, if $z_j \leq z'_j$ for each *j*=1,...,*p*.

An attainable objective point $z \in Z_X$ is called *non dominated* if and only if there is no other $\widehat{z} \in Z_X$ such that $\widehat{z} \prec z$.

A feasible solution $x \in X$ is called *efficient* if its image in 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 defined as $z_j^* := \min_{x \in X} f_j(x)$ for each *j*=1,...,*p*. The *nadir point* is defined as $\eta_j := \max_{x \in X_E} f_j(x)$ for each *j*=1,...,*p*. Let $\lambda = (\lambda_1, ..., \lambda_p)$ be a weighting vector (*weight*).

The weighted sum problem is given by :

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

In this paper, the weighting vector λ is normalized such that $\lambda_j \geq 0$ for each *j*=1,...,*p* and $\sum_{j=1}^{p} \lambda_j = 1$. The resulting scalarized problem is a single-objective instance of the original MO problem.

Supported efficient solutions are optimal solutions of a weighted sum scalarization for some vector $\lambda > 0$. The image in objective space of the supported efficient solutions corresponds 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.

1.2 Single-objective and multi-objective local search

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

Given a single-objective minimization problem with objective function $g : X \rightarrow \mathbb{R}$, an *iterative improvement local search* algorithm searches, at each step, from the current solution $x \in X$ for a neighbor $y \in \mathcal{N}_k(x)$ such that $g(y) < g(x)$. If such a neighbor exists, it moves to y, otherwise the method stops and x is a *local optimum*.

A *(stochastic) perturbation move* (called *kick*) 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 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 x.

An *iterated local search* (ILS) algorithm [10] builds a *sequence* of locally optimal solutions by iteratively perturbing the current solution by use of a stochastic kick, and then applying iterative improvement local search after starting from the modified solution.

In MOCO, we search for a set of solutions rather than a single solution. We define the *neighborhood of a set* of feasible solutions as the union of the neighborhood of each solution.

Pareto Local Search (PLS) [2] is the MO version of iterative improvement local search. Starting from a set of solutions, PLS searches for the set of all potentially efficient neighbors. A new iteration starts from the newly found set and searches again for potentially efficient neighbors. PLS continues until no more potentially efficient neighbors have been identified, stopping in what we call a *local optima set*.

Iterated PLS (IPLS) [8] is the MO version of ILS. Starting from an initial set of solutions, IPLS iteratively performs a PLS, then perturbs (by use of stochastic kicks) some potentially efficient solutions previously found. Newly generated solutions compose a new starting set for PLS. This technique can escape from a local optima set.

1.3 Related work

The concept of decomposition [16], introduced by the MO Evolutionary Algorithm based on Decomposition (MOEA/D) method, is simple yet efficient: it decomposes a MO optimization problem into a number of scalarized problems and optimizes them simultaneously in a collaborative way.

Two phase Pareto Local Search (2PPLS) [11] is a bi-objective stochastic local search method. The first phase follows a dichotomous scheme [1], tackling a well-suited number of weighted sum problems by use of a single-objective ILS solver. Resulting solutions serve as a starting set for PLS. The main idea of 2PPLS is the following: 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, 2PPLS first generates a set of high quality solutions covering well the Pareto front by approximating the supported efficient solution set. PLS is used in the second phase to generate a more accurate approximation of the Pareto front by searching for the remaining solutions, the non supported ones. The authors experimentally show that this first phase drastically increases the final quality and convergence speed of PLS.

The Multi-Objective Memetic Algorithm based on Decomposition (MOMAD) [9] is a recent method combining ideas from 2PPLS [11] and MOEA/D [16]. As a decomposition method, MOMAD is an iterative algorithm and first decomposes the MOCO problem into several weighted sum problems. Each of these problems is defined by a weight λ given a unique search direction, and maintains a best-so-far solution (*incumbent*) according to $\mathrm{ws}(\cdot, \lambda)$ for the entire duration of the run. Such as 2PPLS, an ILS is applied on each single-objective problem in order to initialize the incumbents with high quality potentially supported efficient solutions covering well the Pareto front. Then the iterative part begins. At each iteration, MOMAD conducts a PLS. When PLS stops, MOMAD apply 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 by perturbation is not new and has already been used in [8] with the notion of IPLS. However, [9] hybridizes IPLS with the decomposition methodology.

