THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'Université Paris-Dauphine

Local Search, data structures and Monte Carlo Search for Multi-Objective Combinatorial Optimization Problems

École doctorale de Dauphine – ED 543

Spécialité Informatique

COMPOSITION DU JURY :

Tristan CAZENAVE
Professeur à l'Université Paris-Dauphine
Directeur de thèse

Daniel VANDERPOOTEN
Professeur à l'Université Paris-Dauphine
Co-Directeur de thèse

Xavier GANDIBLEUX
Professeur à l'Université de Nantes
Rapporteur

Laetitia JOURDAN
Professeur à l'Université de Lille 1
Rapporteure

Thibaut LUST
Lecturer au National College of Ireland
Membre du jury

Frédéric SAUBION
Professeur à l'Université d'Angers
Président du jury

Soutenue par **Marek CORNU**
le **18/12/2017**

Dirigée par **Tristan CAZENAVE**

DAUPHINE UNIVERSITÉ PARIS | PSL RESEARCH UNIVERSITY PARIS

Abbreviations

2PIPLS/D	2-Phase Iterated Pareto Local Search with Decomposition
2PPLS	2-Phase Pareto Local Search
A-MDW	Adaptive Maximally Dispersed set of Weights
A-NMCS	Aggregation-based Nested Monte Carlo Search
AP	Assignment Problem
C-LK	Chained Lin-Kernighan
FSP	Flowshop Scheduling Problem
ILS	Iterated Local Search
IPLS	Iterated Pareto Local Search
KP	0-1 Knapsack Problem
LS	Local Search
MCTS	Monte Carlo Tree Search
MDW	Maximally Dispersed set of Weights
MO	Multi-Objective
MOCO	Multi-Objective Combinatorial Optimization
MOFRMP	Multi-Objective French Regions Mapping Problem
MoMad	Multi-Objective Memetic Algorithm based on Decomposition
NMCS	Nested Monte Carlo Search
PD-TPLS	Pareto Double Two-Phase Local Search
PLS	Pareto Local Search
PLS-VND	Pareto Local Search with Variable Neighborhood Descent
P-PLS	Partitioned Pareto Local Search
QAP	Quadratic Assignment Problem
SO	Single-Objective
TSP	symmetric Traveling Salesman Problem
VND	Variable Neighborhood Descent

Remerciements

À mes deux encadrants,

merci à Tristan Cazenave et Daniel Vanderpooten d'avoir accepté de diriger ma thèse. Leurs enseignements ont été précieux pour moi et cette thèse m'a permis d'apprendre une quantité incroyable de choses dont je suis persuadé qu'elles me seront essentielles pour l'avenir. Je les remercie de la bienveillance dont ils ont fait preuve à mon égard ainsi que l'autonomie qu'ils m'ont laissée dans mon travail. Au travers de leur proposition de stage de M2, je les remercie de m'avoir introduit dans cet univers qu'est la recherche en informatique dont j'ignorais tout, ainsi que de m'avoir fait rencontré des gens aussi passionants que passionnés. Je remercie Tristan d'avoir accepté de jouer le rôle de directeur officiel avec l'administratif que cela induit, et Daniel d'avoir organisé notre classe "optimisation multi-objectif" à Wuppertal qui fut un bon souvenir.

Aux membres du jury,

je remercie chaleureusement Xavier Gandibleux et Laetitia Jourdan qui ont accepté d'être rapporteurs de ma thèse, ainsi que Thibaut Lust et Frédéric Saubion, qui ont accepté d'être examinateurs. Je remercie tous les membres du jury pour leur relecture de ma thèse, d'autant plus que je les sais être des personnes très demandées dans leur travail.

Aux enseignants et chercheurs de Dauphine et d'ailleurs,

je remercie Anne Epaulard pour son aide sur le problème des régions. Merci à toute l'équipe du LAMSADE, en particulier Cristina Bazgan et Lucie Galand qui étaient du voyage à Wuppertal, ainsi que l'équipe multi-objectif de l'Université de Wuppertal pour leur accueil chaleureux : Britta, Kathrin, Kerstin et Michael. Je remercie aussi les enseignants que j'ai pu avoir durant mes études à Dauphine. En particulier, je remercie Attila Raksanyi de m'avoir donné une sorte de seconde chance pour mon premier contrôle de Java. Je remercie aussi Marie-Jo Bellosta me m'avoir orienté vers un parcours de recherche.

À mes amis lamsadiens,

aux petits jeunes : Amin, Anne, Céline, Diana, George, Mehdi ; aux nouveaux docteurs : Alexandre, Amine, Meriem, Youcef ; aux vieux docteurs : Abdallah, Amine, Dalal, Edouard, Lydia, Nat, Rafael, Raja ; à Cédric, Khalil, Pedro, Yassine et tous les autres thésards du labo, merci pour l'ambiance géniale qui reigne dans les trois bureaux de thésards.

Merci à Fabien de couver sous son aile les petits thésards que nous sommes. Merci à Ian et Thomas de la bonne humeur dont ils font toujours preuve. Merci à mon co-bureau Justin pour son humour.

Merci à Marcel et Tom pour les discussions politiques passionnantes qu'on a pu avoir ensemble.

Je remercie Anaëlle, Florian, Lyes, Olivier, Renaud, Sami et Satya pour toutes ces journées et soirées passées ensemble. Merci à Renaud pour ses différentes invitations dans les Pyrénées ou en Allemagne, qui furent de très bons moments de détente. Merci à Anaëlle, pour la touche de jeunesse et de féminité qu'elle a apportée dans ce groupe de brutes. Je remercie Satya avec qui j'ai pu passer des moments privilégiés. Je lui suis reconnaissant pour ses conseils, sa gentillesse et l'avenance dont il a toujours su faire preuve.

À mes amis non lamsadiens,

merci à Brice B., Brice L., Charlotte, Christophe, Doris, Fabien, Fabiola, Florine, Hit, Laetitia, Louis, Maud, Nathalie, Pauline Ald., Pauline Ali. et ... Pauline G. pour tous ces moments passés ensemble.

Merci à Bastien pour toutes les soirées passées à refaire le monde, toujours un verre à la main et toujours prêt à faire n'importe quoi, quel que soit le lieu, allant du pub irlandais jusqu'aux plages malaisiennes en passant par les ruelles nocturnes parisiennes et scéennes.

À ma belle-famille,

merci à mes beaux-parents de m'avoir permis de squater leurs apparts pendant toutes ces années.

À ma famille,

je remercie mes parents et mon frère, pour le soutien, la patience et la compréhension dont ils ont fait preuve depuis le tout début de cette aventure. Je suis bien conscient que ces dernières années n'ont pas été évidentes pour eux non plus. Je remercie aussi mes parents pour tout ce qu'ils ont fait pour mon frère et moi afin que nous ne manquions jamais de rien.

Merci à mes grands-parents ; mes oncles et tantes, mes cousins et cousines ; à ma belle soeur et mon parrain, ainsi qu'aux amis de la famille.

Je n'oublie pas non plus les petits : Bouboule, Jean-Joseph, Léon, Mitard, Nono, Patrick, Shamu, Tamtam, Tiburcin et Valentine.

Et enfin à Céline,

qui partage ma vie depuis plus de neuf ans déjà. Tout d'abord, merci de supporter mon sale caractère. Je ne peux qu'être admiratif de la tolérance absolue dont tu as fait preuve façe à mon rythme de vie infernal que tu as subi ces dernières années. Merci pour le soutien que tu m'a apporté pendant mes si nombreux moments de doute et de stress. Merci de me connaître si bien et d'avoir su quand il fallait que je lâche prise. Merci pour tous les moments passés ensemble et qui ont su me faire oublier la thèse quelques instants.

Contents

Introduction **1**

1 Fundamental concepts of MOCO **5**

1.1 Fundamental concepts and techniques 6

1.2 Complexity, intractability and examples of MOCO problems 10

2 Methods in MOCO **13**

2.1 MO quality indicators 15

2.2 MO meta-heuristics 18

2.3 MO Traveling Salesman Problem: state-of-the-art 38

2.4 Archives: state-of-the-art 43

3 New archives **47**

3.1 AVL-Archive 48

3.2 NDR*-Archive 66

3.3 Experiments 77

3.4 Conclusion 89

4 New optimization methods **91**

4.1 Single-objective optimizers: improvements for MOCO 92

4.2 Adaptive Maximally Dispersed set of Weights 93

4.3 Partitioned Pareto Local Search 101

4.4 2-Phase Iterated Pareto Local Search with Decomposition 109

4.5 Preservation of a good distribution of points in the objective space 113

4.6 Conclusion 115

5 Application of 2PIPLS/D to MOTSP **117**

5.1 MOTSP benchmarks and experimental design 118

5.2 Implementation of 2PIPLS/D to address MOTSP 121

5.3 Empirical evidence of global convexity on MOTSP 123
5.4 Parameter setting of 2PIPLS/D 126
5.5 Comparison of 2PIPLS/D with state-of-the-art methods 137
5.6 Conclusion 154

6 Application of 2PIPLS/D to MOFRMP **155**
6.1 Presentation of MOFRMP 156
6.2 Implementation of 2PIPLS/D to address MOFRMP 161
6.3 Study of global convexity on MOFRMP 163
6.4 Experiments 164
6.5 Conclusion 169

Conclusion **171**

References **173**

Introduction

Nowadays, companies and even people are frequently faced with problems of discrete nature, called *combinatorial optimization problems*. A combinatorial problem involves managing (selecting a subset of or ordering) a finite set of elements while respecting a number of *constraints*. These elements are the elementary components of the *solutions* of the problem. The quality of a solution of the problem is evaluated by an *objective* (also called *criterion*) value. When two solutions are compared, either one solution is better than the other given this objective, or both have the same objective value. The aim of *Combinatorial Optimization* is to find one (or even several) of the best solutions of the problem.

Combinatorial optimization problems are numerous in today's lives and appear in various situations. A simple example of such problems would be the packing of your bag before vacations. This situation forces you to select carefully the most useful stuffs for your trip, knowing that your bag has a limited place. Another situation regularly experienced by anyone is to find the shortest path for moving from a location to another, regardless of the mean of transport; problem generally solved by geolocation systems (Galileo, GPS, ...) embedded in some portable devices, or even solved by web applications. Companies are also confronted to combinatorial optimization problems specific to their field of activity. For example, an airline company faces each day the problem of combining schedules of thousands of flight attendants, in order to operate all the daily flights with a minimum cost. Telecommunications companies (Orange, AT&T, ...) are extensively confronted to different combinatorial optimization problems. Among them, an important issue is related to the design of a telecommunication network: how to connect a number of homes of a village or a city block to Internet with high-speed access minimizing the total length of the cables used? Another example concerns the service of any delivery company (La Poste, Fedex, ...), which has to deliver packages to a set of customers by using a fleet of transport vehicles. The problem arising each day is to deliver packages while minimizing either the total fuel consumed or the total distance covered by all vehicles through their traveling from the warehouses to the consumers.

There are many different methods to optimize combinatorial problems but we can categorize them into two main classes : *exact methods* and *approximation methods*. Exact methods aim at finding an *optimal* (i.e. best) solution of the problem, while the purpose of approximation methods is to find a sub-optimal (i.e. *approximate*) solution of good quality. There are two sub-classes of approximation methods: the *approximation methods with performance guarantee* and the *heuristics*. The approximation methods with performance guarantee provide a sub-optimal solution guaranteed to be optimal at a given precision. A heuristic does not provide any guarantee on the quality of the solution generated. In addition to the quality, a key aspect of all these methods is to provide a solution in a time said to be *reasonable*. In general, guaranteeing the quality of a solution takes extra time compared to a method not providing any guarantee, and proving that a solution is optimal takes even more extra time. Therefore, each one of these three classes of methods is useful depending on the *difficulty* of finding an optimal solution for the addressed problem. Globally, the difficulty of the *instance* of a problem depends on two main axes: the number of elements composing the instance (called the *size* of the instance) and the type of constraints involved. Concerning problems said to be *easy*, like the *Shortest Path Problem* or the telecommunication network design problem previously cited, exact methods generally provide an optimal solution in a reasonable time, even for large-size instances. On the other hand, for problems similar to the first given example, called *Packing Problems*, it

may be difficult for an exact method to find a solution in a reasonable time and it might be advisable to use instead an approximation method with performance guarantee to reduce the execution time for finding a good quality solution. Finally, concerning the presented problem faced by the delivery companies, called the *Vehicle Routing Problem*, it is considered as a *hard* problem, and even for small instances, an exact method will have great difficulties to find an optimal solution. Optimizing an instance of such a problem with an approximation method with performance guarantee is realistic up to a certain size, but for large-size instances only heuristics are able to find good quality solutions in a reasonable time.

Such hard problems like the Vehicle Routing Problem belong to the class of *NP-hard problems* [Garey and Johnson, 1979]. A conjecture widely accepted nowadays states that the time necessary to find an optimal solution of a NP-hard problem is, in the worse case, exponential in the size of the instance, making a proof of optimality infeasible in practice or even a good performance difficult to guarantee as the size of real-world instances is often too large. Meta-heuristics, which are heuristics proposing a general framework to optimize different problems, are extensively used for such NP-hard problems [Dréo et al., 2003, Gendreau and Potvin, 2005]. There is a wide range of different meta-heuristics for combinatorial problems, among others: evolutionary algorithms [Goldberg, 1989, Bäck et al., 1997], local search methods [Kirkpatrick et al., 1983, Glover, 1989, Feo and Resende, 1995, Hoos and Stützle, 2004], nature-inspired meta-heuristics [Dorigo et al., 2006, Kennedy and Eberhart, 1997], Monte Carlo Search methods [Ginsberg, 2001, Gelly and Silver, 2007, Browne et al., 2012].

At this point, we have considered the optimization of a single objective. On the other hand, solutions of combinatorial problems may be evaluated on several objectives, often conflicting. Conflicts between objectives generally lead to the following fundamental difference with single-objective optimization: two solutions can have different objective values, representing different *trade-offs* of the objectives, and consequently none is better than the other. Optimizing together several objectives of a combinatorial problem gave rise to the domain of *Multi-Objective Combinatorial Optimization* (MOCO).

A classical example of conflicting objectives concerns routing problems like the Shortest Path Problem or the Vehicle Routing Problem previously presented. For such problems, several objectives can be considered: the travel time, the ecological footprint and naturally the travel cost depending on the mean of transport chosen. We can easily see that a number of these objectives are generally conflicting, like the travel time versus the travel cost or the ecological footprint. However, as a topical example, it can be interesting for an environment-conscious company (or even a person) to select a route optimizing both travel time and its ecological footprint. Considering such an extra ecological-oriented objective would benefit the company by promoting its environment-consciousness for attracting more customers. Concerning the scheduling problem faced by the airplane company, in addition to the cost minimization, we could consider as second objective the well-being of flight attendants, by taking into account their preferences of working days. Although potentially conflicting with a minimization of costs, optimizing as well the well-being of employees would potentially reduce social movements and also improve the brand image of the company. Another example of conflicting objectives appears in a common problem faced in finance. It consists in choosing a portfolio out of a given set of investment proposals while optimizing two conflicting objectives: the expected value of portfolio returns to maximize, and the risk inherent to portfolio returns to minimize. More generally, MOCO presents numerous real world applications in finance, transportation, medicine and telecommunication [Ehrgott, 2009], routing problems and telecommunications [Gabrel and Vanderpooten, 2002, Martins and Ribeiro, 2006, Madakat et al., 2013, Zavala et al., 2014], structural design problems [Clímaco and Pascoal, 2012].

As we saw, MOCO is often a valuable alternative for decision makers compared to single-objective combinatorial optimization, as it covers different points of view existing for a problem. In contrast, MOCO is confronted to computational issues when multiple and conflicting objectives are considered. More particularly, most MO problems are theoreticly difficult, in the sense that their decision version is knowing to be NP-hard, even if the underlying single-objective version is *polynomially solvable*, i.e. easy.

As for single-objective optimization, MOCO methods are classified into exact methods, approximation meth-

ods with performance guarantee and (meta-)heuristics. In addition to this methodology classification, three main approaches exist in MOCO and each one is more or less demanding for the decision maker [Deb et al., 2016]. The first one is the *interactive approach*, which iteratively interacts with the decision maker by asking him/her preferences through the selection, classification or ranking of different solutions suggested by the method in order to guide the search, and to finally obtain a solution that suits him/her [Roy, 1985, Vanderpooten and Vincke, 1989]. The second one is the *a priori approach* which aims at first asking the preferences of the decision maker among all objectives, then focus the optimization process given these guidelines. At the end, the method produces a single or a small set of solutions. These two types of approaches often aggregate the objectives into a single one such that the problem can then be solved as a single-objective problem. A wide variety of aggregations can be found in the literature, from weighted sum or weighted Tchebychev [Belhoul et al., 2014] to more complex ones able to model complex decision maker's preferences as the Choquet integral [Galand et al., 2010, Lust and Rolland, 2013]. The third one is the *a posteriori approach*, for which the preferences of the decision maker are not known a priori, i.e. before the optimization process of the problem. This approach is strongly attached to the notion of *Pareto dominance*: a solution dominates another solution if it is equal or better on all objectives and strictly better on at least one objective. Considering this notion, the decision maker would not be interested in dominated solutions, and consequently the aim of an a posteriori approach is to find the whole set of solutions not dominated by any other solution, called the *efficient set*. Despite the fact that in practice we search for a reduced efficient set, i.e. an efficient set such that only a single solution is memorized among equivalent ones, the set may be extremely large, particularly in the case of multiple and conflicting objectives. Thus, even for moderately-sized problems, it is usually computationally prohibitive to identify an exact reduced efficient set. In particular, several MOCO problems are *intractable*, in the sense that the number of points can be exponential in the size of the instance [Ehrgott, 2006]. That is why, in addition to the inherent difficulty of MO problems previously mentioned, exact methods finding an exact reduced efficient set are generally used for easy problems or small instances of harder problems. On the other hand, given a predefined tolerance in the dominance relation, approximation methods with performance guarantee will be able to find an approximation of the efficient set hence exact methods become impracticable. These approximation methods seem to be efficient in theory [Papadimitriou and Yannakakis, 2000] as in practice for certain problems [Bazgan et al., 2009a]. Finally, meta-heuristics do not provide any guarantee of performance on the quality of the approximation set found, but due to their practical effectiveness, they are currently massively used to optimize (large instances of) hard problems. Indeed, single-objective meta-heuristics have been adapted to MOCO, such as MO Evolutionary Algorithms [Coello et al., 2007], MO local search [Ulungu and Teghem, 1992, Hansen, 1997, Talbi et al., 2001, Paquete et al., 2004, Vianna and Arroyo, 2004], MO nature-inspired meta-heuristics [Barán and Schaerer, 2003, López-Ibáñez and Stützle, 2012, Xue et al., 2013], and more recently MO Monte Carlo Search [Wang and Sebag, 2012]. Once (an approximation of) the efficient set has been found, the decision maker chooses the solution which fits the best to his/her preferences using Multiple Criteria Decision Aid method [Greco et al., 2005].

Scope of the thesis

In this thesis, we are interested in designing data structures and meta-heuristics for finding approximation of efficient set of MOCO problems. The main point of the thesis is to propose new and efficient (in terms of time and quality) methods, independent of the addressed problem as far as possible, and scalable both in the size and in the number of objectives of the instance. Indeed, we did not limit ourselves to the bi-objective case, and have considered problems with up to 5 objectives. Moreover, the proposed tools are modular, in the sense that they can be used independently from each other. Due to recent successes of local search on hard MOCO problems [Li and Landa-Silva, 2009, Ke et al., 2014, Lust and Tuyttens, 2014], we are particularly concerned by introducing new tools for MO local search, Single-Objective (SO) optimizers and data structures. As a secondary axis, considering recent development of efficient Monte Carlo Search methods

on many SO optimization problems [Browne et al., 2012, Edelkamp et al., 2016], we are also interested in the combination of Monte Carlo Search methods with MO local search.

Outline of the thesis

Chapter 1 recalls fundamental definitions and notions of MOCO. Chapter 2 exposes an overview of methods for optimizing MOCO problems, focusing on meta-heuristics. The chapter also presents a literature review of methods on MO symmetric Traveling Salesman Problem (MOTSP), focusing on meta-heuristics and exposes the state-of-the-art of archives, which are sets of mutually non-dominated solutions. Chapter 3 deals with the new proposed archives: AVL-Archive for bi-objective optimization problems and NDR*-Archive for the general case. The experimental results on artificial and MOTSP benchmarks are presented. Chapter 4 introduces the new tools proposed to optimize MOCO problems: a new MO meta-heuristic called 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D), embedding Adaptive Maximally Dispersed set of Weights (A-MDW) for generating an initial population, Partitioned Pareto Local Search (P-PLS) as a speed-up technique for PLS, and some modifications on SO optimizers to improve their efficiency on MOCO problems. Chapter 5 is devoted to the application of 2PIPLS/D to MOTSP on a benchmark of bi-objective and tri-objective instances, and proposes an empirical evidence of global convexity on MOTSP. Finally, Chapter 6 introduces the MO version of the French Regions Mapping Problem and shows the application of 2PIPLS/D to this new five-objective problem.