To our knowledge, MOMAD is the state-of-the-art algorithm on the bTSP and the MO multidimensional knapsack problem. This paper compares our algorithm PAD, with MOMAD on the bTSP. Recent experiments [9] have shown that MOMAD outperforms the 2PPLS method on all the tested instances, so comparing our algorithm with 2PPLS does not seem to be relevant.

2 Pareto Adaptive Decomposition algorithm

This section describes the algorithm presented in this paper, the Pareto Adaptive Decomposition algorithm (PAD). Sect. 2.1 presents the PAD framework, while the following subsections describe its main components. In the different algorithms, the symbols $\downarrow$, $\uparrow$, and $\updownarrow$ specify the parameter transmission, respectively in, out and in-out.

2.1 General framework

PAD is a two phase iterative method using the idea of decomposition [16]. Basically, the first phase decomposes the search into a number of *sub-problems*; the second phase iteratively produces a starting set by use of ILS and conducts a PLS from this set of potentially efficient solutions.

First phase begins by the *decomposition phase*. It produces several *sub-problems*, initially weighted sum problems. Each of them is defined by a weight λ; and maintains the best so far solution minimizing $\mathrm{ws}(\cdot, \lambda)$ called *incumbent*, regularly updated during the search. The number of sub-problems remains fixed during the entire duration of the run. Then, an efficient ILS is applied on each sub-problem to initialize the incumbents with high quality potentially supported efficient solutions covering well the Pareto front. What follows makes a great difference compared with MOMAD. Each sub-problem, corresponding up to now to a weighted-sum one, has already been optimized by an ILS. Thus many potential supported efficient solutions have been already found. We claim that continuing the optimization on these directions during the second phase, like MOMAD does, may be too restrictive. Indeed, we will search for the potentially non supported ones, which are usually much more numerous than supported ones in a MOCO problem.

This way, the set of starting solutions provided for running PLS will be much larger, thus interesting

for the global quality and the convergence behavior of PLS. Because these solutions can not be found by a single-objective solver on a weighted-sum problem, we use the data perturbation technique [7] which adds a random noise to a cost matrix related to a weighted sum problem. By adding such a noise in a scalarized cost matrix, we produce a non-linear change to the direction of the search of the sub-problem, which remains a single-objective version of the MOCO problem. Thus, ILS still can optimize such problem and provide potentially efficient supported or non supported solutions. So the scalarized matrix related to each sub-problem are perturbed.

Framework: Pareto Adaptive Decomposition

Input : stopping criterion, MO cost matrix C, decomposition parameter σ, maximum depth of PLS l, data perturbation parameter d

Output: set A of potentially efficient solutions

Step 1) Decomposition phase: $\Pi \leftarrow$`Decomposition` $(C\downarrow, \sigma\downarrow, A\updownarrow)$, where $\Pi = \left(\pi^1, ..., \pi^K\right)$

Step 2) Initial settings:

Step 2.1) Set $P_{PLS} \leftarrow A$, the starting set of PLS

Step 2.2) Perturbation: for each $\pi^k \in \Pi$ do : $C^k \leftarrow$`DataPerturbation` $(C\downarrow, \lambda^k\downarrow, d\downarrow)$

Step 3) Adaptive optimization phase: for it=1 to K do :

Step 3.1) Select a sub-problem: $\pi^b \leftarrow$`AdaptiveGreedySelection` $(\Pi\downarrow)$

Step 3.2) Iterated local search: Set $P \leftarrow$`IteratedLocalSearch` $(C^b\downarrow, x^b\downarrow)$

Step 3.3) Update of Π, A and P_{PLS}: for each $x \in P$ do :