Chapter 1

Fundamental concepts of MOCO

This chapter first recalls the formal definition of a MOCO problem, then basic notions like dominance relations, the concept of non-dominance, ideal and nadir points and (non-)supported solutions. Basic techniques are also discussed such as aggregation functions and weighting vector generation. Finally, some classical MOCO problems are introduced.

Introduction

In a MOCO problem, several objectives are taken into account, which often leads to obtain incomparable solutions representing different possible trade-offs between objectives. Section 1.1 first recalls the formal definition of a MOCO problem, then introduces important notions and techniques of MOCO. Section 1.2 presents a number of classical theoretical MOCO problems.

1.1 Fundamental concepts and techniques

1.1.1 Problem setting

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 *multi-objective cost function.*

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

$$\begin{cases} \min f(x) = (f_1(x), ..., f_p(x)) \\ \text{subject to } x \in X \end{cases} \tag{1.1}$$

where $X \subset 2^E = \{0, 1\}^q$ is the *feasible set* and the p potentially conflicting *objective functions* $f_j : X \to \mathbb{R}$ mapping solutions to $\mathbb{R}$, for $j = 1, ..., p$.

Given a feasible solution $x \in X$, an objective function can take several functional forms: usually additive, i.e. $f_j(x) = \sum_{e \in x} c_j(e)$, sometimes bottleneck, i.e. $f_j(x) = \max_{e \in x} c_j(e)$, or even more complex non linear forms.

The vector function $f = (f_1, ..., f_p)$ hence maps solutions to *points* in the *objective space* $Z \subset \mathbb{R}^p$. Let $Y := f(X) \subset Z$ be the *set of all feasible points.*

1.1.2 Dominance relations

Points in the objective space are compared using the concept of Pareto dominance. For any points $z, z' \in Z$, we define the following relations:

- z *weakly dominates* z', denoted by $z \leqq z'$, iff. $z_j \leq z'_j$ for each $j = 1, ..., p$
- z *strictly dominates* z', denoted by $z < z'$, iff. $z_j < z'_j$ for each $j = 1, ..., p$
- z *dominates* z', denoted by $z \leq z'$, iff. $z_j \leq z'_j$ for each $j = 1, ..., p$ and there exists $i \in \{1, ..., p\}$ such that $z_i < z'_i$
- z and z' are incomparable, denoted by $z \parallel z'$, iff. $z \nleqq z'$ and $z' \nleqq z$

We can introduce a tolerance in the above dominance relations by defining an approximate dominance relation. This can be relevant if small differences between objective values are judged non significant. For any points $z, z' \in Z$ and given a tolerance $\epsilon > 0$, we define the following relations:

- z ϵ-dominates z', denoted by $z \leqq_\epsilon z'$, iff. $z_j \leq (1 + \epsilon) z'_j$ for each $j = 1, ..., p$
- z and z' are ϵ-incomparable, denoted by $z \parallel_\epsilon z'$, iff. $z \nleqq_\epsilon z'$ and $z' \nleqq_\epsilon z$

In addition, these original definitions are propagated in the decision space as follows, for any $x, x' \in X$:

- x *weakly dominates* x', denoted by $x \leqq x'$, iff. $f(x) \leqq f(x')$
- x *strictly dominates* x', denoted by $x < x'$, iff. $f(x) < f(x')$
- x *dominates* x', denoted by $x \leq x'$, iff. $f(x) \leq f(x')$
- x *ϵ-dominates* x', denoted by $x \leqq_\epsilon x'$, iff. $f(x) \leqq_\epsilon f(x')$
- x and x' are ϵ-incomparable, denoted by $x \parallel_\epsilon x'$, iff. $f(x) \nleqq_\epsilon f(x')$ and $f(x') \nleqq_\epsilon f(x)$

1.1.3 Non-dominance and efficiency

A point $z \in Y$ is called *non-dominated* if and only if there is no other point $z' \in Y$ such that $z' \leq z$. In other words, a non-dominated point is a point such that no improvement on any objective is possible without sacrificing on at least another objective. 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 $Y_{nd} \subseteq Y$ is called the *non-dominated set* (or *Pareto front*). The set of all efficient solutions $X_e \subseteq X$ is called the *efficient set*.

Given a tolerance $\epsilon > 0$, a set of points $\widetilde{Y}_{nd} \subset Y$ is called an *ϵ-approximation* of the non-dominated set Y_{nd} if any point $z \in Y_{nd}$ is ϵ-dominated by at least one point $z' \in \widetilde{Y}_{nd}$ [Papadimitriou and Yannakakis, 2000]: $\forall z \in Y_{nd}, \exists z' \in \widetilde{Y}_{nd} : z' \leqq_\epsilon z$.

1.1.4 Ideal and nadir points

Let $z^* = (z_1^*, ..., z_p^*) \in Z$ be the *ideal point*, which has the best values for each objective:

$$z_j^* = \min_{x \in X} f_j(x) \text{ for } j = 1, ..., p$$

Let $\eta = (\eta_1, ..., \eta_p) \in Z$ be the *nadir point*, which has the worst values for each objective on the non-dominated set:

$$\eta_j = \max_{x \in X_e} f_j(x) \text{ for } j = 1, ..., p$$

Let $X' \subset X$ be a subset of *incomparable solutions*.

The *local ideal point* $z^*(X') \in Z$ of X' is the point with the best objective values among all solutions belonging to X':

$$z_j^*(X') = \min_{x \in X'} \{f_j(x)\},\ j = 1, ..., p$$

The *local nadir point* $\eta(X') \in Z$ of X' is the point with the worst objective values among all solutions belonging to X':

$$\eta(X') = \max_{x \in X'} \{f_j(x)\},\ j = 1, ..., p$$

Approximations of these extreme points can be introduced.

An *approximate local ideal point* $\widetilde{z}^*(X') \in Z$ of X' is a point which weakly dominates $z^*(X')$, i.e. such that $\widetilde{z}^*(X') \leqq z^*(X')$.

An *approximate local nadir point* $\widetilde{\eta}(X') \in Z$ of X' is a point which is weakly dominated by $\eta(X')$, i.e. such that $\eta(X') \leqq \widetilde{\eta}(X')$.

Naturally, both $z^*(X')$ and $\widetilde{z}^*(X')$ weakly dominate all solutions from X', and all solutions from X' weakly dominate both $\eta(X')$ and $\widetilde{\eta}(X')$.

A *Minimum Bounding Box* (MBB) of a set of spatial objects (resp. points) in Z is a hyper-rectangle of minimum hypervolume including all the objects (resp. points). It is uniquely defined by a pair of *local ideal and nadir points.*

1.1.5 Aggregation functions

An aggregation function aims at aggregating the objectives of a MOCO problem so that the resulting problem is a single-objective problem. Two aggregation functions are used in this document: the *weighted sum* and the *weighted augmented Tchebychev* functions, detailed below.

1.1.5.1 Weighted sum

Let $\lambda = (\lambda_1, ..., \lambda_p) \in \mathbb{R}^p$ such that $\lambda_j \geq 0$ for $j = 1, ..., p$, be a weighting vector (called *weight*). The *weighted sum problem* is given by:

$$\begin{cases} \min \lambda f(x) = \sum_{j=1}^{p} \lambda_j f_j(x) \\ \text{subject to } x \in X \end{cases} \tag{1.2}$$

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

1.1.5.2 Weighted augmented Tchebychev

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

$$\begin{cases} \min \text{wat}(x, \lambda, z^*) = \max_{j=1,...,p} \lambda_j (f_j(x) - z_j^*) + \varepsilon \sum_{j=1}^{p} \lambda_j (f_j(x) - z_j^*) \\ \text{subject to } x \in X \end{cases} \tag{1.3}$$

where $\varepsilon > 0$ is a fixed small positive real.

The *weighted Tchebychev problem* is a weighted augmented Tchebychev problem with $\varepsilon = 0$.

1.1.6 Supported and non-supported solutions

Supported efficient solutions are optimal solutions of a weighted sum problem for some vector $\lambda > 0$ [Geoffrion, 1967]. Let $\text{conv}(Y)$ be the convex hull of the set of all feasible points. The images in the objective space of the supported efficient solutions correspond to the *supported non-dominated points*, which are non-dominated points located on $\text{conv}(Y)$.

The set of all supported efficient solutions is called the *supported efficient set*, and the set of all supported non-dominated points is called the *supported non-dominated set.*

We can make a distinction between supported efficient solutions:

- *Extreme supported efficient solutions*, whose image in the objective space, called *extreme supported non-dominated points*, are non-dominated points located on the vertex set of conv(Y).
- *Non-extreme supported efficient solutions*, whose image in the objective space, called *non-extreme supported non-dominated points*, are not located on the vertex set of conv(Y).

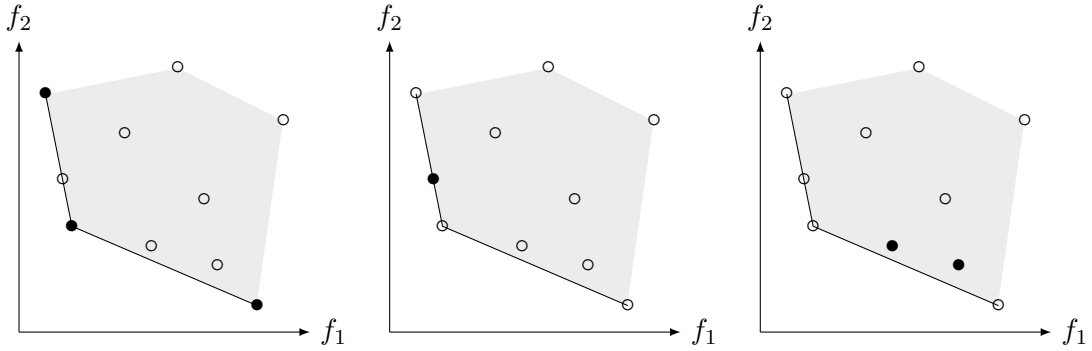


Figure 1.1 – Extreme supported non-dominated points (left part), a non-extreme supported non-dominated point (middle part), non-supported non-dominated points (right part). The gray area represents the convex hull of the set of all feasible points conv(Y).

Non-supported efficient solutions are efficient solutions that are not optimal solutions for any weighted sum problem with $\lambda > 0$. The image in Y of the non-supported efficient solutions are the *non-supported non-dominated points*, located in the interior of conv(Y).

Figure 1.1 summarizes the different types of non-dominated points.

By taking into consideration these different types of solutions, a distinction can be made between weighted sum and weighted augmented Tchebychev functions. Indeed, in contrast with weighted sum, optimizing a weighted augmented Tchebychev function with the appropriate weight allows us to find any efficient solution (not only a supported one) at the expense of a harder problem to solve.

1.1.7 Weight space and generation of weights

A closed subset $\Lambda \subset \mathbb{R}^p$ is called a *d-dimensional simplex* in $\mathbb{R}^p$ ($d \leq p$) if Λ is the convex hull of $d+1$ affinely independent points $\lambda^1, ..., \lambda^{d+1} \in \mathbb{R}^p$. More formally:

$$\Lambda = \{\lambda \in \mathbb{R}^p : \lambda = \sum_{j=1}^{d+1} w_j \lambda^j,\ w_j \in [0,1],\ \sum_{j=1}^{d+1} w_j = 1\}$$

Any point of a simplex can be expressed as a linear combination of the vertices of the simplex. We note $\Lambda = \triangle(\lambda^1, ..., \lambda^{d+1})$ to indicate that $\lambda^1, ..., \lambda^{d+1}$ are the *vertices* of Λ.

Let

$$\Lambda_0 := \{\lambda \in \mathbb{R}^p : \lambda_j \geq 0,\ j = 1, ..., p,\ \sum_{j=1}^{p} \lambda_j = 1\}$$

be a (p-1)-dimensional simplex in $\mathbb{R}^p$, called the *normalized weight space* (or simply the *weight space*). Λ_0 contains all the positive weights normalized to unity. Generally, any weight generated for optimizing a given aggregation function will belong to the normalized weight space Λ_0.

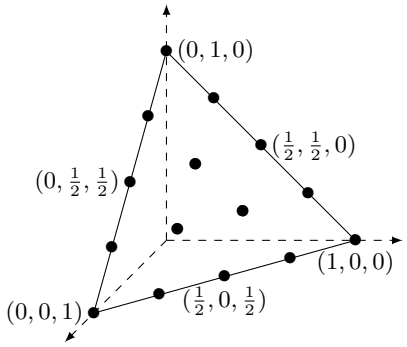


Figure 1.2 – Set of 15 weights (black dots) generated in Λ_0 by MDW with $D = 4$ and $p = 3$. Some vectors are indicated as examples.

A method often used to create weights into Λ_0 is the *Maximally Dispersed set of Weights* (MDW) [Steuer, 1986] which produces a number of equally dispersed weights into Λ_0 (see Figure 1.2). Given a parameter $D \in \mathbb{N}^*$controlling the number of weights generated, MDW provides the set of weights given by:

$$\begin{cases} \text{MDW}(D) := \{\lambda = (\lambda_1, ..., \lambda_p) : \lambda \in \Lambda_0,\ \lambda_j = \frac{d}{D},\ d \in \{0, ..., D\},\ j = 1, ..., p\} \\ \text{such that } \min\limits_{\lambda^a, \lambda^b \in \Lambda_0 :\ \lambda^a \neq \lambda^b} \sum\limits_{j=1}^{p} \left|\lambda_j^a - \lambda_j^b\right| = \frac{2}{D} \end{cases} \tag{1.4}$$

where $|\text{MDW}(D)| = \binom{D+p-1}{p-1}$.

1.2 Complexity, intractability and examples of MOCO problems

The difficulty to generate the efficient set comes essentially from two main issues:

- the *complexity* of the addressed problem;
- the potentially extremely large number of non-dominated points.

Concerning the complexity of a problem, we will simply oppose problems for which the decision version is polynomially solvable in time given the size of the instance; and problems with a NP-hard decision version, for which a conjecture widely accepted states that the problem is, in the worst case, exponentially solvable in time given the size of the instance.

Concerning the second issue, a situation often encountered is that the addressed problem is *intractable*, meaning that the number of non-dominated points can be exponential in the size of the instance.

We can list some classical MOCO problems, among others: MO Minimum Spanning Tree, MO (Quadratic) Assignment Problem, MO (Multidimensional) Knapsack Problem, MO symmetric / asymmetric Traveling Salesman Problem. Let us detail three of them:

MO Assignment Problem In MO Assignment Problem (MOAP), n agents have to be assigned to n machines. Each couple (agent,machine) has a cost vector. Any agent can be assigned to any machine. It is required to assign exactly one agent to each machine and exactly one machine to each agent while minimizing the total cost. More formally, the combinatorial structure of MOAP is represented by a complete bipartite graph $G = (V, E)$ with V the set of vertices, divided into two disjoint sets $A = \{a_1, ..., a_n\}$ and $M = \{m_1, ..., m_n\}$, both of size n; and $E = \{(a_k, m_l) : k, l = 1, ..., n\}$ be the set of edges. MOAP is defined by (1.1), where $f_j(x) = \sum_{e \in x} c_j(e)$ for $j = 1, ..., p$ and X represents the set of all perfect matchings on G. While the decision version of AP is polynomially solvable, the decision version of MOAP is NP-hard [Ehrgott, 2006].

MO 0-1 Knapsack Problem The MO 0-1 Knapsack Problem (MOKP) consists in inserting n items into a knapsack with a limited integer capacity $W > 0$. Each item k has a positive integer weight w^k and p positive integer profits $v_1^k, ..., v_p^k$, $k = 1, ..., n$. A feasible solution is represented by a vector $x = (x_1, ..., x_n)$ of variables $x_k \in \{0, 1\}$, such that $x_k = 1$ if x contains item k and 0 otherwise, while satisfying the capacity constraint $\sum_{k=1}^{n} w^k x_k \leq W$. MOKP is defined by (1.1), where $f_j(x) = \sum_{k=1}^{n} v_j^k x_k$ is the value of a feasible solution $x \in X$ on the j-th objective, $j = 1, ..., p$. Decision versions of both KP and MOKP are NP-hard [Ehrgott, 2006].

MO symmetric Traveling Salesman Problem In symmetric Traveling Salesman Problem (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. A single cost value is associated to each edge between two cities. The goal is to find a tour such that the total cost is minimized.

In MO symmetric Traveling Salesman Problem (MOTSP), a cost vector is associated to each edge between two cities, and the traveling salesman has to minimize all the total costs. More formally, we define 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 (1.1), where $f_j(x) = \sum_{e \in x} c_j(e)$ for $j = 1, ..., p$ and X represents the set of Hamiltonian cycles on G. The decision version of TSP is NP-hard. MOTSP is intractable and its decision version is also NP-hard [Ehrgott, 2006].

Chapter 2

Methods in MOCO

This chapter proposes an overview of the different methods existing in MOCO, by focusing on meta-heuristics. First, the notion of quality indicator for a set of points is introduced and indicators used for our futur experiments are detailed. Then, different classes of meta-heuristics are presented among which local search and Monte Carlo Search are extensively described. In particular, we intend to propose a clear categorization of the different MO local search methods. Finally, literature reviews are proposed concerning optimization methods for MOTSP and archives, which are sets of incomparable solutions.

Introduction

There are many different general methods to deal with MOCO problems, categorized into two classes: exact methods and approximation methods. An exact method aims at finding the whole non-dominated set and providing one corresponding efficient solution for each point found. Numerous different general exact methods exist, like MO Branch-and-bound [Sourd and Spanjaard, 2008, Cerqueus et al., 2017, Przybylski and Gandibleux, 2017], MO Dynamic Programming [Bazgan et al., 2009b], ϵ-constraint [Haimes et al., 1971, Florios and Mavrotas, 2014], two-phase methods [Ulungu and Teghem, 1995, Przybylski et al., 2010b] and recently, search region-based method [Lacour, 2014, Klamroth et al., 2015, Dächert et al., 2017].

Approximation methods aim at finding an approximation of the non-dominated set and memorizing a single solution for each point found. There are two types of approximation methods: those with performance guarantee and those without any performance guarantee, called *heuristics*. *Approximation methods with performance guarantee* (also called *approximation methods with a priori guarantee*) are methods introducing a tolerance in the dominance relation, in general the usual dominance relation $\leq$ is replaced by the epsilon dominance $\leqq_\epsilon$ with a predefined ϵ. By relaxing the dominance relation, such methods are able to find a good approximation of the efficient set in a reasonable time, when exact methods become impracticable. These approximation methods with performance guarantee seem to be efficient in theory [Papadimitriou and Yannakakis, 2000] as in practice for certain problems (see for example [Bazgan et al., 2009a, Lacour, 2014]).

Both exact methods and approximation methods with performance guarantee are used on problems like MO Shortest Path Problem, MO Assignment Problem, MO 0-1 Knapsack Problem, MO Minimum Spanning Tree problem and others. But for harder problems like MOTSP, MO Multidimensional 0-1 Knapsack Problem or MO Quadratic Assignment Problem and many others, it is difficult in practice to obtain a valuable guarantee on the quality of the generated solutions in a reasonable time (see for example [Lust and Teghem, 2012, Florios and Mavrotas, 2014]).

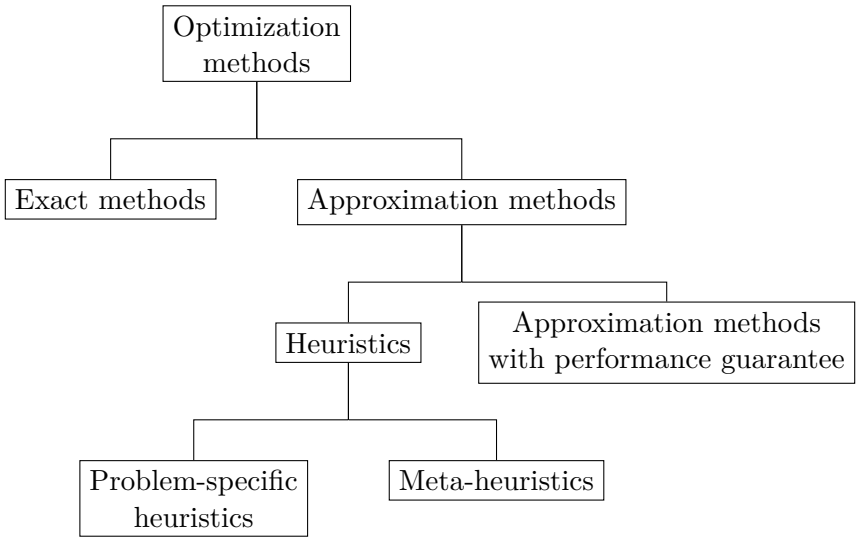


Figure 2.1 – A taxonomy of MOCO optimization methods.