- `UpdateIncumbents` $(x\downarrow, \Pi\updownarrow)$
- if `UpdateSolutionSet` $(x\downarrow, \mathrm{A}\updownarrow)$ then : `UpdateSolutionSet` $(x\downarrow, P_{PLS}\updownarrow)$

Step 3.4) Update attributes: $r^b \leftarrow r^b + |P \cap A|$; $s^b \leftarrow s^b + 1$

Step 4) Reset the worst sub-problem: Select $\pi^w \leftarrow \underset{\pi^k \in \Pi}{\arg\min}\{\frac{r^k}{s^k}\}$

Then: $C^w \leftarrow$`DataPerturbation` $(C\downarrow, \lambda^w\downarrow, d\downarrow)$; $r^w, s^w \leftarrow 0$

Step 5) Pareto local search: `ParetoLocalSearch` $(l\downarrow, P_{PLS}\downarrow, A\updownarrow, \Pi\updownarrow)$; $P_{PLS} \leftarrow \emptyset$

Step 6) : If the stopping criterion is satisfied, then stop and return A. Otherwise, go to **Step 3**.

The second phase iteratively: selects several sub-problems and applies an ILS from their respective incumbent; then runs a PLS starting from the newly found potentially efficient solutions. The sub-problem selection used at each iteration is inspired by a recent Multi-armed Bandit (MAB) selection policy [4] called Adaptive Greedy Selection [6]. Knowing that a major role of ILS is to provide new potentially efficient solutions to restart PLS stuck in a local optima set, the MAB policy will direct the selection towards the sub-problems from which ILS finds more such solutions.

As previously mentioned, each sub-problem maintains an incumbent. Each time a new (potentially efficient) solution is found, it will be compared to all the incumbents and may replaces one of them if it is better than the incumbent on the related weighted sum value. Introduced in [9], this procedure allows to regularly updates incumbents. This way, incumbents follow the evolution of the approximation set and provide good quality and well-dispersed solutions for ILS.

Because data perturbation is a stochastic procedure, it can produce problems whose optimal solution is not efficient for the addressed MOCO problem, or simply problems which provide via ILS fewer

potentially efficient solutions than the other ones. To overcome this issue, we need to modify them by perturbing again their scalarized cost matrix. However it may be very resource-consuming to do this for each sub-problem concerned. Instead, at each iteration, just before PLS, we select the sub-problem from which ILS was the least efficient for finding potentially efficient solutions. Then we reset its cost matrix by perturbing it again. The combination of data perturbation, sub-problem selection policy and sub-problem reset procedure constitutes a major contribution of our work and provides to our algorithm an adaptive behavior. Indeed, PAD initially proposes a set of search directions by perturbing all the sub-problems once; then, at each iteration, it tends to apply more ILS from the best ones and to search for new search directions by perturbing again the less effective ones.

Finally, at the end of an iteration, PAD conducts a PLS from the set of newly generated potentially efficient solutions, provided by ILS.

The framework of PAD (page 4) depicts the main steps previously described. PAD is an anytime algorithm and takes as input any stopping criterion. It considers a MO cost matrix C, a decomposition parameter σ used by the decomposition phase to define the number of sub-problems, the maximum depth of PLS l limiting the computational cost of PLS, and the data perturbation parameter d controlling the maximum variation of the noise put into scalarized cost matrix. PAD maintains two set of solutions: the set of potentially efficient solutions found so far A and the starting set of PLS P_{PLS}.

PAD uses several auxiliary procedures. The `IteratedLocalSearch` procedure conducts an ILS which not only returns the best solution, but the set of local optima whose images in the objective space do not dominate each other. This is one of the differences between [9, 11] and our work. Updates of solution sets are made by the `UpdateSolutionSet` procedure, which adds a solution to a set of solutions whose images do not dominate each other. It returns true if and only if the solution has been added to the set. The method maintains the set of K sub-problems $\Pi = (\pi^1, ..., \pi^K)$, where K remains fixed. For each $k = 1, ..., K$, a sub-problem π^k is a tuple $(\lambda^k, C^k, x^k, r^k, s^k)$ composed of five elements: the weight λ^k linearly scalarizing the MO cost matrix C into the scalarized matrix C^k, the incumbent minimizing ws $\left(\cdot, \lambda^k\right)$, and two attributes used in the second phase of PAD: the cumulative number r^k of efficient solutions found by ILS conducted from x^k, and the selection counter s^k counting the amount of times π^k has been selected during the second phase. The procedure `UpdateIncumbents` manages incumbent updates. Given a solution $x \in X$ and the set of sub-problems Π, the procedure searches in a random order for an incumbent x^k in Π such that ws $\left(x, \lambda^k\right) <$ ws $\left(x^k, \lambda^k\right)$, then x^k is replaced by x and the procedure stops. Otherwise x^k is discarded and the procedure continues until all incumbents have been discarded. In PAD, each time a solution is added to the set A of potentially efficient solutions or generated by ILS, it will be compared to the incumbents through this procedure.

2.2 Decomposition phase

Given the MO cost matrix C, the parameter σ fixing the number K of sub-problems and the set A of potentially efficient solutions, the `Decomposition` procedure (**Step 1**) decomposes a MOCO problem into a number of well dispersed weighted sum problems. It first generates the set of weights $\Lambda = \{\lambda = (\lambda_1, ..., \lambda_p) : \sum_{j=1}^{p} \lambda_j = \sigma, \lambda_j \in \{0, ..., \sigma\}, j = 1, ..., p\}$ such that $|\Lambda| = K = \binom{p+\sigma-1}{\sigma}$ following the method presented in [5]. For each weight λ^k from Λ, it first builds the linearly scalarized cost matrix C^k and then solves the related weighted-sum problem ws $\left(\cdot, \lambda^k\right)$ using an ILS. The starting solution for ILS is generated by a domain-dependent heuristic. Given P the set of solutions whose images are non dominated to each other, we set as incumbent the solution x^k which minimizes ws(x, λ^k) where $x \in P$, and updates A with the solutions of P. The attributes r^k and s^k are both initialized to 0. Finally, the sub-problem $\pi^k \leftarrow (\lambda^k, C^k, x^k, r^k, s^k)$ is added to the set of sub-problems Π, returned at the end of the procedure.

2.3 Data perturbation

Data perturbation [7] in MO optimization has been first introduced in [11]. In this paper, data perturbation needs only one parameter, whereas the one in [11] needs three parameters. The principle of data

perturbation used in PAD is to add a random noise into a linearly scalarized cost matrix. Given the MO cost matrix C, a weight λ and the data perturbation parameter $d :\in]0;1[$ controlling the maximum variation of the noise, the `DataPerturbation` procedure (**Step 2.2**) computes for each $e \in \xi$ its perturbed value $ws_{perturb}(e, \lambda, d) = \mathcal{U}(1-d, 1+d) \times \sum_{j=1}^{p} \lambda_j c_j(e)$ and inserts it into a new single-objective perturbed cost matrix, returned at the end of the procedure. The higher d, the greater the perturbation is.

2.4 Sub-problem selection

The purpose of the sub-problem selection issue is to select sub-problems providing a maximum of potentially efficient solutions. This situation involves an exploration vs exploitation trade-off and it is natural to link our issue to the Multi-Armed bandit (MAB) problem [4]. A MAB problem models a set of K slot machines, where each machine (arm) has a different and unknown reward distribution. The goal is to sequentially select the optimal arm to pull, in order to maximize the cumulative reward. Obviously, arms in the MAB problem stand for sub-problems in our selection issue and reward stands for the number of potentially efficient solutions generated by a sub-problem. As seen in SubSect 2.1, at each iteration the worst sub-problem is reset. This refers to the Mortal MAB problem [6], where available arms are assumed to be born and die regularly such that the number K of arms remains fixed, like the number of our sub-problems. In [6], the authors show that the UCB algorithm [14] commonly used for the original MAB problem, is no longer suitable for the Mortal version and propose the Adaptive Greedy algorithm [6]. The `AdaptiveGreedySelection` procedure (**Step 3.1**) works as follows: given the set of sub-problems Π, the sub-problem π^m maximizing the ratio $\{\frac{r^i}{s^i}\}$ $(i = 1, ..., K)$ is chosen. Then, π^m is selected with probability equals to $\min(1, \frac{r^m}{s^m})$, otherwise a random one is selected among the set of sub-problems Π.