To handle this difficulty, researchers have been interested in developing heuristics. Heuristics are subdivided into two sub-categories: problem-specific heuristics and *meta-heuristics* (see Figure 2.1 summarizing the different types of optimization methods). A meta-heuristic is a general heuristic method able to optimize different combinatorial problems using the same framework. The process of any meta-heuristic is based on two conflicting notions: *intensification* and *diversification* of the search. Intensification (also called *exploitation*) aims at intensifying the search towards promising regions of the search space, while the goal of *diversification*

is to explore as much as possible the search space in order to find new promising regions. In meta-heuristics, randomness is extensively used to provide a certain degree of diversity to the search. A crucial dilemma of meta-heuristics consists in distributing the available computational resources between intensification and diversification.

While meta-heuristics do not provide any guarantee on the quality of the solutions found, they have the advantage of being computationally efficient, general, and relatively simple in implementation. Indeed, they are currently massively used to optimize problems for which exact and approximation methods with performance guarantee are impracticable. Because meta-heuristics only produce approximations of the efficient set, it is necessary to be able to evaluate the quality of any approximation in order to compare different meta-heuristics.

Section 2.1 presents three among the most popular quality indicators used to evaluate the quality of approximation sets generated by the meta-heuristics. Section 2.2 describes several meta-heuristics used in MOCO. Then, Section 2.3 proposes a literature review of methods for MOTSP, focusing on meta-heuristics. Finally, Section 2.4 discusses about the different data structures published these last few years for managing a set of incomparable solutions, called *archives*.

2.1 MO quality indicators

In single-objective optimization, it is quite easy to measure the quality of a solution since it is to be compared with a unique value. It is a more difficult task in the MO case, because MO outputs are represented by sets of trade-off solutions, incomparable in terms of Pareto dominance. Consequently, we use several indicators, called quality indicators, to measure the quality of an approximation of the non-dominated set.

This section presents three of the most used quality indicators to compare approximation sets in MOCO: the *hypervolume difference indicator* I_{H^-} [Zitzler, 1999], the ϵ *indicator* I_ϵ [Zitzler et al., 2003] and the *R2 indicator* I_{R2} [Hansen and Jaszkiewicz, 1998]. We use only the *unary version* of these indicators as they measure the quality of a single set, by contrast with the *binary version* measuring the difference of quality of a couple of sets.

The computation of these indicators implies to know the exact non-dominated set Y_{nd}, which is generally unknown for a given instance of the addressed problem. Thus we approximate it by merging all the approximations generated during the experimental phase and keeping only the non-dominated points, forming the approximation of the non-dominated set $\widetilde{Y}_{nd}$. Let $\widetilde{z}^* \in Z$ be the approximate ideal point defined as the ideal point computed on $\widetilde{Y}_{nd}$.

2.1.1 Hypervolume difference indicator

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 an unary quality indicator 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^- [Zitzler, 1999] (to be minimized). Given an approximation set A and the reference point $\bar{z}$, the indicator value is defined as:

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

$I_H^-(A, \widetilde{Y}_{nd}, \bar{z})$ defines the hypervolume of the subspace that is weakly dominated by $\widetilde{Y}_{nd}$ but not by A (see Figure 2.2). In contrast to the original hypervolume indicator, the lower $I_H^-(A)$, the better the quality of A

is. The computation of hypervolume is time consuming, particularly when the sets are large and the number of objectives is high. Therefore, we utilize two publicly available and efficient algorithms, each one being the best known method for computing hypervolume for a particular range of p. For $p \leq 4$, we use the algorithm of Fonseca et al. [Fonseca et al., 2006, Beume et al., 2009][1]. For $p \geq 5$, we use instead the Walking Fish Group algorithm [While et al., 2012][2].

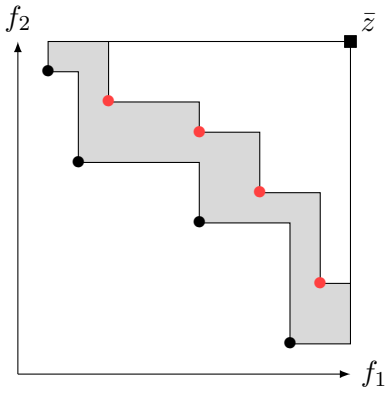


Figure 2.2 – Hypervolume difference I_H^- (gray area) between the approximation of the efficient set $\widetilde{Y}_{nd}$ (black dots) and an approximation set A (red dots) given a reference point $\bar{z}$.

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:

$$z'_j \leftarrow \frac{z_j - z_j^{min}}{z_j^{max} - z_j^{min}} + 1 \quad \text{for } j = 1, ..., p \tag{2.1}$$

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 (2.1), 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. [Fonseca et al., 2006, Beume et al., 2009] 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, $\bar{z} = (2.1, ..., 2.1)$.

2.1.2 Epsilon indicator

Given an approximation set A and the approximation of the non-dominated set $\widetilde{Y}_{nd}$, the ϵ indicator I_ϵ [Zitzler et al., 2003] (to be minimized) gives the smallest factor $\epsilon^* \geq 0$ by which A is worse than $\widetilde{Y}_{nd}$ with respect to all objectives, defined as:

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

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

[1] http://iridia.ulb.ac.be/~manuel/hypervolume
[2] http://www.wfg.csse.uwa.edu.au/hypervolume/index.html#data

2.1.3 R2 indicator

Given an approximation set A and a set of weights Λ, the R2 indicator I_{R2} [Hansen and Jaszkiewicz, 1998] (to be minimized) value of A is defined as:

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

where wat is the weighted augmented Tchebychev function defined in (1.3). The lower $I_{R2}(A, \Lambda, \widetilde{Y}_{nd}, \widetilde{z}^*)$, the better the approximation set A is comparing to $\widetilde{Y}_{nd}$. As indicated in [Fonseca et al., 2005], the set Λ is defined using the MDW method with a parameter D which should be sufficiently large to cover well $\widetilde{Y}_{nd}$. Figure (2.3) illustrates I_{R2}.

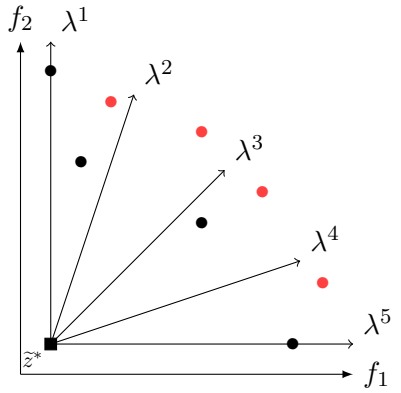


Figure 2.3 – Representation in Z of the set of weights Λ used to compute the I_{R2} value of an approximation set A (red dots), given the approximation of the efficient set $\widetilde{Y}_{nd}$ (black dots) and the approximation of the ideal point $\widetilde{z}^*$ ($p = 2$).

In order to have a number of weights proportional to the size of $\widetilde{Y}_{nd}$, we set $D := \arg\inf\{\binom{D'+p-1}{p-1} \geq \frac{1}{10}|\widetilde{Y}_{nd}| : D' \in \mathbb{N}\}$. As suggested by Fonseca et al. [Fonseca et al., 2005], normalization is not mandatory for this indicator.

For all presented indicators, the value of the approximation of the non-dominated set $\widetilde{Y}_{nd}$ is 0.

2.1.4 Mann-Whitney statistical test

In order to statistically compare the results of the different algorithms which will be tested in this document, the Mann-Whitney non-parametric statistical test [Mann and Whitney, 1947] is 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 I_H^-, I_ϵ, I_{R2}) given an instance, the levels of risk of the tests have been adjusted with the Holm sequential rejective method (see [Holm, 1979] for more details). The starting level of risk of the Mann-Whitney test is fixed to 1%.

2.2 MO meta-heuristics

We can categorize the meta-heuristics into four main classes which are of particular interest in this thesis: nature-inspired meta-heuristics [Barán and Schaerer, 2003, López-Ibáñez and Stützle, 2012, Xue et al., 2013], MO evolutionary algorithms [Coello et al., 2007], MO local search [Ulungu and Teghem, 1992, Hansen, 1997, Talbi et al., 2001, Paquete et al., 2004, Vianna and Arroyo, 2004, Drugan and Thierens, 2010], and MO Monte Carlo Search [Wang and Sebag, 2012]. Most of the presented meta-heuristics are said *population-based*, as they manages simultaneously a set of solutions (called a population) instead of a single solution.

In general, a meta-heuristic maintains a number of *archive*s. An archive is a set of incomparable solutions. In particular, the *global archive* is the best-so-far approximation of the efficient set, i.e. the archive of all incomparable solutions found so far by the meta-heuristic.

This section focuses on meta-heuristics introducing important notions or having inspired the new methods presented in this document. The level of detail allowed for a method depends both on its recency and its relevance with respect to this thesis. We refer the reader to [Ehrgott and Gandibleux, 2004, Ehrgott and Gandibleux, 2008, Talbi, 2009] for overviews of MO meta-heuristics.

2.2.1 Nature-inspired meta-heuristics

Ant Colony Optimization (ACO) [Dorigo et al., 2006] is a population-based *constructive* meta-heuristic, as it iteratively constructs a solution. It is inspired by the natural behavior of real ants which communicate indirectly by means of trails of a chemical substance called pheromone. For optimizing an addressed combinatorial problem, an ACO method controls a number of agents called (artificial) *ants* whose object consists in generating new solutions by following a stochastic construction policy. Each ant starts with an empty solution and then iteratively adds elementary components to the current partial solution until a complete solution is created. The policy uses two types of information: the *pheromone information* reflecting the experience accumulated by previous ants; and the *heuristic information*, depending on the addressed problem. The construction policy is called a *learning policy* in the sense that the pheromone information it uses is regularly updated in function of the solutions found by the ants (this mechanism is called *reinforcement learning*).

Multi-Objective ACO (MOACO) [Barán and Schaerer, 2003] is an extension of ACO for optimizing MOCO problems and works similarly as single-objective ACO. Generally, MOACO methods are *aggregation-based* in the sense that they use aggregation functions to generate new solutions (see [López-Ibáñez and Stützle, 2012] for example). MOACO methods have been applied to different MOCO problems, like MOKP [Alaya et al., 2007], MOTSP [López-Ibáñez and Stützle, 2012], MO Vehicle Routing with Time Windows [Barán and Schaerer, 2003] and large-scale MO Shortest Path [Ghoseiri and Nadjari, 2010].

Another nature-inspired meta-heuristic is the Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1997], which originally mimics the behavior of a swarm of particles or animals, like bird flocks or fish schools. Instead of *constructing* new solutions, PSO explicitly maintains a population and iteratively improves the current solutions, called the *particles* (see [Kennedy, 2011] for more details on PSO). PSO has recently been adapted to MOCO [Xue et al., 2013] but, to our knowledge, is not considered as a leading general method in MOCO.

As rightly pointed out by Sörensen [Sörensen, 2015], many meta-heuristics based on a metaphor of some natural or physical process have emerged in recent years and propose diverse approaches for optimizing a problem. Like Sörensen, we believe that a great majority of these meta-heuristics are only specific variants of existing ones. None of these methods will be developed.

2.2.2 Evolutionary Algorithms

Evolutionary Algorithm (EA) -see [Bäck et al., 1997] for a survey of EA- is a fundamental class of population-based meta-heuristic. In an EA, a solution is called an *individual*. The population of an EA is involved into an evolution process by iteratively generating and eliminating individuals, given a *fitness function* evaluating the generated individuals. Each iteration of a basic EA is composed of three main steps:

1. Selection of parents: a number of individuals (the *parents*) are selected among individuals from the population.
2. Combination of parents: the parents are combined in order to generate new individuals (called *children* or *offspring*) with a good fitness score. The children are inserted into the population.
3. Update of population: a subset of the current population is selected to compose the next population.

There are many different categories of EA, the main one is the Genetic algorithms (GA) [Goldberg, 1989, Holland, 1992] which emulate the principles of natural evolution. In a GA, several parents (often two) are combined through a *crossover operator* which creates a number of feasible solutions (often a single or two) by combining the elements composing the parents. Then a *mutation operator* is applied to each generated child, slightly modifying the individual in order to diversify the search. These operators are called *genetic operators* and are problem-specific.

Another interesting and more recent category of EA is Estimation of Distribution Algorithms (EDA) [Larrañaga and Lozano, 2001]. Instead of using crossover and mutation operators, an EDA samples an explicit probabilistic model (like Bayesian networks) built from a selected subset of the current population.

The adaptation of EA for MOCO problems (MOEA) is quite natural. In general, a MOEA manages two populations:

- a global archive, which is the archive of all incomparable solutions found so far;
- a smaller population containing good quality solutions according to the fitness function used *and* well-distributed in the objective-space. This population is generally essential for generating the offspring at the current iteration.

Depending on the MOEA, different fitness functions are used to compare individuals during parent selection and population update steps. Indeed, the fitness function can be an aggregation function [Murata and Ishibuchi, 1995, Zhang and Li, 2007], or dominance-based [Deb et al., 2000, Zitzler et al., 2001], or even more complex like quality indicator-based [Zitzler and Künzli, 2004]. During the parent selection step, a number of parents are selected from the second population in order to maintain a certain degree of diversity during the combination process.

The research field of MOEA is very active: see [Schwarz and Ocenasek, 2001, Pelikan et al., 2006] for typical Multi-objective EDA (MOEDA) and [Coello et al., 2007, Zhou et al., 2011] for surveys on MOEA. But in this thesis, we are particularly interested in algorithms following the notion of *decomposition* [Murata et al., 2001, Zhang and Li, 2007].

Decomposition method

Murata et al. [Murata et al., 2001] introduced the concept of *cellular structure* for MO Genetic Algorithms. This concept has been generalized and renamed *Decomposition* by Zhang and Li [Zhang and Li, 2007] with their MOEA based on Decomposition (MOEA/D) method. MOEA/D is an aggregation-based MOEA

as it decomposes a MOCO problem into a fixed number of K aggregation functions and optimizes them simultaneously in a collaborative way. Each function is defined by a weight λ giving a unique search direction, and maintains a best-so-far solution (called an *incumbent*) according to the corresponding chosen aggregation function $F(\cdot, \lambda)$ (e.g. $\lambda f(\cdot)$, $\text{wat}(\cdot, \lambda, z^*)$ or others) for the entire duration of the run. The weights defining the directions are well-dispersed in the weight space as they are generated through MDW.

Algorithme 1 : `MOEA/D`

Input : *stopping criterion*, number of sub-problems K, weight neighborhood size T
Output : global archive $\overline{X}_{all}$

1 $\overline{X}_{all} \leftarrow \emptyset$
2 $(\lambda^1, ..., \lambda^K) \leftarrow$`MDW(`$\kappa$`)` such that $K = \binom{\kappa+p-1}{p-1}$
3 **for** $k \leftarrow 1, ..., K$ **do**
4 $N(\lambda^k) \leftarrow T$ nearest neighbors of λ^k from $(\lambda^1, ..., \lambda^K)$
5 $(x^1, ..., x^K) \leftarrow$`Initialization(`$K$`)`
6 **repeat**
7 **for** $k \leftarrow 1, ..., K$ **do**
8 $\lambda^i, \lambda^j \leftarrow$`RandomSelection(`$N(\lambda^k)$`)`
9 $\widehat{x}^k \leftarrow$`Crossover(`x^i`,` x^j`)`
10 $\widehat{x}^k \leftarrow$`Mutation(`$\widehat{x}^k$`)`
11 $\widehat{x}^k \leftarrow$`Improvement(`$\widehat{x}^k$`)`
12 **for** $l \leftarrow 1, ..., K$ **do**
13 **if** $F(\widehat{x}^k, \lambda^l) < F(x^l, \lambda^l)$ **then**
14 $x^l \leftarrow \widehat{x}^k$
15 `Add(`$\widehat{x}^k$`,` $\overline{X}_{all}$`)`
16 **until** stopping criterion *is met*;

Let $\lambda^k \in \mathbb{R}^p$ be a weight, $x^k \in X$ be the incumbent related to $F(\cdot, \lambda^k)$, and $\pi^k = (\lambda^k, x^k)$ be a pair denoting the k^{th} *sub-problem*, for $k = 1, ..., K$. The neighborhood of the k^{th} sub-problem consists of all the sub-problems with a weight among the T nearest neighbors of λ^k, where T is a fixed parameter. Collaboration between sub-problems is performed by optimizing neighboring sub-problems of π^k to generate new solutions for π^k, for $k = 1, ..., K$. The major motivation of MOEA/D is that collaboration between close sub-problems should improve the global optimization process.

Algorithm 1 details MOEA/D. The method maintains two populations: the global archive $\overline{X}_{all}$ and the set of incumbents $(x^1, ..., x^K)$. First, sub-problems and corresponding neighborhoods are generated (lines 2-5). Weights are generated via MDW, and the incumbents are initialized randomly or with a problem-specific heuristic. Then, starts the iterative phase (lines 6-16). At a given iteration, each sub-problem π^k is considered for optimization (lines 8-11), for $k = 1, ..., K$. Two neighboring sub-problems of π^k are randomly selected and their corresponding incumbents are recombined. The resulting offspring is mutated then improved with a problem-specific improvement heuristic, like local search (see next Section 2.2.3) for example. Because mutation may generate an unfeasible offspring, a repair heuristic may be called before improvement (or replace simply it) in order to make feasible the offspring. The improved generated individual will replace any worse incumbent x^l on the aggregated problem $F(\cdot, \lambda^l)$, for each $l = 1, ..., K$ (lines 12-14).

The `Add` procedure (Algorithm 2) is a crucial method shared by all MOCO optimization methods. It aims at presenting a new candidate solution (first parameter) to an archive (second parameter) and updating accordingly the archive. It returns true if and only if the candidate is accepted. If this candidate is weakly

Algorithme 2 : `Add`

Input : solution x, archive $\overline{X}$
Output : boolean

```
1 if ∄x' ∈ X̄ : f(x') ≦ f(x) then
2 |   X̄_dom ← {x'' ∈ X̄ : f(x) ≤ f(x'')}
3 |   X̄ ← X̄ \ X̄_dom
4 |   X̄ ← X̄ ∪ {x}
5 |   return true
6 else
7 |   return false
```

dominated by any solution in the archive, the candidate is *rejected*; otherwise the candidate is *accepted*: all solutions dominated by the candidate are removed from the archive, then the candidate is inserted. Instead of the usual dominance relation $\leq$, one can use the ϵ dominance $\leqq_\epsilon$ given a predefined ϵ. Different algorithms with different theoretical guarantee and practical efficiency exist for processing the `Add` procedure (see Section 2.4 for existing algorithms and Chapter 3 for new algorithms).

A similar procedure is commonly used, the `AddAll` procedure. It consists in presenting one by one each solution of a solution set (first parameter) to an archive (second parameter) and updating accordingly the archive. It returns the updated archive.

Despite its simplicity, MOEA/D generally obtains better results on MOCO problems than the most widely used MOEA called NSGAII (see [Li and Zhang, 2009]) and has been regularly improved (see [Chen et al., 2009, Zhao et al., 2012]); in particular, a recent modification proposes an adaptive adjustment of the weights in order to better fit to the shape of complex non-dominated sets [Qi et al., 2014]. MOEA/D has been applied to different MOCO problems but does not produce sufficiently good results to be considered as a state-of-the-art MOCO optimization method [Chang et al., 2008, Ke et al., 2013]. By contrast, our concern about MOEA/D is its ability to obtain a well-diversified population thanks to the sub-problems mechanism.

2.2.3 Local search

Local search (LS) is a fundamental optimization methodology and is nowadays a key component of many state-of-the-art meta-heuristics for optimizing MOCO. Local search algorithms aim at iteratively modifying a solution (or a set of solutions) in order to improve it. In this section, we first introduce basic notions of LS, then present some well-known LS methods, and finally focus on LS methods used in this thesis.

2.2.3.1 Basic notions of LS

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

Given a (single-objective or MO) combinatorial problem, a fitness function $F : X \to \mathbb{R}$ to be minimized and a neighborhood structure $\mathcal{N}_k$, a *local search descent* explores, at each step, the neighborhood of the current solution $x \in X$ so as to find a neighbor $x' \in \mathcal{N}_k(x)$ such that $F(x') < F(x)$. There exist two main *exploration neighborhood strategies* of the neighborhood structure $\mathcal{N}_k$ from the current solution:

- either the exploration of the neighborhood is stopped at the first neighbor improving the fitness (*first improvement* strategy);
- or the neighborhood is completely explored (*best improvement* or *complete exploration* strategy).