In the vast majority of decomposition algorithms [16], all sub-problems receive about the same amount of computational resource. To our knowledge, [17] first introduced the idea of managing in a different way the sub-problem selection. However this approach is not founded on the MAB paradigm but rather on an ad hoc one and their set of sub-problems was not dynamic, thus we do not consider their algorithm.

2.5 Pareto Local Search

Algorithm 1 (**Step 5**) presents a modified version of the original PLS [2]. 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 incumbents (Line 5). When the neighborhood of all solutions of P has been explored, a new PLS is conducted on P_{next} (Line 9). Following [9], PLS stops when P_{next} is empty or the limited depth of the algorithm l is exceeded. During the process, P is static and A can be updated with potentially efficient neighbors. Thus it may append that the image of a newly generated neighbor entering A dominates the image of a solution in P. Our modifications prevent the exploration of the neighborhood of such a non efficient solution: Line 2 checks if $x \in P$ is still in A, and Line 7 switches of current solution as soon as its image is dominated. Thus contrary to previous works [9, 11], we never explore the neighborhood of a solution proven to be non efficient. These modifications are added from the original PLS to limit the computational resources allocated to PLS and transfer them to ILS.

3 Computational experiments

Computational experiments of PAD are conducted on the bTSP. We first recall the formal definition of the multi-objective TSP (MO TSP), then present the benchmark on bTSP instances and introduce the notion of quality indicators. After parameters settings, we compare the results of PAD and MOMAD, the state-of-the-art algorithm on bTSP.

Algorithm 1: `ParetoLocalSearch`

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

Output: A, Π

```
1 P_next ← ∅
2 foreach x ∈ P ∩ A do
3     foreach y ∈ N_PLS(x) such that f(x) ⊀ f(y) do
4         if UpdateSolutionSet(y↓, A↕) then
5             UpdateIncumbents(y↓, Π↕)
6             UpdateSolutionSet(y↓, P_next↕)
7             if f(y) ≺ f(x) then break;
8 if P_next ≠ ∅ and l > 1 then
9     ParetoLocalSearch(l − 1↓, P_next↓, A↕, Π↕)
```

3.1 Definition of the problem

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 one. The goal is to find a tour such that the total cost is minimized. In the MO TSP, the traveling salesman has to minimize several different costs (potentially conflicting) at the same time. More formally, we define the MO TSP as follows. Given a complete graph $G = (V, \xi)$ with $V = \{v_1, ..., v_n\}$ the set of n nodes and $\xi := \{e_1, ...e_q\}$ corresponds to the set of edges such that $q = \frac{n(n-1)}{2}$, the MO TSP is defined by Eq. 1 where $x \in X$ represents an Hamiltonian cycle on G. We are interested here in the bTSP, that is $p = 2$. The MO TSP is $\mathcal{NP}$-hard and intractable, even if $p = 2$.

3.2 Benchmark instances

We experiment PAD on nineteen bTSP instances. We consider four different types of instances.

First, the Euclidean instances: an instance is composed of two single-objective euclidean instances. For each objective, the costs between the edges correspond to the Euclidean distance between two cities in a plane. The ten following euclidean instances are used: three instances published in [13]: euclid100, euclid300 and euclid500. Two instances generated on the basis of TSPLIB instances [15]: kroAB100, kroAB200. And finally five other instances generated in [11] : kroAB300, kroAB400, kroAB500, kroAB750, kroAB1000.

Second, the random instances: the costs between the edges are randomly generated from a uniform distribution. We use three random instances published in [13]: rdAB100, rdAB300 and rdAB500.

Next, the mixed instances: 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 [13] are used: mixedAB100, mixedAB300 and mixedAB500.

Finally, the clustered instances: an instance is composed of two 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. We consider three clustered instances generated in [11]: ClusteredAB100, ClusteredAB300 and ClusteredAB500.

3.3 Multi-objective quality indicators

This section presents two of the most used quality indicators to compare approximation sets in MOCO, the *hypervolume indicator* [18] and the ϵ*-indicator* [19]. The computation of these indicators implies to know the 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 $\widetilde{Z}_{ND}$.

Given a set A and a reference point $\bar{z} \in Z$ weakly dominated by every point of A, the hypervolume indicator I_H value of A with regards of $\bar{z}$ measures the hypervolume of the region of the objective space which is weakly dominated by A and weakly dominates $\bar{z}$. The higher $I_H(A)$, the better the approximation set A is. On each tested instance, the reference point $\bar{z}$ has been set to $\bar{z} = \widetilde{\eta} + \frac{1}{10}(\widetilde{\eta} - \widetilde{z}^*)$ where $\widetilde{\eta}$ and $\widetilde{z}^*$ are respectively the approximation of η and z^*, based on $\widetilde{Z}_{ND}$.

Given an approximation set A and the approximation of the Pareto front $\widetilde{Z}_{ND}$, the ϵ-indicator I_ϵ gives the factor by which A is worse than $\widetilde{Z}_{ND}$ with respect to all objectives: $I_\epsilon(A, \widetilde{Z}_{ND}) = \inf_{\epsilon \in \mathbb{R}} \{\forall z' \in \widetilde{Z}_{ND} \exists z \in A : z \preceq_\epsilon z'\}$, where $z \preceq_\epsilon z'$ if and only if $z_j \leq (1+\epsilon) z'_j$ for $j = 1, ..., p$. The lower $I_\epsilon(A, \widetilde{Z}_{ND})$, the better the approximation set A is comparing to $\widetilde{Z}_{ND}$.

3.4 Experimental parameter setup and adaptation of PAD to the bTSP

We have implemented ourselves MOMAD [9]. Parameters have been fixed as follows, as done in [9]: the maximum number of iterations for PLS is set to 10. The number of sub-problems is set to $\min(n, 600)$ except for rdAB300 and rdAB500, for whose MOMAD had difficulties and thus authors have increase the number of sub-problems to respectively 400 and 600. The number of iterations is fixed to 500 for all instances. As done in [11], MOMAD uses the Chained Lin-Kernighan (LK) heuristic [3] as ILS in initialization, a 2-opt best improvement with random double-bridge kicks as ILS in the main loop, and a 2-exchange neighborhood for PLS.

Concerning PAD, we also choose as ILS for the decomposition phase the Chained LK heuristic. We use its default parameters but we modified it to memorize all the generated local optima. The neighborhood function of PLS $\mathcal{N}_{PLS}$ chosen is the *2-exchange* neighborhood, as suggested in [5]. A candidate edge list is associated to $\mathcal{N}_{PLS}$ in the same manner as [9]: it consists of all edges composing at least one solution of the current population P_{PLS} of our method. The list is updated each time P_{PLS} is updated by the `UpdateSolutionSet` procedure. During the main loop of PAD, a 3-opt first improvement with biased random double-bridge kicks and same candidate edge list as PLS, is conducted as ILS. We use the implementation of Paquete[1]. For each instance, the number K of sub-problems in PAD is the same as done in MOMAD to have a fair comparison. The parameter fixing the depth of PLS has been set to 10 as in [9], and the parameter controlling the perturbation has experimentally been fixed to 5%. The stopping criterion used by PAD for a given instance corresponds to the average time spent by MOMAD on this instance. Due to the lack of space, we can not present different parameter settings of PAD for the tested instances.

The two algorithms have been run 20 times on each instance and compared using I_H and I_ϵ. All experiments presented were performed on a 3.4 GHz computer with 16Gb RAM on a Linux OS. All algorithms are written in C++.