At the end, a LS descent stops in a *local optimum* w.r.t. $\mathcal{N}_k$, for which no improving neighbor can be found.

A Variable Neighborhood Descent (VND) [Brimberg et al., 2000] is a LS descent which explores different neighborhood structures in a deterministic order. We consider the simplest form of VND which explores a limited number of neighborhood $\mathcal{N}_j$ of increasing size $j = 1, 2, ..., k$ (s.t. initially $j = 1$). Globally, the idea of VND is to explore like LS descent the neighborhood of the current solution with the neighborhood of smallest possible size while an improving solution is found. Once we are stuck into a local optimum according to the current neighborhood structure, we increase its size to escape from the local optimum. More precisely, at each step of the descent, once a neighborhood $\mathcal{N}_j$ has been explored from a current solution $x \in X$:

- if an improving neighbor $x' \in \mathcal{N}_j(x)$ has been found, then VND switches to x' and reset j to 1.
- otherwise, VND increases the size j of the neighborhood. If j exceeds k, then the method terminates.

Upon termination, the current solution is locally optimal w.r.t. all neighborhoods $\mathcal{N}_j$, $j = 1, ..., k$. The key idea of the VND presented above is to prioritize the search of an improvement into the smallest possible neighborhood.

S*tochastic local search* (SLS) [Hoos and Stützle, 2004] is a general concept of local search algorithm restarting the local search descent by use of a stochastic process.

A *perturbation move* (also 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 perturbation 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 induced by $\mathcal{N}_k$ from x.

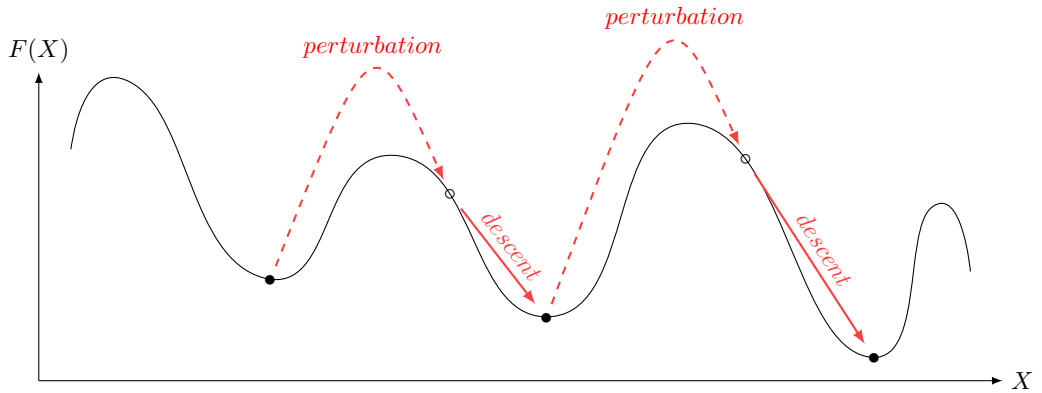


Figure 2.4 – An example of ILS run minimizing a fitness function $F : X \to \mathbb{R}$ on the feasible set X.

An *Iterated Local Search* (ILS) algorithm [Lourenço et al., 2003] is a SLS. It builds a *sequence* of locally optimal solutions by iteratively applying a perturbation to the current locally optimal solution and restarting a local search descent from this modified solution. Figure 2.4 illustrates the mechanism of ILS.

Let us now present a number of important LS meta-heuristics and their adaptation to MOCO.

2.2.3.2 Simulated Annealing

Simulated Annealing (SA) [Kirkpatrick et al., 1983] is a meta-heuristic inspired from a technique consisting in cooling a material down at a controlled rate, altering its physical properties and making it more resistant at the end of the process. Given a fitness to optimize and a neighborhood function, SA originally manages a single solution. At each iteration, the method randomly selects a neighbor of the current solution, then decides to move to the neighbor by following a stochastic acceptance rule (generally the Metropolis-Hastings rule [Hastings, 1970]). This rule encourages the quality improvement between the current solution and the neighbor, and makes acceptance of a worse neighbor harder over time with a mechanism called *temperature cooling*.

Different MO versions of SA have emerged. To our knowledge, all versions use an aggregation function as fitness function, the weighted-sum in a great majority of cases. As MO methods, they manage an archive. The main difference between the versions comes from the choice of the weight(s) used for aggregating the objectives and providing direction(s) to the search.

Ulungu and Teghem's MOSA [Ulungu and Teghem, 1992] and Serafini's MOSA [Serafini, 1994] follow the same scheme than the original SA. In order to approximate an efficient set, both methods are applied several times with different weights. By contrast, PSA [Czyżżak and Jaszkiewicz, 1998] and EMOSA [Li and Landa-Silva, 2011] are population-based methods as they manage simultaneously a number of weights and their respective incumbents.

In particular, EMOSA proposes a very similar framework as MOEA/D, the main differences come from the fact that genetic operators are replaced by SA and weights may be replaced by similar ones under certain conditions, in order to diversify the search directions during the run. EMOSA has been compared to Ulungu and Teghem's MOSA, Serafini's MOSA and PSA and obtains better results on the MOKP and MOTSP tested instances.

2.2.3.3 Tabu Search

Tabu Search (TS) [Glover, 1989] is a meta-heuristic that escapes from local optima by forbidding (moves to) solutions recently visited. These moves or solutions are memorized into a memory structure called *tabu list*. Given a fitness to optimize and a neighborhood function, TS originally manages a single solution. At each iteration, the best non-tabu neighbor of the current solution is selected, then the tabu list is updated with the move/solution, finally the method moves to the neighbor. Different data structures can be used to memorize information in the tabu list. If the addressed problem instance has a large decision space and fast look-ups are required, one can use a hash table with Zobrist [Zobrist, 1970] or Woodruff and Zemel [Woodruff and Zemel, 1993] hashing functions.

Concerning the adaptation of TS to MOCO, different methods propose a general framework: MOTS [Gandibleux et al., 1997, Hansen, 1997], TAMOCO [Hansen, 2000] and MOTAS [Loukil et al., 2005]. To our knowledge, all versions use the (augmented) weighted Tchebychev function as fitness function and apply a number of TS runs with different weights.

2.2.3.4 Greedy Randomized Adaptive Search Procedure

Greedy Randomized Adaptive Search Procedure (GRASP) [Feo and Resende, 1989] is an iterative meta-heuristic combining two algorithmic aspects. Let $F : X \to \mathbb{R}$ be a fitness optimized by GRASP. At each iteration, GRASP first constructs an initial solution through a randomized constructive procedure, then conducts from this solution a local search descent using a neighborhood function, finally updates the best-so-far solution given F. At each step of the construction procedure, an element is selected uniformly at random from a candidate list, then added to the partial solution.

About the adaptation to MOCO, the research field of MO GRASP is pretty active and proposes different frameworks. To our knowledge, the first adaptation of GRASP to MOCO is proposed by Gandibleux et al. [Gandibleux et al., 1998b]. Recently, in [Martí et al., 2015], the authors made a survey and a classification of MO GRASP. A large majority follow the same scheme as the original GRASP and use the weighted-sum as fitness function. Like in SA, a number of MO GRASP (such as [Vianna and Arroyo, 2004]) first generate several weights, then optimize the related aggregation functions with GRASP. Other MO GRASP algorithms (such as [Li and Landa-Silva, 2009]) are population-based as they manage simultaneously a number of weights, optimize the related aggregation functions and memorize their respective incumbents.

2.2.3.5 Pareto Local Search

Pareto Local Search (PLS) [Talbi et al., 2001, Paquete et al., 2004] is the MO extension of LS descent. Given a neighborhood function $\mathcal{N}_k$, PLS is an iterative population-based meta-heuristic starting from a set of solutions $\overline{X}$. At each iteration, a number of unexplored solutions are selected from $\overline{X}$ then extracted and transferred into a temporary set, finally for each selected solution $x \in \overline{X}$, $\mathcal{N}_k(x)$ is explored such that all neighbors not dominated by any solution found so far are accepted and inserted into $\overline{X}$. PLS stops once all solutions from $\overline{X}$ have been visited and is stuck in a *locally efficient set* with respect to $\mathcal{N}_k$, i.e. a set of solutions such that all neighbors are weakly dominated by at least one solution found so far.

Two similar versions of PLS have emerged: the so-called *Talbi et al. PLS* [Talbi et al., 2001] and *Paquete et al. PLS* [Paquete et al., 2004]. Note that related approaches to Paquete et al. PLS can be originally found in [Hamacher and Ruhe, 1994, Andersen et al., 1996] for the bi-objective Spanning Tree Problem only, in [Gandibleux et al., 2001] where only a single iteration is performed and employed for bi-objective Permutation Scheduling and 0-1 Knapsack Problems, and finally in [Ehrgott and Gandibleux, 2004], where a more general framework is detailed. In [Liefooghe et al., 2012], the authors propose a common framework and identify two important components in PLS:

- the *selection strategy*, defining how many solutions are selected (then extracted) from $\overline{X}$
- the *exploration neighborhood strategy*, defining how the neighborhood $\mathcal{N}_k(x)$ of a selected solution x is explored (as in single-objective LS descent)

Two selection strategies have been proposed: either a single solution is extracted from $\overline{X}$ (Paquete et al. PLS), or all solutions from $\overline{X}$ are extracted (Talbi et al. PLS).

Algorithme 3 : `PLS`

Input : neighborhood structure $\mathcal{N}_k$, boolean *first-dominating*, set of solutions to explore $\overline{X}$, global archive $\overline{X}_{all}$
Output : $\overline{X}_{all}$

```
1 X̄_all ←AddAll( X̄, X̄_all)
2 while X̄ ≠ ∅ do
3 |   X̄_new ←PLS-iteration( N_k, first-dominating, X̄, X̄_all)
4 |   X̄ ← X̄_new
5 return X̄_all
```

There is no major differences in terms of quality of approximation obtained by these two different strategies, as experimentally shown by [Liefooghe et al., 2012] on MOTSP and MO Permutation Flowshop Scheduling Problem, and by [Lust and Teghem, 2010] on bi-objective TSP. Anyway, it is important to remark that any solution stored in the temporary set at a given iteration is still explored even if a newly generated

solution dominates it. We say that solutions stored in the temporary set are *protected*. Consequently, in Paquete et al. PLS, only the currently explored solution is protected, thus any other dominated solution is immediately deleted and thus, never explored. On the other hand, Talbi et al. PLS enables to protect much more solutions. Therefore, if we add the option to Talbi et al. PLS not to explore dominated solutions stored in the temporary set, as we will propose later in Section 4.3, then we can consider that Talbi et al. PLS is more general than Paquete et al. PLS.

For all these reasons, and to have a unique coherent framework, we consider only the Talbi et al. PLS, consisting in selecting all solutions from $\overline{X}$ at each iteration.

Algorithme 4 : `PLS-iteration [vanilla]`

Input : neighborhood structure $\mathcal{N}.$, boolean *first-dominating*, set of solutions to explore $\overline{X}$, global archive $\overline{X}_{all}$

Output : archive of new solutions $\overline{X}_{new}$

```
1 X̄_new ← ∅
2 foreach x ∈ X̄ do
3     foreach x' ∈ N.(x) do
4         if f(x) ⊈ f(x') then
5             if Add(x', X̄_all) then
6                 Add(x', X̄_new)
7             if first-dominating and f(x') ≤ f(x) then
8                 break
9 return X̄_new
```

Concerning the exploration neighborhood strategy, both [Liefooghe et al., 2012] and [Drugan and Thierens, 2012] propose two interesting strategies to explore $\mathcal{N}_k(x)$ of a current solution x:

- either the exploration of $\mathcal{N}_k(x)$ is stopped at the first neighbor dominating x (*first dominating neighbor* strategy);
- or $\mathcal{N}_k(x)$ is completely explored (*complete exploration* strategy).

Note that two other exploration strategies have been proposed in [Liefooghe et al., 2012] and [Drugan and Thierens, 2012] but Drugan and Thierens [Drugan and Thierens, 2012] proved that these strategies stop prematurely, thus we do not consider them.

These two studies have concluded that the first dominating neighbor strategy is more efficient than the complete exploration strategy when the computational resources (time or number of iterations) are limited.

Algorithm 3 describes PLS and Algorithm 4 details the core of a PLS iteration. PLS takes as input parameters a neighborhood structure $\mathcal{N}_k$, a boolean *first-dominating* fixed to true if and only if the first dominating neighbor strategy is chosen, a set of solutions $\overline{X}$ to explore; finally the *global archive* $\overline{X}_{all}$, memorizing all incomparable solutions found so far. PLS returns $\overline{X}_{all}$.

At each iteration (Algorithm 4), PLS explores the neighborhood of each solution $x \in \overline{X}$, and retains in an auxiliary archive $\overline{X}_{new}$ all the mutually incomparable neighbors of x not weakly dominated by any solution memorized in $\overline{X}_{all}$. If the *first dominating neighbor strategy* is activated, and once a current solution x is dominated by its neighbor, then the neighborhood exploration of x stops. Finally (cf. Algorithm 3), solutions from $\overline{X}_{new}$ are transferred into $\overline{X}$.

PLS continues this process until no more new non-weakly dominated neighbors have been found.

Algorithme 5 : `PLS-iteration [with local archive]`

Input : neighborhood structure $\mathcal{N}.$, boolean *first-dominating*, set of solutions to explore $\overline{X}$, global archive $\overline{X}_{all}$

Output : archive of new solutions $\overline{X}_{new}$

```
1  X̄_new ← ∅
2  foreach x ∈ X̄ do
3      X̄_local ← ∅
4      foreach x' ∈ N.(x) do
5          if f(x) ⊈ f(x') then
6              Add(x', X̄_local)
7              if first-dominating and f(x') ≤ f(x) then
8                  break
9      foreach x'' ∈ X̄_local do
10         if Add(x'', X̄_all) then
11             Add(x', X̄_new)
12 return X̄_new
```

Some remarks can be made on this algorithm. First, the use of the auxiliary set $\overline{X}_{new}$ prevents the exploration of the neighborhood of an already visited solution. Second, the generated solutions (thus the final set $\overline{X}_{all}$) do not depend on the order according to which the solutions of $\overline{X}$ are examined. Finally, passing $\overline{X}_{all}$ as input parameter allows PLS to consider as global archive an external archive from a higher level method using PLS as a component.

In [Jaszkiewicz and Lust, 2016], the authors propose to slightly modify the original PLS by using an additional archive, called here *local archive* (Algorithm 3 using Algorithm 5 at each iteration). In fact, instead of directly presenting each generated solution to the global archive $\overline{X}_{all}$ (which can be computationally costly as $\overline{X}_{all}$ might be large), they use a local archive $\overline{X}_{local}$ (generally much smaller) which memorizes only the mutually incomparable neighbors of the current solution x. Like in the original version, if the *first dominating neighbor strategy* is activated, and once a current solution x is dominated by its neighbor, then the neighborhood exploration of x is stopped. Then, all solutions from $\overline{X}_{local}$ are presented to the auxiliary set $\overline{X}_{new}$.

It is important to note that both sub-versions of PLS (with or without local archive) generate the same solutions and thus provide the same final set $\overline{X}_{all}$.

PLS has been applied to different problems like MOTSP [Paquete et al., 2004, Liefooghe et al., 2012, Dubois-Lacoste et al., 2012], MO Flowshop Scheduling Problem (MOFSP) [Liefooghe et al., 2012], MO Quadratic Assignment Problem (MOQAP) [Paquete and Stützle, 2009a]. In [Dubois-Lacoste et al., 2015], the authors propose additional algorithmic components for PLS in order to enhance its anytime behavior.

In [Lust and Tuyttens, 2014], the authors propose a PLS applying the concept of VND (Algorithm 6), and we call it *PLS-VND*. PLS-VND has the same parameters than PLS except that it uses a number $k \geq 1$ of distinct neighborhood structures $\mathcal{N}_1, ..., \mathcal{N}_k$. Globally, the idea of PLS-VND is to explore like PLS the neighborhood of the current set with the neighborhood structure of smallest possible size while new solutions are found. Once we are stuck into a locally efficient set with respect to the current neighborhood structure, we increase its size to escape from the attraction basin.

Algorithme 6 : `PLS-VND`

Input : set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$, boolean *first-dominating*, set of solutions to explore $\overline{X}$, global archive $\overline{X}_{all}$

Output : $\overline{X}_{all}$

```
1  X̄_all ← AddAll( X̄, X̄_all)
2  j ← 1
3  while j ≤ k do
4      while X̄ ≠ ∅ do
5          X̄_new ← PLS-iteration( N_j, first-dominating, X̄, X̄_all)
6          foreach x ∈ X̄ do j(x) = j;
7          if X̄_new ≠ ∅ then
8              j ← 1
9          X̄ ← X̄_new
10     j ← j + 1
11     X̄ ← {x ∈ X̄_all : j(x) < j}
12 return X̄_all
```

More precisely, a label $j(x) \geq 1$ is associated to each $x \in \overline{X}_{all}$ and gives the maximal size of the neighborhood that has been explored from x. This label avoids to explore the neighborhood of a solution if it has already been explored before. Let j be the neighborhood structure size currently considered, initially set to 1.

At each iteration of PLS-VND, the neighborhood of each solution from $\overline{X}$ is explored using $\mathcal{N}_j$, and $\overline{X}_{all}$ is updated with all newly found solutions memorized into $\overline{X}_{new}$ (lines 5-6). Then:

- If some new solutions are effectively found ($\overline{X}_{new} \neq \emptyset$), then we aim at exploring their neighborhood. Besides, a solution locally optimal for $\mathcal{N}_j$ is not necessarily locally optimal for $\mathcal{N}_i$, $i < j$. Thus we reset the neighborhood structure to $\mathcal{N}_1$, set the current set $\overline{X}$ to $\overline{X}_{new}$, and continue the exploration with $\overline{X}$ (lines 7-9).
- Otherwise $\overline{X}_{new} = \emptyset$, so all solutions from $\overline{X}_{all}$ are locally optimal according to $\mathcal{N}_j$. Thus the neighborhood structure size j currently considered is increased (line 10) and $\overline{X}$ is set with all solutions from $\overline{X}_{all}$ for which the neighborhood has not already been explored with $\mathcal{N}_j, \mathcal{N}_{j+1}, ..., \mathcal{N}_k$ (line 11).

PLS-VND stops once the neighborhood of all solutions from $\overline{X}_{all}$ has been explored with $\mathcal{N}_1, ..., \mathcal{N}_k$. Obviously, PLS-VND is a generalization of PLS such that PLS corresponds to PLS-VND with a single neighborhood structure. It is quite recent and to our knowledge, it has only been applied to the MO Set Covering Problem by Lust and Tuyttens [Lust and Tuyttens, 2014].

The generation process of the initial set of solutions to explore is crucial for PLS. Indeed, as experimentally shown in [Lust and Teghem, 2010] and [Dubois-Lacoste et al., 2011a], with an initial set of poor quality, such as a number of randomly generated solutions (like in [Paquete et al., 2004, Drugan and Thierens, 2010, Liefooghe et al., 2012] among others), PLS converges slowly towards the non-dominated set and provides poor results. The next section discusses about a crucial notion to improve PLS.

2.2.3.6 2-Phase Pareto Local Search

Originally, the exact *2-phase method* [Ulungu and Teghem, 1995, Przybylski et al., 2010b] is an exact algorithm with the following principle: phase one finds the extreme supported non-dominated points, and phase

two the non-extreme and non-supported ones.

The key idea of the first phase is that any supported non-dominated point can be found by solving a weighted sum problem with an appropriate weight. Thus the first phase consists in generating a number of *appropriate* weights then solving the related weighted sum problems with a single-objective exact solver. When the single-objective version of the addressed MOCO is polynomially solvable, like Assignment Problem or Shortest Path Problem for example, this task becomes easy. The solver optimizing each weighted sum problem is problem-dependent, contrary to the weight generation procedure. Initially designed for the bi-objective case [Cohon, 1978, Aneja and Nair, 1979], different methods to generate the extreme supported non-dominated points are proposed nowadays [Przybylski et al., 2010a, Özpeynirci and Köksalan, 2010, Bökler and Mutzel, 2015]. Except in the bi-objective case, all these methods remain complex to implement and difficult to use in practice.

The second phase is problem-dependent and computationally harder because single-objective exact solvers are no longer sufficient to find non-supported non-dominated points, without adding additional constraints to the original problem, like, for example, the ϵ-constraint [Haimes et al., 1971] method does.

Originally based on their previous work [Gandibleux et al., 1998a], Gandibleux et al. [Gandibleux et al., 2001] have been the first, to our knowledge, to propose a hybrid 2-phase method mixing exact and approximation methodologies for bi-objective problems. In particular, the 2-phase framework they proposed first finds either the exact or an approximation of a reduced supported efficient solution set by following the dichotomic scheme [Cohon, 1978, Aneja and Nair, 1979], then applies in second phase a memetic algorithm employing a single iteration of PLS. They tested their method on bi-objective Permutation Scheduling and 0-1 Knapsack Problems.