3.5 Experimental results

Table 1 presents for the nineteen bTSP benchmark instances the average values and standard deviation of I_H and I_ϵ, the average size $|A|$ of the approximation set returned for each algorithm, the size $|\widetilde{Z}_{ND}|$ of the Pareto front approximation and the average time spent by the two algorithms. Numbers in bold indicate a better average value. PAD has better average values on both I_H, I_ϵ and even $|A|$ on all the tested instances. Note that unlike I_ϵ, even a minor difference between values of I_H is important. In order to statistically compare PAD and MOMAD, the Mann-Whitney non-parametric statistical test [12] has been applied. For a specific indicator on a given instance, this test assesses whether two algorithms come from the same distribution. When the hypothesis is satisfied, it means there is no difference between the values of the quality indicator obtained by the algorithms. Otherwise average values are simply

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

compared. Given a starting level of risk of 1%, the Mann-Whitney test indicates that PAD is better than MOMAD for each tested instance and quality indicator used.

Instance Name	Size	I_H $(\times 10^6)$ PAD	I_H $(\times 10^6)$ MOMAD	$I_\epsilon(\times 10^{-3})$ PAD	$I_\epsilon(\times 10^{-3})$ MOMAD	$\lvert A\rvert$ PAD	$\lvert A\rvert$ MOMAD	$\lvert\widetilde{Z}_{ND}\rvert$	Time (s)
ClusterAB100	100	**26019** (± 0.15)	26016 (± 0.20)	**1.68** (± 0.34)	4.06 (± 0.49)	**2774**	2454	3009	14
ClusterAB300	300	**249323** (± 1.58)	249256 (± 6.37)	**1.65** (± 0.29)	6.19 (± 0.47)	**18423**	15931	20952	209
ClusterAB500	500	**853581** (± 2.50)	853532 (± 1.97)	**1.28** (± 0.44)	7.02 (± 1.01)	**45605**	40936	51496	678
euclidAB100	100	**21005** (± 0.06)	21002 (± 0.27)	**2.04** (<0.01)	3.64 (± 0.39)	**1613**	1400	1790	10
euclidAB300	300	**226973** (± 0.50)	226962 (± 0.27)	**0.87** (± 0.10)	1.28 (± 0.01)	**16329**	14436	18191	164
euclidAB500	500	**712502** (± 2.37)	712477 (± 0.40)	**0.90** (± 0.14)	1.08 (± 0.01)	**37851**	34148	42987	606
kroAB100	100	**26905** (± 0.04)	26901 (± 0.28)	**1.98** (± 0.37)	3.94 (± 0.28)	**3046**	2569	3298	11
kroAB200	200	**105196** (± 0.13)	105188 (± 0.18)	**1.06** (± 0.09)	3.41 (± 0.07)	**7884**	6649	8760	59
kroAB300	300	**248952** (± 0.98)	248942 (± 0.27)	**0.90** (± 0.14)	1.82 (± 0.14)	**16900**	14880	18685	169
kroAB400	400	**432421** (± 1.72)	432400 (± 0.48)	**1.21** (± 0.29)	2.07 (<0.01)	**24911**	21793	28942	346
kroAB500	500	**674447** (± 2.66)	674422 (± 0.44)	**1.11** (± 0.22)	2.18 (<0.01)	**37914**	33436	43164	632
kroAB750	750	**1598812** (± 4.99)	1598769 (± 0.72)	**1.13** (± 0.16)	1.61 (± 0.07)	**66829**	60415	74519	1417
kroAB1000	1000	**2806068** (± 10.71)	2805997 (± 1.02)	**1.55** (± 0.34)	2.45 (<0.01)	**105198**	98476	112892	2466
mixedAB100	100	**32589** (± 0.14)	32576 (± 0.56)	**4.69** (± 0.67)	15.22 (± 1.26)	**1480**	983	1824	11
mixedAB300	300	**345841** (± 1.87)	345803 (± 0.51)	**8.22** (± 0.88)	24.70 (± 0.14)	**7386**	5786	10274	185
mixedAB500	500	**1029245** (± 4.02)	1029198 (± 0.59)	**7.89** (± 1.24)	12.18 (± 0.39)	**16545**	14008	21460	721
rdAB100	100	**53556** (± 0.54)	53522 (± 1.25)	**6.37** (± 0.88)	21.88 (± 2.88)	**1167**	639	1636	11
rdAB300	300	**490918** (± 4.44)	490817 (± 1.10)	**10.38** (± 0.94)	32.14 (± 1.06)	**3245**	2078	5334	206
rdAB500	500	**1443005** (± 7.61)	1442858 (± 1.14)	**13.09** (± 1.50)	24.02 (± 0.60)	**4842**	3533	6702	757

Table 1: Comparison between PAD and MOMAD results for each instance.

Standard deviation is globally lower for MOMAD compared to PAD, particularly for I_H on most of the instances and for I_ϵ on large scale euclid and kro instances. This shows in the same time the strength and the limit of MOMAD: by constantly running ILS every time in the same directions (without data perturbation), the incumbents will be updated in an homogeneous way from a run to another. This process provides a reliable algorithm, therefore it brings a lack of diversity as well. On the other hand, the variations of values for our method are not negligible for a large part of tested instances.

This behavior is due to the use of data perturbation which induces an additional level of stochasticity in our method compared with MOMAD. However, we inform the reader that for both I_H and I_ϵ collected values, the best value found by MOMAD never reaches the worst one found by our method for all the instances, except for I_ϵ on euclidAB500. We can not show this result by lack of place.

4 Conclusion

We showed in this paper that by using a combination of local search and MAB techniques, we can obtain very good results on the bTSP. To our knowledge, this is the first time in a decomposition algorithm [16] the MAB paradigm [4] and the Adaptive Greedy Algorithm [6] are used to manage the selection of sub-problems and the data perturbation technique [7] is used.

PAD has been compared on large-scale bTSP instances to MOMAD, another state-of-the-art method on this problem. Our method outperforms MOMAD both on the I_H and I_ϵ indicators. Furthermore, it can be adapted to other MOCO problems, such as the MO multi-dimensional knapsack and the MO Assignment Problem. However, the results of PAD are not as stable as the MOMAD ones.

It would be important to study the influence of the parameters and the different components on the result quality. Focusing on the MO TSP, it would be a great improvement to use the Chained Lin-Kernighan heuristic [3] as single-objective ILS solver in second phase. Maybe the most interesting future work concerning PAD is its application to a $p \geq 3$ optimization. The reader can notice that PAD can be directly used for a $p \geq 3$, however some modifications have to be done to enhance its efficiency in such spaces.

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

References

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

[2] E. Angel, E. Bampis, and L. Gourves. A dynasearch neighborhood for the bicriteria traveling salesman problem. In *Metaheuristics for Multiobjective Optimisation*, pages 153–176. Springer, 2004.

[3] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[4] D. A. Berry and B. Fristedt. *Bandit problems: sequential allocation of experiments*. Springer, 1985.

[5] P. C. Borges and M. P. Hansen. A basis for future successes in multiobjective combinatorial optimization. *Technical report*, 1998.

[6] D. Chakrabartiand, R. Kumar, F. Radlinski, and E. Upfal. Mortal multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 273–280, 2009.

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

[8] M. Drugan and D. Thierens. Path-guided mutation for stochastic pareto local search algorithms. In *Parallel Problem Solving from Nature XI*, pages 485–495. Springer, 2010.

[9] L. Ke, Q. Zhang, and R. Battiti. A simple yet efficient multiobjective combinatorial optimization method using decompostion and pareto local search. *IEEE Trans on Cybernetics*, 2014.

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

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

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

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

[14] A. Peter, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

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

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

[17] Q. Zhang, W. Liu, and H. Li. The performance of a new version of moea/d on cec09 unconstrained mop test instances. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 203–208, 2009.

[18] E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, volume 63. Citeseer, 1999.

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