2-Phase PLS is a heuristic adaptation of the exact 2-phase method and introduced by two different groups of authors: Lust and Teghem [Lust and Teghem, 2010] with 2PPLS ("2-Phase PLS"), and Dubois et al. [Dubois-Lacoste et al., 2011a] with TP+PLS ("Two-Phase PLS") Both methods globally follow the same scheme:

- The first phase generates a number of well-diversified weights, then optimize the corresponding weighted sum problems through an effective single-objective problem-specific heuristic, in order to quickly approximate the non-dominated supported set. The diversification of the weights (in the weight space) is fundamental as it allows the generation of an approximation covering well the non-dominated set. In such a case, we say that the generated points are well dispersed *along* the non-dominated set. Besides, the efficiency of the selected solver (often problem-specific) is mandatory to generate good quality points *towards* the non-dominated set.

- The second phase refines the approximation of the non-dominated set with PLS(-VND).

In 2-Phase PLS, the first phase uses much simpler weight generation procedures than in the exact case. Originally, [Lust and Teghem, 2010] uses an approximation version of the dichotomic scheme [Cohon, 1978, Aneja and Nair, 1979] for the bi-objective case, but nowadays MDW is preferred as it works for any number of objectives.

In [Lust and Teghem, 2010] and [Dubois-Lacoste et al., 2011a], the authors show that the speed convergence *and* the quality result of PLS is greatly improved by initializing the starting set of PLS with such a good quality approximation of the non-dominated supported set. 2-Phase PLS rapidly became state-of-the-art methods on different and hard MOCO problems such as MOTSP [Lust and Teghem, 2010], MO Multi-dimensional KP [Lust and Teghem, 2012], MO Flowshop Scheduling Problem (MOFSP) [Dubois-Lacoste et al., 2011a], MO Set Covering Problem [Lust and Tuyttens, 2014]. The anytime behavior of TP+PLS is studied in [Dubois-Lacoste et al., 2014].

Algorithme 7 : `2-Phase PLS`

Input : *weight generation strategy*, *SO optimizer*, set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$, boolean *first-dominating*

Output : global archive $\overline{X}_{all}$

`// Phase 1:`
$\overline{X}_{init} \leftarrow$`Initialization(`*weight generation strategy, SO optimizer*`)`
$\overline{X}_{all} \leftarrow \emptyset$
`// Phase 2:`
$\overline{X}_{all} \leftarrow$`PLS-VND(`$(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating*, $\overline{X}_{init}$, $\overline{X}_{all}$`)`
return $\overline{X}_{all}$

Algorithm 7 describes the general framework of 2-Phase PLS. The method takes as input parameters: a weight generation strategy (generally MDW) to generate the weights during the first phase, a single-objective solver to optimize the weighted-sum problems produced by the first phase, a set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$ and finally the usual possibility of first-dominating strategy for PLS(-VND).

2.2.3.7 Iterated Pareto Local Search

A large disadvantage of PLS(-VND) (and 2-Phase PLS) is that it always ends up by being stuck into a locally efficient set. To prevent this drawback and allow PLS(-VND) to escape from a locally efficient set, ILS for MOCO called *Iterated PLS* (IPLS) [Drugan and Thierens, 2010] have been developed. At each iteration, IPLS first forms a new set of perturbed solutions from the best-so-far approximation set, then performs PLS(-VND) from this set.

Algorithme 8 : `IPLS`

Input : *stopping criterion*, set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$, boolean *first-dominating*, boolean *independent-pls*

Output : global archive $\overline{X}_{all}$

$\overline{X}_{init} \leftarrow$`Initialization()`
$\overline{X}_{all} \leftarrow \emptyset$
$\overline{X}_{all} \leftarrow$`PLS-VND(` $(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating*, $\overline{X}_{init}$, $\overline{X}_{all}$`)`
repeat
  $\overline{X}_{select} \leftarrow$`SubsetSelection(` $\overline{X}_{all}$`)`
  $\overline{X}_{perturb} \leftarrow$`SubsetPerturbation(` $\overline{X}_{select}$`)`
  if *independent-pls* **then**
    $\overline{X}_{new} \leftarrow$`PLS-VND(` $(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating*, $\overline{X}_{perturb}$, $\emptyset$`)`
  else
    $\overline{X}_{new} \leftarrow$`PLS-VND(` $(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating*, $\overline{X}_{perturb}$, $\overline{X}_{all}$`)`
  $\overline{X}_{all} \leftarrow$`AddAll(` $\overline{X}_{new}$, $\overline{X}_{all}$`)`
until stopping criterion *is met*;
return $\overline{X}_{all}$

We propose a general framework of IPLS depicted in Algorithm 8. IPLS has the same parameters as PLS-VND with the addition of a stopping criterion (a maximum time or number of iterations for example) and a boolean *independent-pls* set to true if and only if PLS-VND is in *independent mod*.

First, IPLS builds an initial set of solutions $\overline{X}_{init}$. PLS-VND is applied on $\overline{X}_{init}$ and initializes the global archive $\overline{X}_{all}$. Then at each iteration:

- a subset $\overline{X}_{select}$ of solutions is selected from $\overline{X}_{all}$;
- a number of solutions from $\overline{X}_{select}$ are perturbed (with for example, genetic operators or LS perturbation moves), then inserted into the set $\overline{X}_{perturb}$;
- a PLS-VND is conducted from $\overline{X}_{perturb}$, and $\overline{X}_{all}$ is updated with all newly generated solutions memorized in $\overline{X}_{new}$. In addition to the first dominating strategy, PLS-VND offers the possibility to be in independent mod or not. If independent mod is activated, PLS-VND ignores the global archive $\overline{X}_{all}$ by not passing it as input parameter (line 8). Otherwise, $\overline{X}_{all}$ is passed as input parameter. When independent mod is activated, PLS-VND explores a largest portion of the decision space, at the cost of a greater computational effort than if independent mod is deactivated.

IPLS stops once the stopping criterion is met and finally returns $\overline{X}_{all}$.

To our knowledge, Drugan and Thierens [Drugan and Thierens, 2010, Drugan and Thierens, 2012] first propose an IPLS and test different versions of the method on MOQAP. In their IPLS, the initialization consists in running several PLS from randomly generated solutions; at each iteration, a single solution is selected then perturbed with mutation and crossover; they use PLS in independent mod and test it with and without first dominating strategy.

2.2.4 Hybrid meta-heuristics

A hybrid meta-heuristic is a method combining different algorithmic components coming from different meta-heuristics. By combining the qualities of different meta-heuristics, superior results are generally obtained compared to the original methods from which they are based on. Maybe the most popular hybridization scheme consists in combining genetic operators with local search, introducing the concept of *memetic algorithms* [Moscato, 1989]. Hybrid meta-heuristics is a large field of research and we refer the reader to [Ehrgott and Gandibleux, 2008] about the multiple hybrid meta-heuristics developed for MOCO.

Due to recent successes of local search on hard MOCO problems [Dubois-Lacoste et al., 2013, Ke et al., 2014, Lust and Tuyttens, 2014], we are particularly concerned in hybridization implying local search only. In particular, the Multi-Objective Memetic Algorithm based on Decomposition (MoMad) [Ke et al., 2014] developed quite recently obtains remarkable results on some classical MOCO problems. The method combines the principle of Decomposition of MOEA/D with PLS. As a decomposition method, MoMad decomposes a MOCO problem into a fixed number of weighted sum problems and optimizes them simultaneously in a collaborative way. At each iteration, the newly generated solutions form a starting set for PLS.

Algorithm 9 describes MoMad. The method takes as input parameters a stopping criterion, a single-objective solver used during initialization, a decomposition parameter controlling the number of weights generated by MDW during initialization, and a neighborhood structure for PLS.

The initialization of MoMad (cf. Algorithm 10) is similar to the initialization of MOEA/D: a number of weights is produced with MDW and to each weight λ is assigned an incumbent $x \in X$ created through the optimization of the weighted sum problem $\lambda f(\cdot)$ by the single-objective optimizer. Each pair (λ, x) is inserted into the set of all sub-problems Π, and the global archive $\overline{X}_{all}$ is updated with all generated incumbents. The initialization returns Π and $\overline{X}_{all}$. Then (cf. Algorithm 9), the current starting set of PLS $\overline{X}_{pls}$ is initialized with $\overline{X}_{all}$.

Each iteration of the main loop is composed of two steps:

Algorithme 9 : `MoMad`

Input : *stopping criterion*, *single-objective optimizer*, decomposition parameter D, neighborhood structure $\mathcal{N}_k$
Output : global archive $\overline{X}_{all}$

1 $(\Pi, \overline{X}_{all}) \leftarrow$`Initialization(`*single-objective optimizer, D*`)`
2 $\overline{X}_{pls} \leftarrow \overline{X}_{all}$
3 **repeat**
  `// PLS step:`
4 $\overline{X}_{all} \leftarrow$`PLS(` $\mathcal{N}_k$, false, $\overline{X}_{pls}$, $\overline{X}_{all}$`)`
5 $\overline{X}_{pls} \leftarrow \emptyset$
  `// perturbation step:`
6 **foreach** $(\lambda, x) \in \Pi$ **do**
7 $x' \leftarrow$`Perturbation(`x`)`
8 $x'' \leftarrow$`LocalSearchDescent(`$\lambda f(\cdot)$, x'`)`
9 `UpdateSubProblems(` x'', Π`)`
10 **if** `Add(` x'', $\overline{X}_{all}$`)` **then**
11 `Add(` x'', $\overline{X}_{pls}$`)`
12 **until** stopping criterion *is met*;
13 return $\overline{X}_{all}$

Algorithme 10 : `Initialization (MoMad)`

Input : *single-objective optimizer*, decomposition parameter D
Output : set of sub-problems Π, global archive $\overline{X}_{all}$

1 $\overline{X}_{all} \leftarrow \emptyset$
2 $\Lambda \leftarrow$`MDW(`D`)`
3 $\Pi \leftarrow \emptyset$
4 **foreach** $\lambda \in \Lambda$ **do**
5 $x \leftarrow$`Solve(`$\lambda f(\cdot)$, *single-objective optimizer*`)`
6 $\Pi \leftarrow \Pi \bigcup (\lambda, x)$
7 `Add(`x, $\overline{X}_{all}$`)`
8 return $(\Pi, \overline{X}_{all})$

- PLS step (lines 4-5 of Algorithm 9): a PLS is conducted from $\overline{X}_{pls}$. During this PLS, the incumbents of Π are updated with the newly generated solutions in the following way: a new solution x' replaces at most one single incumbent of a sub-problem $\pi = (\lambda, x) \in \Pi$ if $\lambda f(x') < \lambda f(x)$. After PLS, $\overline{X}_{pls}$ is reset.

- Perturbation step (lines 6-11 of Algorithm 9): for each sub-problem $\pi = (\lambda, x) \in \Pi$, the incumbent is first perturbed, then a local search descent optimizing $\lambda f(\cdot)$ is applied from the perturbed solution. The optimized solution updates Π, $\overline{X}_{pls}$ and $\overline{X}_{all}$. This perturbation step aims at generating a new starting set $\overline{X}_{pls}$ for the PLS run of the next iteration.

Although not defined as such by the authors [Ke et al., 2014], it is interesting to see that MoMad is an IPLS and falls indeed in the framework we have introduced in the previous section.

To our knowledge, MoMad is one of the most competitive meta-heuristics as it is the current state-of-the-art method for bi-objective TSP and MO Multidimensional KP [Ke et al., 2014].

2.2.5 Monte Carlo Search

This section presents Monte Carlo Search. We first discuss about basic notions of Monte Carlo Search, then present a key Monte Carlo Search method and its MO version, and finally make a focus on the method used in this thesis. Exceptionally, we consider in this section only, that objective(s) have to be maximized, in order to be consistent with some important formulas presented below.

2.2.5.1 Basic notions of Monte Carlo Search

Monte Carlo Search is a class of stochastic search algorithms for optimizing *sequential problems*, i.e. problems that can be represented as tree of sequential decisions. Over the last few years, Monte Carlo Search had a profound impact on many different domains of Artificial Intelligence, particularly on single/multi-player games on combinatorial or even continuous decision spaces. In addition, the Monte Carlo Search research community is particularly active and we refer the reader to the essential survey work of Browne et al. [Browne et al., 2012] on this domain.

From this thesis perspective, we are only concerned by single-player combinatorial problems with perfect information and non-stochastic transition model. Such a problem can be expressed by:

- A set of states $S \subset 2^E = \{0,1\}^q$. A *state* $s \in S$ corresponds to a *partial feasible solution* of the addressed problem. We note $s_0 \in S$ the initial state such that $s_0 = \{\}$, and $S_{term} \subset S$ the set of all *terminal states*. A *terminal state* respecting the constraints of the addressed problem corresponds to a feasible solution $x \in X$.
  For example, for KP, a state could be a feasible knapsack containing some items. Concerning AP, a state could be a feasible assignment of some people to some machines, and a path for TSP.
- A set of actions A. An action $a \in A$ is an operation on a state $s \in S$ adding and/or removing some elements $e \in E$ from s.
  For example, an action for KP could be the operation of adding or removing a number of items from a knapsack. Concerning AP, an action could be the act of (un)assigning a number of people to machines; and for TSP, the insertion/deletion of a valid edge in a path.
- A *transition function* $t : S \times A \to S$. Given a state $s \in S$ and an action $a \in A$, the transition function applies a on s, leading to a new state.
- A *reward function* $F : X \to \mathbb{R}$ to maximize.

The search space of a sequential combinatorial problem can be represented as a tree or a directed acyclic graph, whose nodes correspond to states and directed edges to actions leading to subsequent states. The root represents the initial state, and leaves of the tree correspond to terminal states. The exploration of the tree is performed by the transition function t. This type of tree is called a *game tree*.

Solving such a sequential problem consists in finding, through the exploration of the game tree, the *sequence* of actions starting from the initial state s_0 and leading to a terminal state (i.e. a feasible solution) $x^* \in X$ optimizing F. Therefore, the aim of a method optimizing such a problem is to iteratively select an action from a current state to finally reach the optimal solution x^*.

Let a *random simulation* be a sequence of actions selected via a (possibly biased) stochastic process called *policy*. A random simulation begins from any state and leads to a terminal state (i.e. a feasible solution). The *reward* associated to a random simulation is the fitness value of the terminal state reached at the end of the simulation.

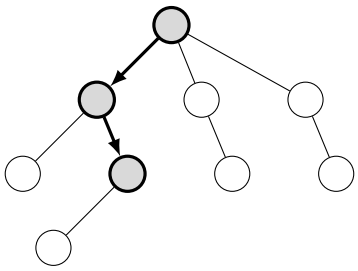
Monte Carlo Search is based on the fundamental work [Abramson, 1990] which demonstrates that the average reward obtained by performing many random simulations with a uniform distribution from a given state (resp. action) evaluates well this state (resp. action). In other words, performing many random simulations is a good indicator for measuring the ability of this state (resp. action) to lead to the optimal solution x^*.

Since this preliminary work, several Monte Carlo Search methods with many different versions have been proposed and still follow the same idea of guiding the search through random simulations, although random simulations currently employed generally follow more complex policies than uniform distribution.

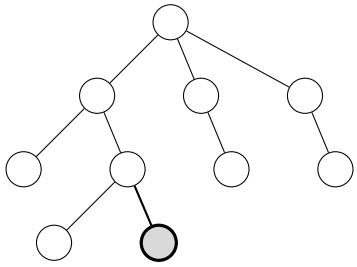
Besides, these methods mainly differ on two major aspects:

- The random simulation policy. Different random policies are available and the choice mainly depends on the Monte Carlo Search method applied and the problem addressed. As examples of simple sampling strategies, one can consider uniform sampling, Boltzmann sampling [Landau and Lifshitz, 1980], epsilon greedy strategy [Sutton and Barto, 1998] and many others. Naturally, the use of problem-specific heuristics generally enhances sampling results (see [Cazenave, 2016] for example).

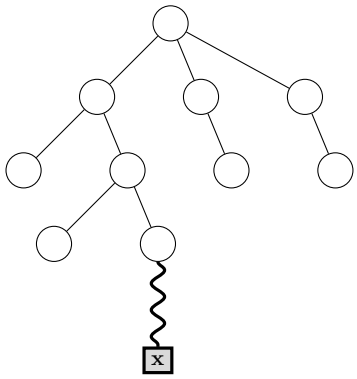
- The online management of collected data. The idea is to use the data provided by the simulations to improve the simulation process for the next iterations through online reinforcement learning techniques.



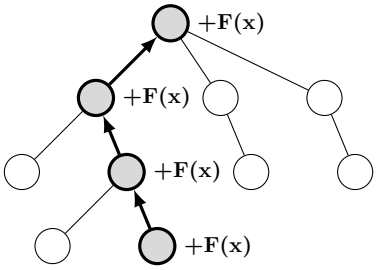
(1) **Selection** following the child node selection policy UCB.



(2) **Expansion**.



(3) **Simulation** following a random simulation policy.



(4) **Back-propagation**.

Figure 2.5 – One iteration of the UCT approach.

2.2.5.2 Monte Carlo Tree Search

Single-objective case

Monte Carlo Tree Search (MCTS) [Coulom, 2006] is a crucial Monte Carlo Search framework. We describe the most popular method in the MCTS family, called Upper Confidence bounds applied to Trees (UCT) [Kocsis and Szepesvàri, 2006]. UCT iteratively builds a partial game tree in an asymmetric manner, associating to each node some statistics collected from random simulations. These statistics are used in return to guide the construction of the tree and reinforce the accuracy of gathered statistics. More precisely, each iteration of UCT consists in four steps [Chaslot et al., 2008] (cf. Figure 2.5):

1. *Selection*: starting from the root, a *child node selection policy* called Upper Confidence Bound (UCB) [Auer et al., 2002] is recursively applied to select from the current node the next child node to visit, and this way descends through the tree until the most urgent expandable node is reached. A node is expandable if it represents a non-terminal state and has still unvisited children. From a given node s, the selected action is the following:

$$\underset{a \in A(s)}{\arg\max} \left\{ \frac{\sum F(s,a)}{N(s,a)} + c \times \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$

   where $A(s)$ is the set of actions available from s, $\sum F(s,a)$ is the sum of rewards of random simulations obtained by choosing action a from s (in other words: $\sum F(s,a)$ is the sum of all rewards of terminal states attained by random simulations through the selection of action a from s). $N(s,a)$ represents the number of times the action a was chosen from state s, $N(s)$ counts how many times the node s was visited in the tree, and $c > 0$ is a constant to be defined. This formula enables a trade-off between intensification, represented by the first term, and exploration, represented by the second term. The larger c is, the more promoted the exploration will be.

2. *Expansion*: a node is added to expand the tree.

3. *Simulation*: a random simulation is run from the new node according to a random simulation policy. Let $x \in X$ be the newly generated solution.

4. *Back-propagation*: the simulation reward $F(x)$ is back-propagated through each selected node s to update its statistics: $F(x)$ is added to the sum of rewards $\sum F(s,a)$ related to s, and the counter $N(s)$ is incremented.

UCT stops when a given stopping criterion is reached. The other MCTS algorithms mainly differ from UCT by using different child node selection policy and back-propagation mechanisms. A number of improvements have been developed for MCTS approaches these last years [Browne et al., 2012], like Progressive Widening [Coulom, 2007] or (Generalized) Rapid Action Value Estimate [Gelly and Silver, 2007, Cazenave, 2015]. Algorithms following the MCTS framework have been successfully applied to many domains, including single player games or problems [Schadd et al., 2008], General Game Playing [Finnsson and Björnsson, 2008], two-player games such as Hex [Arneson et al., 2010], and naturally Go [Coulom, 2006], for which the now famous AlphaGo program [Silver et al., 2016] became the first program to beat a professional Go player.

Multi-objective case

The use of MCTS for MO optimization is quite recent. Let us assume in this section that the reward function to maximize is multi-dimensional: $F : X \longrightarrow \mathbb{R}^p$. Except in [Wu et al., 2013] and [Wang and Sebag, 2013], the few works published have made a focus on a specific domain of MO optimization: MO

Online Reinforcement Learning (MOORL) [Perez et al., 2013]. The general idea of MOORL consists in controlling an agent in an unknown environment in which the agent has to discover step by step which actions should be performed in order to optimize a number of objectives. Naturally, computational resources allocated do not allow the agent to explore the whole environment. A MOORL problem can be modeled as the sequential decision problem defined in previous section with the following difference: the transition function t is generally stochastic: the state returned by the function partly depends on randomness, so that applying the same action from the same state twice may lead to a different state;

The great difference between MOORL problems with classical problems of Operations Research we consider (like MOTSP, MOKP, MOQAP, MOFSP, etc.) is that, in addition to the uncertainty induced by the transition function, the problem is not known in advance but locally discovered by the agent while performing actions. Given these features, the algorithms optimizing MOORL are highly general methods and a focus is made on their adaptability to any problem through online reinforcement learning techniques.

An example of MOORL problem is the MO Physical TSP (MOPTSP), a real-time game in which the player must drive a ship in a continuous 2D space in order to visit a series of way-points scattered around the world. The path that is steered may contains obstacles to avoid, but unknown in advance. The three objectives to optimize are the time spent and the fuel consumed by the ship, but also the damages caused by obstacles to the ship.

MCTS algorithms developed for MO optimization can be separated into two distinct categories.

The first category of works concerns single-objective MCTS algorithms applied to MO problems. In [Powley et al., 2013], the authors apply a single-objective MCTS method to MOPTSP for finding a single solution through optimization of a weighted-sum aggregation with a single fixed weight.

In [Wu et al., 2013], the authors propose a single-objective UCT method specifically designed for the MO Flexible Job-shop Scheduling Problem and combine it with single-objective LS. While the objectives are never optimized simultaneously, potentially efficient solutions are still memorized during the whole run. The method obtains similar results than competitors on very small instances containing less than five non-dominated points.

The second category of works consists in real adaptations of MCTS to MOCO for finding the efficient set. To our knowledge, only two groups of authors have proposed such adaptations, differing from single-objective MCTS in two main aspects:

- an archive is now systematically maintained in order to memorize the best so far efficient set approximation found.
- as a random sampling now returns a vector reward instead of a scalar reward, the selection and back-propagation steps have to be adapted accordingly, as well as the statistics related to each node.

An initial attempt at adapting MCTS to MOCO was addressed in [Wang and Sebag, 2012, Wang and Sebag, 2013]. Unfortunately, as suggested by the authors themselves, the two proposed methods were computationally prohibitive, and did not obtain convincing results (see [Perez et al., 2013]).

In the meantime, Perez et al. proposed an interesting adaptation of MCTS to MOCO into two studies [Perez et al., 2013, Perez et al., 2015], and we call it *MOMCTS*. They globally use the same scheme as UCT while including some differences. The main idea of the method is to evaluate a state by using the hypervolume indicator. First, each node of the partial tree built by the method contains an archive. Hence a random simulation generates a new solution $x \in X$, it is back-propagated in the tree and presented to the archive of each encountered node. If x is accepted in the archive of the current node, the back-propagation of x

continues, otherwise it is stopped. This new back-propagation mechanism induces a slightly modified UCB formula for the selection step from a state s :

$$\arg\max_{a \in A(s)} \left\{ \frac{I_H(\overline{Z}(s), \overline{z})}{N(s,a)} + c \times \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$

where $\overline{Z}(s)$ is the image in the objective space of the archive stored in the state s, and $\overline{z}$ is a predefined reference point (not indicated in the paper, probably the null vector).

This way, the root node memorizes the best-so-far approximation of the efficient set, and each node of the tree has an estimate of the quality of the solutions reachable from there. To our knowledge, MOMCTS has been tested on MOORL problems only: a small game called Deep Sea Treasure, on MOPTSP and a very similar bi-objective game, the Puddle Driver. MOMCTS found better results than the well-known MOEA called NSGAII [Deb et al., 2000] (which is, to our knowledge, far for being a state-of-the-art MOEA nowadays).

As suggested by the authors in [Perez et al., 2013], MOMCTS is not intended to compete with state-of-the-art methods on classical Operations Research problems, but rather to propose an efficient method for MOORL research field only.

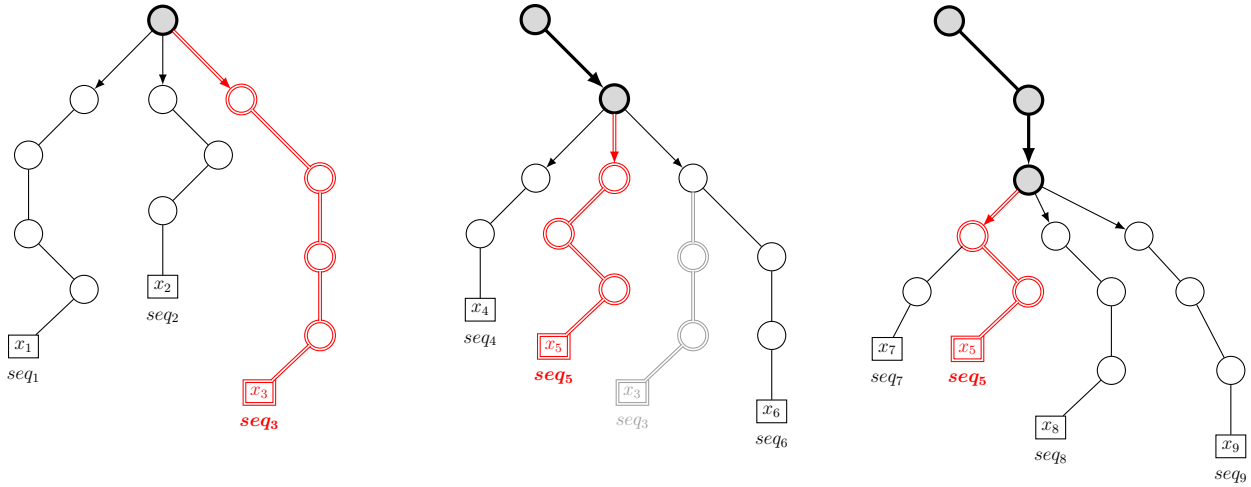
2.2.5.3 Nested Monte Carlo Search

An efficient Monte Carlo Search algorithm for single-player combinatorial problems is Nested Monte Carlo Search (NMCS) [Cazenave, 2009]. The principle of NMCS is different from the principle of MCTS. It still uses a lot of random simulations in order to find a good sequence of actions, but instead of building a partial game tree and collecting statistics from simulations, it memorizes the best sequence of actions found so far and follows it. It tries every possible actions from each traversed state and uses nested levels of search. Trying all possible actions enables NMCS to diversify the search, while memorizing the best sequence of actions enables it to intensify the search.

Algorithme 11 : `NMCS`

Input : *level*, starting state s_{start}
Output : best sequence of actions seq_{best}, best solution x_{best}

```
1  s ← s_start
2  if level = 0 then
3      return RandomSimulation(s)
4  i ← 1
5  repeat
6      foreach a ∈ Actions(s) do
7          s ←PerformAction(a, s)
8          (seq, x) ←NMCS(level − 1, s)
9          s ←UnperformAction(a, s)
10         if F(x) is better than F(x_best) then
11             x_best ← x
12             seq_best ← seq
13     a_best ←seq_best[i++]
14     s ←PerformAction(a_best, s)
15 until IsTerminal(s);
16 return (seq_best, x_best)
```



(a) $F(x_3)$ is the best ⇒ set seq_3 as best seq.

(b) $F(x_5)$ is the best ⇒ set seq_5 as best seq.

(c) $F(x_5)$ is still the best ⇒ keep seq_5 as best seq.

Figure 2.6 – Illustration of three iterations of a NMCS of level 1. Filled nodes and bold edges correspond to the part of the best sequence already performed. The remaining part of the currently memorized best sequence is indicated with double red strokes.

NMCS is detailed by Algorithm 11. The method is recursive and takes as input parameters a level of recursion *level* and a starting state s_{start} (initially s_0). It returns the best sequence of actions found so far seq_{best} and the related solution x_{best}.

The principle of NMCS is to use different levels of search. At level zero it performs a random simulation (lines 2-3). At greater levels (lines 4-16), it performs an iterative and more informed search. At each iteration, it generates all the available actions from the current state s, and for each action performed, it calls a $level-1$ search (lines 6-9). The nested search returns the best sequence of actions seq it found and the related solution x. As the simulations are stochastic, it is not guaranteed that a nested search will always improve on previous searches. In order not to lose the actions of the best sequence found by a previous search, NMCS memorizes the best sequence (lines 10-12). If none of the actions improve on the best sequence, the action of the best sequence is played; otherwise the best sequence is updated with the newly found sequence and the best action from the current state s is played (lines 13-14).

Figure 2.6 describes how a NMCS of level 1 uses random simulations to iteratively select the action to perform.

NMCS has been successfully applied on a number of different combinatorial problems. It has found world records on difficult problems [Kinny, 2012, Eliahou et al., 2013], has been applied to puzzles [Cazenave, 2009], software engineering [Poulding and Feldt, 2014] and transportation problems, such as the Bus Regulation Problem [Cazenave et al., 2009], and the Traveling Salesman with Time Windows Problem [Rimmel et al., 2011]. A combination with beam search has even been developed [Cazenave, 2012].

A related algorithm is Nested Rollout Policy Adaptation (NRPA) [Rosin, 2011] that learns a random simulation policy online using nested levels of learning. NRPA has found world records on difficult puzzles and has been applied to combinatorial problems such as the Traveling Salesman with Time Windows Problem [Cazenave and Teytaud, 2012, Edelkamp et al., 2013] and the Multiple Sequence Alignment Problem [Edelkamp and Tang, 2015]. Finally, Edelkamp and Greulich [Edelkamp and Greulich, 2014] used NRPA for MOPTSP and optimize a single objective.

To our knowledge, neither NMCS or NRPA have been adapted to MOCO.

2.3 MO Traveling Salesman Problem: state-of-the-art

This section first recalls some important techniques used for single-objective *symmetric* Traveling Salesman Problem (TSP), then reports a non-exhaustive literature review of optimization methods for MOTSP focused on meta-heuristics approaches.

2.3.1 Basic LS techniques for TSP

TSP is probably the most studied NP-hard single-objective combinatorial problem. We refer the reader to [Gutin and Punnen, 2006] for an analysis of various methods for TSP and [Rego et al., 2011] for a survey of meta-heuristics and recent advances on TSP. We are interested here in ILS for TSP.

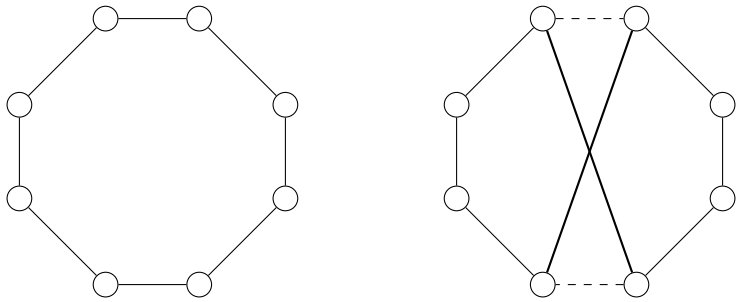


Figure 2.7 – An example of a 2-exchange move applied on a tour.

Let $\mathcal{N}_k$ be a k-neighborhood structure. For a number of 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 TSP, where a solution $x \in X$, which represents a Hamiltonian cycle (also called a *tour*) in a complete graph of n cities, is composed of n edges. In this context, let $x' \in X$ be another feasible solution, then $x' \in \mathcal{N}_k(x)$ if x' is an Hamiltonian cycle obtained from x by exchanging k edges.

The most elementary neighborhood structure for TSP is the so-called *2-exchange neighborhood* (Figure 2.7). A LS descent using a k-exchange neighborhood structure is called a *k-opt*.

A number of speed-up techniques for k-opt have been designed (see [Helsgaun, 2000, Helsgaun, 2009, Blazinskas and Misevicius, 2011] for details on the subject) and among them, two are essential: *candidate edge list* [Johnson and McGeoch, 1997] and *don't look bits* [Bentley, 1992].

The candidate edge list technique consists in associating to each city a fixed-size list of candidate edges, generally the l best edges incident to the city minimizing the cost function of the addressed TSP instance. The list is limited to a reasonable size, compromise between quality and running time.

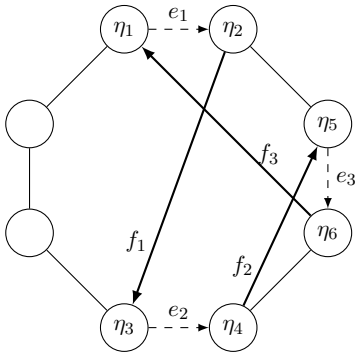


Figure 2.8 – Example of a 3-opt move applied on a tour.

In general, a k-opt (cf. Figure 2.8) works as follows. Let $x \in X$ be the current tour. At each iteration of a k-opt, all cities are successively considered as starting cities for k-exchange moves. From a starting city η_1, a k-opt move attempts to find two sets of edges $E_{out} = \{e_1, e_2, ..., e_k\}$ (*out-edges*) and $E_{in} = \{f_1, f_2, ..., f_k\}$ (*in-edges*) such that, if the edges of E_{out} are deleted from x and replaced by the edges of E_{in}, the result is a better tour. E_{out} and E_{in} are initially empty and iteratively constructed. At each step i, a pair of edges (e_i, f_i) are added to E_{out} and E_{in} respectively, such that $e_i = (\eta_{2i-1}, \eta_{2i})$ and $f_i = (\eta_{2i}, \eta_{2i+1})$ share a city, as well as f_i and e_{i+1}, *and* f_i belongs to the candidate list of city η_{i+1}. At the end, the last in-edge f_k is chosen by default so as to be incident to the starting city, thus f_k may not be a candidate edge and is called the *close-up edge*. This way, the edge sequence $(e_1, f_1, e_2, f_2, ..., e_k, f_k)$ constitutes a close chain of adjoining edges. Note that the choice of the in-edges is restricted at each step so as to obtain a (valid) new tour.

During an iteration of a k-opt, the don't look bits technique consists in ignoring as starting city a city η_1 which previously failed to find an improving move, and such that its city neighbors in the current tour have not changed since that time. In general, this technique greatly speeds-up a k-opt.

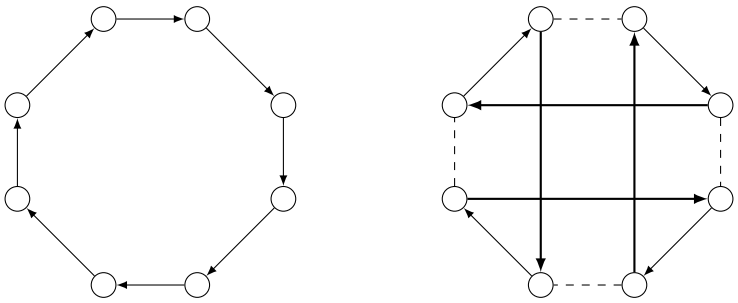


Figure 2.9 – An example of a double bridge move applied on a tour.

Concerning the perturbation move made during an ILS, the *double bridge move* [Martin et al., 1991] (Figure 2.9) is one of the most popular. Given a tour to perturb, this move first gives a direction to the tour, then cuts the current tour at four appropriately chosen edges into four sub-tours and reconnects these in a different order to yield a new starting tour without changing the direction.

Probably the most utilized and efficient method for TSP is an ILS called Lin-Kernighan [Lin and Kernighan, 1973]. This method uses the VND concept and the double-bridge move as perturbation move. Nowadays, two improved versions are generally used in the literature: Chained Lin-Kernighan (C-LK) [Applegate et al., 2003] designed by Applegate, Cook and Rohe, and the Lin-Kernighan implementation of Helsgaun (LKH) [Helsgaun, 2000].

Instead of perturbing the current solution to escape from the attraction basin of a local optimum, the technique of *data perturbation* (also called *noising method*) [Charon and Hudry, 1993, Codenotti et al., 1996, Charon and Hudry, 2002] suggests to modify input data by adding a random noise to the cost function of the addressed instance. This way, one hopes that the next LS descent will direct its search towards a different local optimum.

2.3.2 Literature review for MOTSP

This section presents a number of meta-heuristics and a single exact approach for MOTSP. Some methods are more detailed than others due to their significance for the optimization methods proposed later in the document. In particular, we start with out-of-date yet interesting methods and finish with the current best methods on MOTSP.

- To our knowledge, Jaszkiewicz is the first author having published approximations of non-dominated sets of bi-objective TSP instances. In [Jaszkiewicz, 2002], he proposed an improved version of the

MO Genetic local search (MOGLS) initially proposed by Ishibushi and Murata [Ishibuchi and Murata, 1996, Ishibuchi and Murata, 1998], who applied it on MOFSP.
In particular, the authors use the concept of *locked edges* [Jaszkiewicz, 1999], consisting in forbidding the use of frequent edges in order to speed-up the optimization process.

- Jaszkiewicz and Zielniewicz have experimented on bi-objective instances the Pareto Memetic Algorithm (PMA) [Jaszkiewicz and Zielniewicz, 2006] and found better results than MOGLS.
- Kumar and Singh [Kumar and Singh, 2007] introduce a memetic algorithm and find comparable results to MOGLS and PD-TPLS (presented below in paragraph *Two-Phase Local Search*) on bi-objective instances.
- The Evolutionary MO Simulated Annealing Algorithm (EMOSA) [Li and Landa-Silva, 2011] of Li and Landa-Silva (rapidly described in Section 2.2.3.2) is compared with other MOSA-like algorithms and obtains better results on all tested bi-objective and tri-objective TSP instances (of sizes from 50 to 100). However, the SA used to optimize each considered weighted sum problem uses a simple 2-exchange neighborhood, without any LS speed-up techniques, making EMOSA irremediably non-competitive compared to the best current methods on MOTSP.
- Various MO Ant Colony Optimization (MOACO) algorithms have been proposed in recent years (see [Cheng et al., 2012, García-Martínez et al., 2007] among others). López-Ibáñez and Stützle [López-Ibáñez and Stützle, 2012] 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, like EMOSA, results of MOACO are not comparable to the best current methods on MOTSP.
- MOEA/D-ACO [Ke et al., 2013] combines MOEA/D with a MOACO algorithm. Like EMOSA and MOACO, results of MOEA/D-ACO are not comparable to the best current methods on MOTSP.

Let us now present particularly interesting works concerning MOTSP.

AUGMECON2

Recently, Florios and Mavrotas [Florios and Mavrotas, 2014] proposed an ε-constraint method called AUGMECON2 for solving exactly MOTSP. To our knowledge, they provided the exact non-dominated sets of 16 bi-objective instances of size 100, a single bi-objective instance of size 150 and a single tri-objective instance of size 15. Although AUGMECON2 provides the guarantee of finding the non-dominated set in theory, the method has huge limitations in practice.

First, while current best meta-heuristics obtain quite good results on bi-objective instances of size 100 within running times around several seconds, at most a minute, AUGMECON2 finds in average in 30 hours the non-dominated set of such instances (on similar computers). In addition, the method scales badly with the instance size: it takes several days for finding the non-dominated set of a bi-objective instance of size 150. According to the authors, this instance pushed to its limits AUGMECON2. Finally, the method does not scale at all with the number of objectives as the largest tri-objective instance handle by AUGMECON2 was a 15 cities instance solved in 5 hours.

The authors concluded that the generation of the exact non-dominated set for tri-objective TSP, even of small size (20–30 cities), is rather an utopian task and the use of approximate algorithms seems to be the only choice.

AUGMECON2 corresponds to a great step of exact methods for bi-objective TSP. By contrast, it highlights the difficulty of exact methods to scale well with both the instance size and the number of objectives, and strengthens the idea of the usefulness of meta-heuristics for such a hard problem.

Two-Phase Local Search

Paquete and Stützle introduce in [Paquete and Stützle, 2003] the concept of Two-Phase Local Search (TPLS), later generalized in [Paquete and Stützle, 2009b].

TPLS first generates a sequence of weights with MDW, sorted such that the first weight considers a single objective, and successive weights differ only by $\pm\frac{1}{D}$ in any two objectives, where D is the parameter of MDW controlling the number of weights generated. Then TPLS iterates on the sequence of weights and optimizes the weighted sum problem corresponding to each weight with an ILS, starting from the best solution found at the preceding iteration. The first starting solution is greedily generated at random.

Double TPLS (D-TPLS) runs TPLS p times such that the first weight considers a different objective each time.

Pareto Double TPLS (PD-TPLS) consists in two phases. The first phase runs D-TPLS, and the second phase conducts a *single iteration* of PLS from the solutions generated by D-TPLS.

The ILS optimizing the weighted sum problems uses a 3-opt with the speed-up techniques previously introduced. During the second phase of PD-TPLS, the single iteration of PLS uses the same candidate lists and neighborhood structures as ILS.

PD-TPLS has been tested on bi-objective TSP instances of sizes from 100 to 500 and finds better results than MOGLS [Paquete and Stützle, 2003]. Contrary to PD-TPLS, D-TPLS has been tested on tri-objective instances of sizes from 100 to 300 and finds better results than MOGLS [Paquete and Stützle, 2009b]. As indicated by the authors, PD-TPLS has not been used for $p = 3$ as the single PLS iteration conducted in the second phase generated to much solutions.

In [Dubois-Lacoste et al., 2011b], Dubois-Lacoste et al. analyze and improve the anytime behavior of TPLS.

2PPLS

While 2PPLS [Lust and Teghem, 2010] is no longer the best heuristic for MOTSP, it is probably the most important as their authors proposed several interesting and highly efficient speed-up techniques for MOTSP. As previously indicated in Section 2.2.3.6, 2PPLS introduces the concept of 2-Phase PLS. Originally designed for the bi-objective case only, 2PPLS introduces an approximation version of the dichotomic scheme [Cohon, 1978, Aneja and Nair, 1979] to approximate the supported efficient set in the first phase. The solver used to optimize the different weighted sum problems is chained Lin–Kernighan (C-LK) [Applegate et al., 2003].

For the second phase, PLS uses a 2-exchange neighborhood structure, and originally PLS did not use candidate edge lists, making the convergence of PLS extremely slow. The authors thus propose to speed-up PLS by adapting the concept of candidate edge list for the MO case. In fact, memorizing into the candidate edge list of each city the l best edges is no longer possible in MOCO, as this technique assumes a total order of the edges given a cost function c, while such an order is absent in MOCO. On the other hand, they found that the set of all edges present in at least one solution of the global archive at the end of the first phase, and the set of all edges used in at least one solution of the global archive at the end of the second phase are very similar.

In consequence, they decided to insert into the candidate edge lists of PLS the set of all edges present in at least one solution of the global archive at the end of the first phase. 2PPLS with this speed-up technique is called 2PPLS-SpeedP1 [Lust and Jaszkiewicz, 2010]. Experiments showed that 2PPLS-SpeedP1 obtains in average an approximation of slightly worse quality compared to 2PPLS but with a much faster convergence, particularly when the instance size grows.

In addition to the candidate edge list for PLS, the authors proposed an adaptation of don't look bits for PLS. However, contrary to the single-objective case, even though this technique reduces the running time of

PLS, it also drastically reduces the results quality and this proposition of don't look bits for MOTSP does not seem to be viable for now.

In order to reduce the convergence time of PLS and produce a better quality approximation, the authors propose to generate an even better quality starting set during phase one. The authors stated that while the dichotomic scheme has already found many *supported* efficient solutions, the improvement of the starting set can only be done by generating *non-supported* efficient solutions. To do so, after the dichotomic scheme and before PLS, they generate a number of weights with MDW, then optimize each related weighted sum problem with C-LK after having perturbed it with data perturbation. They tested this technique on small-size instances ($n = 100, 200$) and found good results, but surprisingly, the authors do not go further with data perturbation in their next study on MOTSP [Lust and Jaszkiewicz, 2010], maybe due to the high number of parameters needed (3) while data perturbation is just a component among others of a more general method.

To conclude on 2PPLS, experiments conducted in [Lust and Teghem, 2010] and [Lust and Jaszkiewicz, 2010] show that 2PPLS and its variants outperform MOGLS and PMA on tested bi-objective instances of sizes from 100 to 1000.

MoMad

MoMad [Ke et al., 2014] has already been described in Section 2.2.4. Like 2PPLS-SpeedP1, MoMad uses Chained Lin-Kernighan as solver for initialization; and both PLS and the LS descent of the main loop use a 2-opt with candidate lists containing all edges of the solutions accepted in the global archive. MoMad has been tested on bi-objective instances of sizes from 200 to 1000 and experiments showed that MoMad outperforms 2PPLS-SpeedP1 on tested instances.

Perturbed Decomposition Algorithm

Perturbed Decomposition Algorithm (PDA) [Cornu et al., 2017] is an enhanced version of MoMad we have recently proposed to tackle MOTSP. At main loop, PDA optimizes the sub-problems with a 3-opt ILS run (instead of a 2-opt LS descent), and memorizes all incomparable local optima produced by C-LK (at initialization) and ILS (at main loop). PDA introduces data perturbation into the decomposition framework: the sub-problems are data perturbed at initialization and the sub-problem providing the lowest number of solutions accepted in the global archive is re-data perturbed in order to diversify the search. PDA has been compared to MoMad and PD-TPLS on a benchmark of bi-objective instances of sizes from 100 to 1000, and tri-objective instances of sizes from 30 to 300. PDA obtained better results than its competitors. Therefore, PDA is the current best heuristic for bi-objective and tri-objective TSP.

In consequence, MoMad is the best heuristic for bi-objective TSP proposed by other authors, and (PD-)TPLS is the best heuristic for tri-objective TSP proposed by other authors.

Recent works published during the drafting process

The research field of MOTSP optimization is pretty dynamic, and very recently two interesting works have been published. We do not have taken them into account as they have been published during the drafting process of this thesis. Lust and Jaszkiewicz [Jaszkiewicz and Lust, 2017] propose and improved version of 2PPLS by better controlling the number of runs of ILS in order to re-launch PLS and obtain better results. The method obtains results at least comparable to MoMad.

In [Jaszkiewicz, 2017], Jaszkiewicz proposes a version of PLS able to tackle efficiently the many-objective case.

2.4 Archives: state-of-the-art

In MO optimization problems, archives are used to store incomparable solutions. There exists two main scenarios:

- the *offline scenario*: a whole set of solutions known in advance is presented to an initially empty archive.
- the *online scenario*: an unknown number of solutions are presented one by one to an archive.

Specific algorithms have been developed for each scenario. The offline scenario is intensively studied (see [Kung et al., 1975] for a discussion and some theoretical results on the subject) and many different algorithms have been developed, particularly these few years [Zhang et al., 2016]. This scenario occurs in two main situations: when proceeding to the *non-dominated sorting* [Goldberg, 1989] of a solution set and when computing the hypervolume of a solution set. In both situations, an efficient algorithm for archiving is crucial for good performance.

Given a set of points in the objective space, **Non-Dominated Sorting** [Goldberg, 1989] aims at assigning a rank to each point such that the lower rank, of better quality the point. It consists in partitioning a given set of solutions into several archives. This process is iterative and manages a current set of solutions, initialized with the initial solution set. At each iteration i, all points not dominated are extracted from the current set and constitute the front of solutions of rank i. The process stops when the current solution set is empty. Non-dominated sorting is often used in a number of MOEA, in order to select solutions during the parent selection and population update steps (cf Section 2.2.2). We refer the reader to [Zhang et al., 2016] for a survey on algorithms for non-dominated sorting.

Concerning the use of archives for hypervolume computation of a set of solutions, dedicated algorithms partition the archive into sub-archives in smaller dimensions. We refer the reader to [Fonseca et al., 2006] and [While et al., 2012] for more details.

We are only concerned by the online scenario. In practice, this scenario occurs when an archive is employed within a MO optimization method, which regularly presents a newly generated solution to the archive throughout the run. In fact, the computational cost required to maintain an archive may become an important part of the total computational cost of a number of MOCO methods, such as search regions-based [Klamroth et al., 2015], MO dynamic programming [Bazgan et al., 2009a] or PLS. The rest of this section is dedicated to a non-exhaustive literature review of the most popular or efficient archives for the online scenario. In particular, we make a focus on algorithms which inspired us for the design of a number of components of the archives we propose. We refer the reader to [Altwaijry and Menai, 2012] for a quite recent survey on archives.

2.4.1 Linear list

The simplest data structure for managing an archive is the *linear list*. In this structure, a candidate solution is compared to all solutions except if a solution weakly dominates the candidate; in this case the candidate is rejected. Otherwise, all the dominated solutions of the list are removed and the candidate is inserted. Whether the candidate is accepted or rejected, the complexity in terms of number of dominance comparisons is in $O(p \times s)$, where s is the size of the archive. Although not competitive at all compared to the best current archives [Jaszkiewicz and Lust, 2016], the linear list is probably the most used algorithm for managing a set of incomparable solutions. As examples among many others, [Bazgan et al., 2009a] and [Li et al., 2014] use a linear list as archive, while they are considered as state-of-the-art methods on their respective domains.

In [Bentley et al., 1993], the authors propose a simple yet efficient speed-up technique when using a linear list called *Move-to-Front*:

- if the candidate solution is accepted, then insert it in front of the list;
- otherwise, push in front of the list the solution dominating the candidate.

This technique does not improve the worst case complexity but experiments show that it greatly reduces the overall running time.

2.4.2 Sorted list

A *sorted list* uses the fact that in the bi-objective case, if incomparable solutions are sorted in increasing order of their coordinates on one objective, they are also sorted in decreasing order of their coordinates on the other objective. A sorted list works as follows. The solutions are maintained sorted in increasing order on the first (or the second) objective. When a candidate solution is presented to the list, a binary search using the sorting objective is performed to determine where to place the candidate. During the binary search, if a solution selected for the dichotomy decision weakly dominates the candidate, then the candidate is immediately rejected.

Once the candidate is placed, all the solutions at its left have a better value on the sorting objective while all the solutions at its right have a worse value. Therefore, if the preceding solution weakly dominates the candidate, then the candidate is rejected. Otherwise, the candidate is accepted; and any succeeding solution with a worse value than the candidate on the non-sorting objective is removed.

To have a constant time access to the solutions of the list during the binary search, the sorted list is implemented as a dynamic array. Therefore, once a solution is removed from the list, all the succeeding solutions are shifted to the left. Note that if only one solution is dominated, then the candidate simply replaces it and the succeeding solutions are not shifted; in this case, the complexity in time is in $O(p \times \log_2 s)$, where s is the size of the archive. Otherwise, if the candidate is accepted and dominates more than one solution, the complexity in time is in $O(p \times s)$. But on average, as recalled in [Jaszkiewicz and Lust, 2016], experiments show that the behavior of the sorted list is much better than the simple list since the worst-case scenario rarely occurs.

2.4.3 Quad-tree

Originally, a Quad-tree [Finkel and Bentley, 1974] is a tree-based data structure for indexing points in a multi-dimensional space. In [Habenicht, 1983], Quad-tree has been adapted as archive, then further developed in [Sun and Steuer, 1996] and [Mostaghim and Teich, 2005]. This data structure is based on the following observation: given a point $z \in Z$, the p-dimensional multi-objective space can be partitioned into 2^p regions with z as reference point, called *orthants* of z, such that all points in the same orthant have the same relative positioning with respect to z. In a Quad-tree, solutions are located in both internal and leaf nodes and each node η has at most $2^p - 2$ children. Each child of η is the root of a subtree memorizing all points localized in an orthant of the image $z = f(x)$ of the solution $x \in X$ stored in η. As a Quad-tree is an archive, the orthants dominating z and dominated by z can not exist. A Quad-tree can thus be seen as a hierarchy of orthants and this organization of data allows to guide efficiently the exploration of the tree. Figure 2.10 illustrates a Quad-tree in the bi-objective case.

Different versions of Quad-tree have been developed. We are particularly interested by the implementation proposed in [Tricoire, 2012], working as follows. When a candidate solution is presented to the Quad-tree, a first procedure explores the tree in order to check if the candidate is weakly dominated by a solution. If the candidate is accepted, then a second procedure identifies the nodes dominated by the candidate and extract any subtree whose root is dominated. Finally, a third procedure inserts the candidate and reinsert non-dominated parts of previously extracted sub-trees.

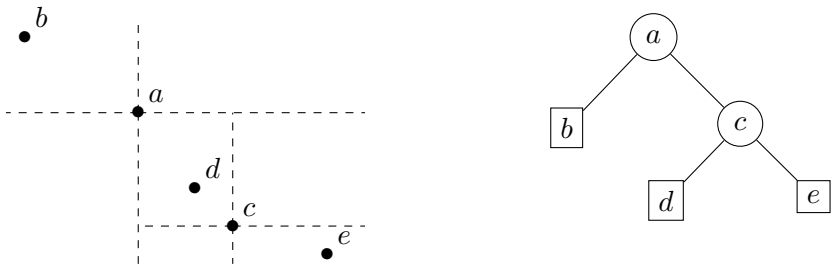


Figure 2.10 – A Quad-tree in the bi-objective case.

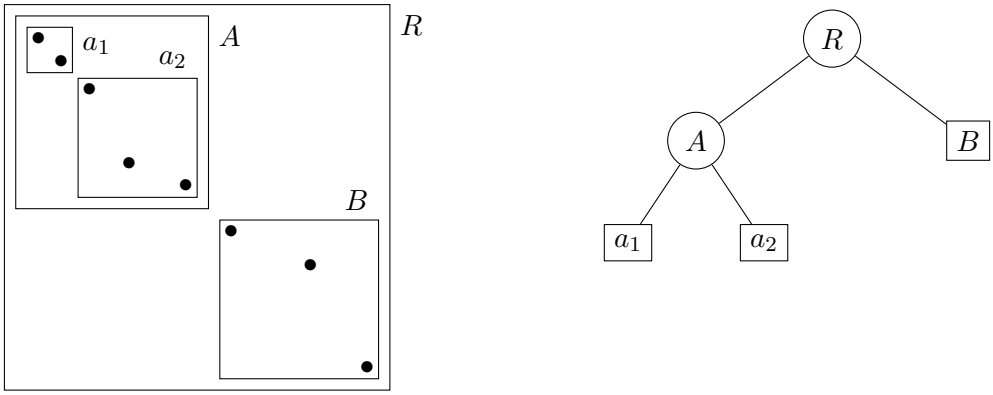


Figure 2.11 – A ND-Tree in the bi-objective case.

Quad-tree is efficient for checking if a candidate solution is weakly dominated or not, and inserting a new solution is computationally cheap. By contrast, once a solution of the Quad-tree is dominated, re-insertions are generally needed, while they are costly operations.

2.4.4 ND-Tree

Recently, Jaszkiewicz and Lust [Jaszkiewicz and Lust, 2016] proposed a new tree-based archive called *ND-Tree*. A ND-Tree stores the solutions in leaf nodes only. Each leaf node stores in a linear list a number of solutions neighbors in the objective space. Each (internal or leaf) node maintains approximate local ideal and nadir points, which define a bounding box including all the images (in the objective space) of the solutions stored in the subtree. The originality of the ND-tree comes from the use of basic properties of local ideal and nadir points to efficiently avoid searching many branches in the tree.

Figure 2.11 illustrates a ND-Tree in the bi-objective case.

A ND-Tree works as follows. When a candidate solution is presented to the ND-Tree, a first procedure explores the tree to check if the candidate is weakly dominated by any solution. If yes, the candidate is rejected. Otherwise, the candidate is accepted and dominated sub-trees and solutions are removed. Then a second procedure inserts it in a suitable leaf.

More precisely, the **first procedure** explores the tree using Depth First Search (DFS) starting from the root. For each traversed node, only four cases are possible (see [Jaszkiewicz and Lust, 2016] for proofs):

- If the approximate local nadir point weakly dominates the candidate, then all solutions contained in the node weakly dominate the candidate, therefore it is rejected.
- Otherwise, if the candidate dominates the approximate local ideal of the node, then all solutions contained in the node are dominated by the candidate. Therefore the subtree rooted at the node is removed and the candidate is accepted.

- Otherwise, if the approximate local ideal weakly dominates the candidate, or the candidate weakly dominates the local nadir, this means that the candidate is potentially weakly dominated or dominates a solution contained in the subtree. In this case:
  - If the node is a leaf, then the candidate is compared to each solution of the node. If a solution weakly dominates the candidate, then the candidate is rejected. Otherwise, any solution dominated by the candidate is removed.
  - Otherwise, the procedure is called from each child node. If the candidate is rejected by a child node, then it is rejected, otherwise it is accepted.
- Otherwise (the node is internal), the candidate is incomparable with both the approximate local ideal and nadir points. Therefore, the candidate is incomparable with all solutions stored in this subtree, and the current node is not explored.

Any empty node is removed from the tree during this first procedure.

Once the candidate is accepted, the **second procedure** aims at inserting the candidate in a suitable leaf of the tree. To do so, the tree is browsed by following a unique top-down path starting from the root and ending to a leaf. For each internal node traversed, the descent continues in the child node η for which the candidate is the closest from the center of its bounding box. When a leaf is reached, the solution is inserted, and the approximate local ideal and nadir of the leaf and of its ascendants are updated such that the corresponding bounding boxes contain the new solution.

Only a limited number of solutions is allowed in a single leaf node. If a leaf becomes overfilled after the insertion of a new solution, then it is split into l ($l \geq 2$) new leaves. The solutions contained in the split node are then redistributed to every new leaves, such that each leave contains at least one solution and, solutions neighbors in the objective space tend to be inserted in the same node. Then, the split node becomes an internal node with the l new leaves as children.

Note that the local and nadir points maintained by each node are approximated because they are updated only when a solution is inserted, but not when a solution is removed, which is less computationally costly.

In [Jaszkiewicz and Lust, 2016], the authors do not evaluate the worst case complexity of ND-tree. However they experimentally show that ND-Tree outperforms previous state-of-the-art structures, i.e. linear list, Quad-tree and the algorithm presented in [Drozdik et al., 2015] on different artificial benchmarks up to 6 objectives, and is equivalent to sorted list on bi-objective instances.

To our knowledge, ND-Tree is the current best archive for the online scenario.

Chapter 3

New archives

This chapter introduces two new archives offering efficient and original features: AVL-Archive specialized for bi-objective space, and NDR*-Archive for any-objective spaces. Both structures are self-balancing trees and benefit from self-adjusting versions, especially designed if a presumption of temporal and spatial locality exists between the solutions presented to the archive. In experiments, the archives we propose are compared with the best known archives of the literature on a large benchmark of instances simulating the generation of points of a MO meta-heuristic, and inside a PLS for MOTSP up to 5 objectives.

Introduction

The current best known archives, the sorted list for the bi-objective case and ND-Tree have shown good results on last studies on the online scenario (previously introduced in Section 2.4). However, we claim that both archives suffer from two crucial drawbacks: they are *unbalanced* tree structures, which can lead to nasty worst case scenario; and they do not take advantage of a principle often present when storing data in an online environment: the *temporal and spatial locality*. Temporal locality refers to the reuse of specific data within a relatively small time duration, while spatial locality refers to the use of data elements within relatively close storage locations[1].

The two data structures proposed in the present section have been designed to overcome the drawbacks of the sorted list and ND-Tree. With this aim, two objectives have been considered:

1. Minimization of the number of dominance comparisons necessary to check the acceptance/rejection of a candidate solution. For this purpose, *self-balancing trees* are used.
2. Minimization of the computational cost necessary to maintain the properties of the structures, in particular the balance property previously stated.

In general, an archive with strong properties constraining its structure will induce an onerous structure maintenance but a low number of dominance comparisons. On the contrary, a weakly constrained structure will tend to a high number of dominance comparisons. Therefore, both objectives are conflicting.

The general idea is to favor the first objective, while constraining the budget of the structure maintenance. The first proposed data structure is a self-balancing binary search tree confined to the bi-objective case, called *AVL-Archive*. The second proposed data structure is a self-balancing k-ary search tree, called *NDR*-Archive*.

For both data structures, two different versions are presented: the *vanilla* version and the *self-adjusting* version. The vanilla versions of both AVL-Archive and NDR*-Archive are general and suitable for any MO optimization task. The self-adjusting versions are especially designed if a presumption of *temporal and spatial locality* exists between the points generated by the MO optimization method. This is the case of PLS, a method we are particularly interested in, because of two main reasons:

• A solution and its neighbors found by PLS are generally close to each other in the objective space.

• The neighborhood of a solution is completely explored before considering the exploration of the neighborhood of another solution.

This chapter is organized as follows. Sections 3.1 and 3.2 detail the design of the proposed data structures and their self-adjusting versions. Section 3.3 is devoted to experiments: the vanilla version of the proposed archives are tested on a large artificial benchmark and their self-adjusting versions are applied within PLS on MOTSP.

3.1 AVL-Archive

An *AVL-tree* [Adelson-Velskii, 1962] is a *self-height-balancing binary search tree* used for storing and managing a number of elements indexed with a scalar key on a single comparison dimension. This tree is characterized by the three following properties:

[1] https://en.wikipedia.org/wiki/Locality_of_reference

▷ As an *indexing structure*, the tree handles the following operations: searching, inserting and removing *a single element*.

▷ As a *binary search tree*:

- any node of the tree has at most two children and each node contains a unique indexed element;
- given a node η of the tree and x the element stored in η: any element stored in the left subtree of η has a smaller key value than x; any element stored in the right subtree of η has a larger key value than x.

▷ As a *self-balancing* structure:

- any node η maintains an attribute called *balance factor*, which is the difference between the height of the right and the left subtrees of η;
- the tree is maintained *balanced*, meaning that the balance factor of any node is maintained in $\{-1, 0, 1\}$. This property directly implies that the height of an AVL-tree is maintained to $O(\log_2 s)$, where s is the number of nodes in the tree.
  To remain balanced, an AVL-tree re-balances any subtree unbalanced after inserting or removing *a single node*. Since the balance factor of any node is maintained in $\{-1, 0, 1\}$, when a subtree is unbalanced, the balance factor of its root is -2 or 2, and it is re-balanced through an operation called *rotation*.

  > A *rotation* is a reorganization of the nodes of a subtree such that parts of the left/right-side of the subtree are transferred to the other side. At the end of a rotation, the root of the subtree has changed and the subtree is balanced.

In [Adelson-Velskii, 1962], the authors distinguish only four possible cases of imbalance of a subtree and propose a unique rotation for each case. Once a node η is inserted or removed, we check if its parent node is balanced and apply a rotation if it does not. When performed, a rotation may propagate the imbalance upward in the tree, thus several rotations might be necessary until the complete tree is balanced.

This section introduces *AVL-Archive*, the new archive we propose for the bi-objective case and based on AVL-tree. In the remaining of this section, we assume the objective space is bi-dimensional, i.e. $p = 2$.

3.1.1 Design of AVL-Archive

Like AVL-tree, AVL-Archive is a self-height-balancing binary search tree using four different rotations to stay balanced. By contrast, an AVL-Archive is especially designed for archiving. Second, it uses interesting properties exclusive to the bi-objective case to store incomparable solutions and efficiently remove dominated solutions. Third, it may happen that complete subtrees are removed; the archive must then be able to re-balance subtrees related to an imbalance factor with an absolute value larger than 2, which is not the case of AVL-tree. Let us now describe more precisely how an AVL-Archive works.

Each node η of an AVL-Archive (cf. Figure 3.1) is associated with:

- a solution $x(\eta) \in X$. In the following, for the sake of simplicity, we say that a solution $\overline{x}$ (resp. weakly) dominates a node η if $f(\overline{x})$ (resp. weakly) dominates $f(x(\eta))$;

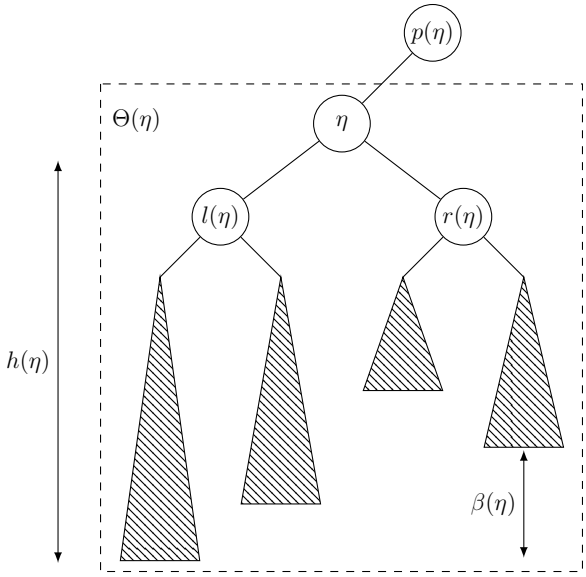


Figure 3.1 – Illustration of a node η with its attributes in an AVL-Archive.

- its parent node $p(\eta)$
- its two children: the left child $l(\eta)$ and the right child $r(\eta)$;
- the subtree $\Theta(\eta)$ rooted at η;
- its height $h(\eta)$ (also noted $h(\Theta(\eta))$), equal to the size of the path from η to the most distant leaf of $\Theta(\eta)$;
- its *balance factor* $\beta(\eta)$, corresponding to the difference between the height of its right subtree and the height of its left subtree:

$$\beta(\eta) = h(r(\eta)) - h(l(\eta))$$

  Let $|\beta(\eta)|$ be called the *absolute balance factor*.

Given these notations, the left (resp. right) subtree of a node η is denoted $\Theta(l(\eta))$ (resp. $\Theta(r(\eta))$).

Definition 3.1.1. A node η is said to be balanced if $|\beta(\eta)| \leq 1$. A tree is said to be balanced iff. each of its node is balanced.

The structural organization of AVL-Archive is defined by the two following fundamental properties.

Property 3.1.1. Any subtree of an AVL-Archive is a binary search tree using the first objective as comparison dimension. More formally, for any node η, for any node $\eta_l \in \Theta(l(\eta))$, we have $f_1(x(\eta_l)) < f_1(x(\eta))$; and for any node $\eta_r \in \Theta(r(\eta))$, we have $f_1(x(\eta)) < f_1(x(\eta_r))$.

The next proposition is complementary to Property 3.1.1.

Proposition 3.1.1. Let η be a node. For any node $\eta_l \in \Theta(l(\eta))$, we have $f_2(x(\eta_l)) > f_2(x(\eta))$; and for any node $\eta_r \in \Theta(r(\eta))$, we have $f_2(x(\eta)) > f_2(x(\eta_r))$.

Proof. By Property 3.1.1, we have $f_1(x(\eta_l)) < f_1(x(\eta))$. Since x is incomparable with any solution of the archive and $p = 2$, we have $f_2(x(\eta_l)) > f_2(x(\eta))$. The proof is similar for $\Theta(r(\eta))$. □

Property 3.1.1 and Proposition 3.1.1 combined indicate that for any node η of the tree:

- solutions in the left subtree of η have a smaller value than $x(\eta)$ in the first objective and a larger value than $x(\eta)$ in the second objective.
- solutions in the right subtree of η have a larger value than $x(\eta)$ in the first objective and a smaller value than $x(\eta)$ in the second objective.

Property 3.1.2. Any subtree of an AVL-Archive is maintained balanced in height. More formally, for any node η, $|\beta(\eta)| \leq 1$.

Proposition 3.1.2. The height of an AVL-Archive is maintained to $O(\log_2 s)$, where s is the number of nodes in the tree.

Proof. Obvious from Property 3.1.2. □

An AVL-Archive works as follows. When a candidate solution $\overline{x} \in X$ is presented to the archive, a dichotomic search is performed from the root of the tree. If the candidate is weakly dominated by a node, then it is rejected. Otherwise, if the candidate dominates a node, then the candidate replaces the solution of the node, dominated subtrees are removed through a procedure called *pruning*, and the candidate is accepted. Otherwise the candidate is incomparable with any solution of the archive, thus it is inserted in a new leaf node and is accepted.

In both cases where the candidate is accepted, node insertions and node removals may unbalance affected subtrees. The imbalance of a subtree is detected when its root has an absolute balance factor greater than or equal to 2. Such subtrees are re-balanced using rotations from their root. Because a rotation performed from a node may propagate the imbalance upward in the tree, several rotations might be necessary until the complete tree is balanced (as in AVL-tree).

3.1.1.1 General algorithm of AVL-Archive

Let us now describe more precisely the AVL-Archive algorithm. The `Add` procedure is the core of an AVL-Archive. It is described with a diagram in Figure 3.2 and detailed in pseudo-code by Algorithm 12. This procedure takes as parameter a calling node η and a candidate solution $\overline{x} \in X$. It is initially called from the root of the tree. It returns true iff. $\overline{x}$ is not weakly dominated by any solution in the archive.

First, $\overline{x}$ is compared to the solution of the current node η:

- If $\overline{x}$ is weakly dominated by $x(\eta)$ (lines 1-2), then the candidate $\overline{x}$ is rejected.
- Otherwise, if $\overline{x}$ dominates $x(\eta)$ (lines 3-18), then $\overline{x}$ replaces it. A number of nodes in the current subtree may be dominated and thus have to be removed. We call this process *pruning*. First, both left and right subtrees of η are disconnected from η (lines 6 and 12). Second, they are *independently* pruned via the `LeftPruning` and `RightPruning` procedures (lines 7 and 13). Third, if the subtrees become unbalanced, they are (independently) re-balanced with rotations via the `ReBalanceAfterDelete` procedure (lines 9 and 15). As both subtrees have been previously disconnected from η, the re-balancing procedure does not operate upward their respective root. Fourth, both subtrees are reconnected to η (lines 10 and 16), then the whole tree is re-balanced by calling `ReBalanceAfterDelete` procedure from η (lines 17-18). Finally the candidate is accepted.

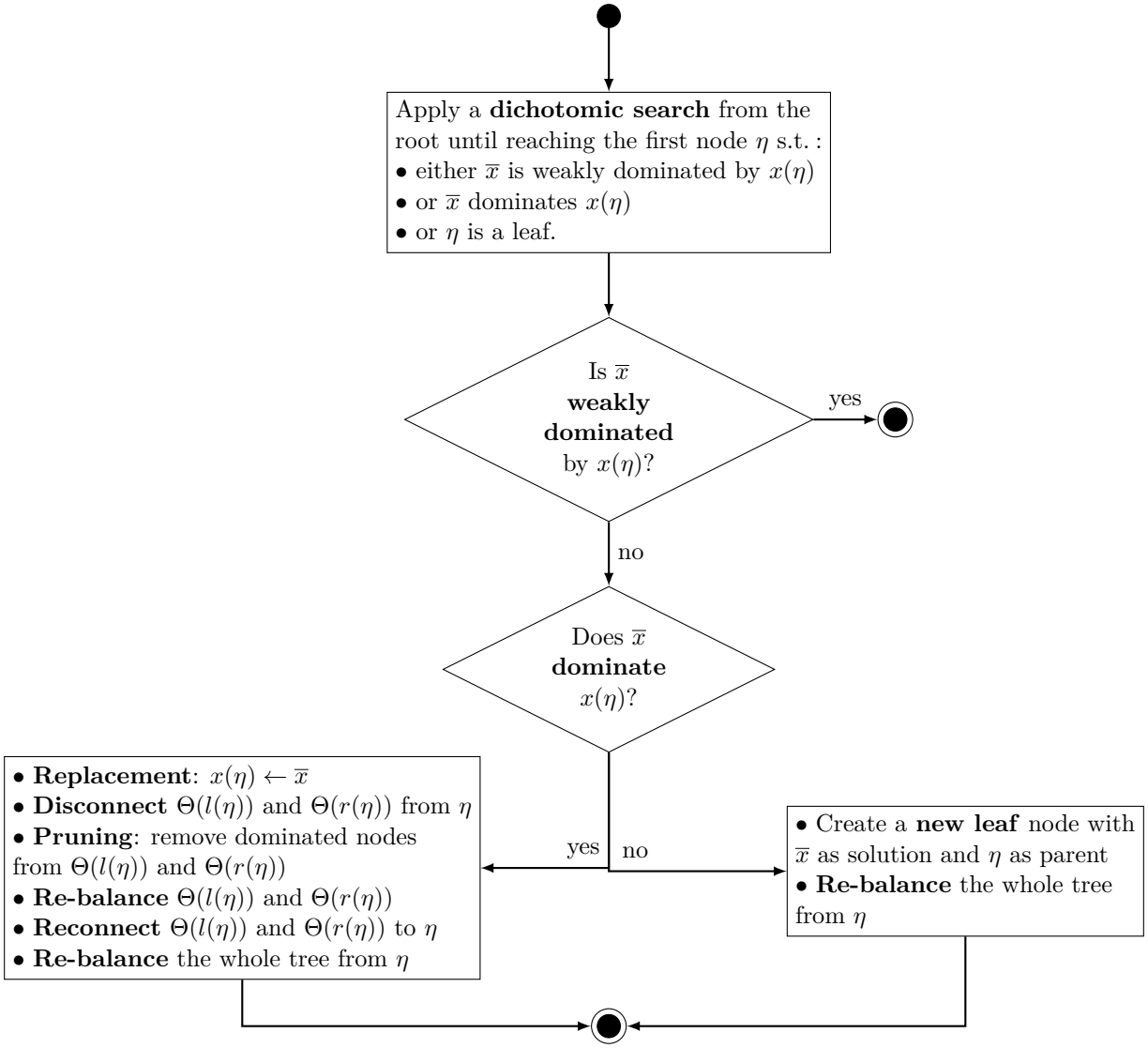


Figure 3.2 – The `Add` procedure of an AVL-Archive with a candidate solution $\overline{x} \in X$.

Algorithme 12 : `Add`

Input : calling node η, candidate solution $\overline{x}$
Output : boolean

```
1  if f(x(η)) ≦ f(x̄) then
2      return false
3  else if f(x̄) ≤ f(x(η)) then
4      x(η) ← x̄
5      if η has a left child then
6          p(l(η)) = none
7          uη ← LeftPruning(l(η), l(η), x̄)
8          if a node has been deleted from Θ(l(η)) then
9              lη ← ReBalanceAfterDelete(uη)
10             l(η) ← lη; p(lη) ← η
11     if η has a right child then
12         p(r(η)) = none
13         uη ← RightPruning(r(η), r(η), x̄)
14         if a node has been deleted from Θ(r(η)) then
15             rη ← ReBalanceAfterDelete(uη)
16             r(η) ← rη; p(rη) ← η
17     if a node has been deleted from Θ(l(η)) or Θ(r(η)) then
18         root ←ReBalanceAfterDelete(η)
19 else if f_1(x̄) < f_1(x(η)) then
20     if η has a left child then
21         return Add(l(η), x̄)
22     else
23         l(η) ← new node(x̄, η)
24         ReBalanceAfterInsert(η)
25 else
26     if η has a right child then
27         return Add(r(η), x̄)
28     else
29         r(η) ← new node(x̄, η)
30         ReBalanceAfterInsert(η)
31 return true
```

- Otherwise (lines 19-30), x and $x(\eta)$ are incomparable:
  - If $f_1(\overline{x}) < f_1(x(\eta))$, the `Add` procedure is recalled from the left child of η.
  - Otherwise, the `Add` procedure is recalled from the right child of η .

    In both cases, if there is no child, x is inserted in a new node and the whole tree is re-balanced by calling the `ReBalanceAfterInsert` procedure from its parent, which performs a number of rotations, then the candidate is accepted.

3.1.1.2 Pruning process

Let us assume that during the dichotomic search of the `Add` procedure, a node η is found such that the candidate solution $\overline{x}$ dominates η. Therefore, $x(\eta)$ is replaced by $\overline{x}$, then $\Theta(l(\eta))$ and $\Theta(r(\eta))$ are disconnected from η. This triggers the pruning process which aims at removing dominated nodes in $\Theta(l(\eta))$ and $\Theta(r(\eta))$ by starting independently from $l(\eta)$ and $r(\eta)$.

Proposition 3.1.3. Let $\overline{x} \in X$ be a candidate solution, η a node, $x = x(\eta)$, $\eta_l \in \Theta(l(\eta))$, $x_l = x(\eta_l)$, $\eta_r \in \Theta(r(\eta))$ and $x_r = x(\eta_r)$ such that $f(\overline{x}) \leq f(x)$.

(1) (i) If $f_1(\overline{x}) \leq f_1(x_l)$, then $f(\overline{x}) \leq f(x_l)$ and $f(\overline{x}) \leq f(x')$ for any solution x' in $\Theta(r(\eta_l))$

(ii) Otherwise, $f(\overline{x}) \not\leq f(x_l)$ and $f(\overline{x}) \not\leq f(x'')$ for any solution x'' in $\Theta(l(\eta_l))$.

(2) (i) If $f_2(\overline{x}) \leq f_2(x_r)$, then $f(\overline{x}) \leq f(x_r)$ and $f(\overline{x}) \leq f(x')$ for any solution x' in $\Theta(l(\eta_r))$.

(ii) Otherwise, $f(\overline{x}) \not\leq f(x_r)$ and $f(\overline{x}) \not\leq f(x'')$ for any solution x'' in $\Theta(r(\eta_r))$.

Proof. (1) (i) From Proposition 3.1.1, we have $f_2(x) < f_2(x_l)$ which involves $f_2(\overline{x}) < f_2(x_l)$ and thus $f(\overline{x}) \leq f(x_l)$ since $f_1(\overline{x}) \leq f_1(x_l)$. Moreover, since x' is in the right subtree of η_l we have $f_1(x_l) \leq f_1(x')$, which gives $f_1(\overline{x}) \leq f_1(x')$ since $f_1(\overline{x}) \leq f_1(x_l)$. Finally, since x' is in the left subtree of η, we have $f_2(x) < f_2(x')$, which gives $f_2(\overline{x}) \leq f_2(x')$ since $f(\overline{x}) \leq f(x)$. It follows that $f(\overline{x}) \leq f(x')$.

(1) (ii) we have $f_1(x_l) < f_1(\overline{x})$ which involves $f_1(x'') < f_1(\overline{x})$ since x'' is in the left subtree of η_l, thus $f(\overline{x}) \not\leq f(x_l)$ and $f(\overline{x}) \not\leq f(x'')$.

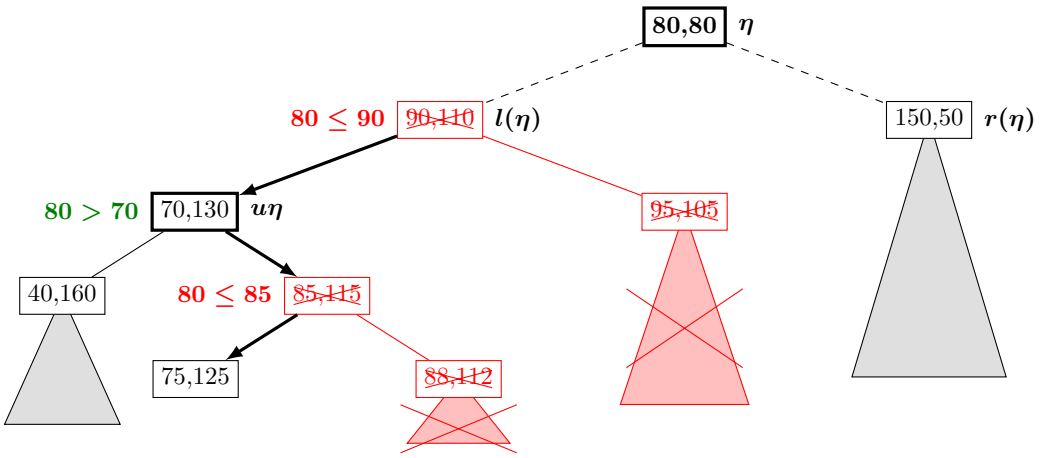
The proof is similar for (2). □

Proposition 3.1.3 is fundamental for the pruning process. It states that when $\overline{x}$ dominates η, then if a node η_l (resp. η_r) in the left (resp. right) subtree of η:

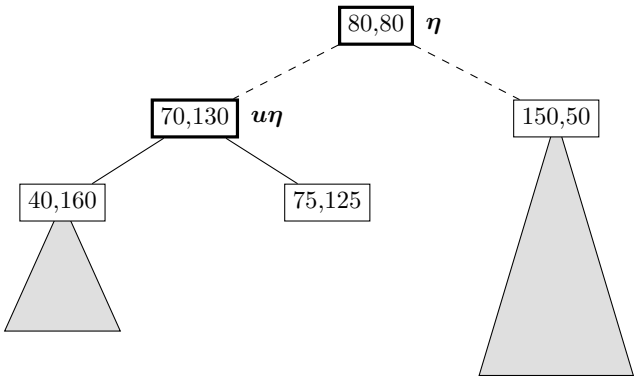
- is dominated by $\overline{x}$, then both η_l (resp. η_r) and its whole right (resp. left) subtree are dominated, and thus can be removed.
- is not dominated by $\overline{x}$, then $\overline{x}$ does not dominate any node in the left (resp. right) subtree of η_l (resp. η_r).

The pruning process is implemented by the `LeftPruning` and `RightPruning` procedures. These procedures are symmetrical as the `LeftPruning` aims at pruning the left subtree of η and is initially called from $l(\eta)$, while the `RightPruning` aims at pruning the right subtree of η and is initially called from $r(\eta)$.

Let us only describe the `LeftPruning` procedure, depicted with a diagram in Figure 3.4 and in pseudo-code by Algorithm 13. Figure 3.3 illustrates an example of execution of the `LeftPruning` procedure on a subtree of an AVL-Archive. The `LeftPruning` procedure is recursive and takes as input parameters a calling node $\eta_l \in \Theta(l(\eta))$, the deepest node of the subtree to re-balance $u\eta \in \Theta(l(\eta))$ and the candidate solution $\overline{x}$.



(a) During pruning of $\Theta(l(\eta))$.



(b) After pruning of $\Theta(l(\eta))$.

Figure 3.3 – Example of execution of the `LeftPruning` procedure. The node η is s.t. $x(\eta)$ has just been replaced by a candidate solution $\overline{x} \in X$ s.t. $f(\overline{x}) = (80, 80)$, and $\Theta(l(\eta))$ and $\Theta(r(\eta))$ have just been disconnected from η. Then, `LeftPruning` procedure is started from $l(\eta)$ and follows a unique top-down path (indicated in bold edges) until reaching a leaf node. The procedure finally returns the node $u\eta$ which is the deepest node to re-balance. The symmetrical process is performed in $\Theta(r(\eta))$ via the `RightPruning` procedure but is not detailed.

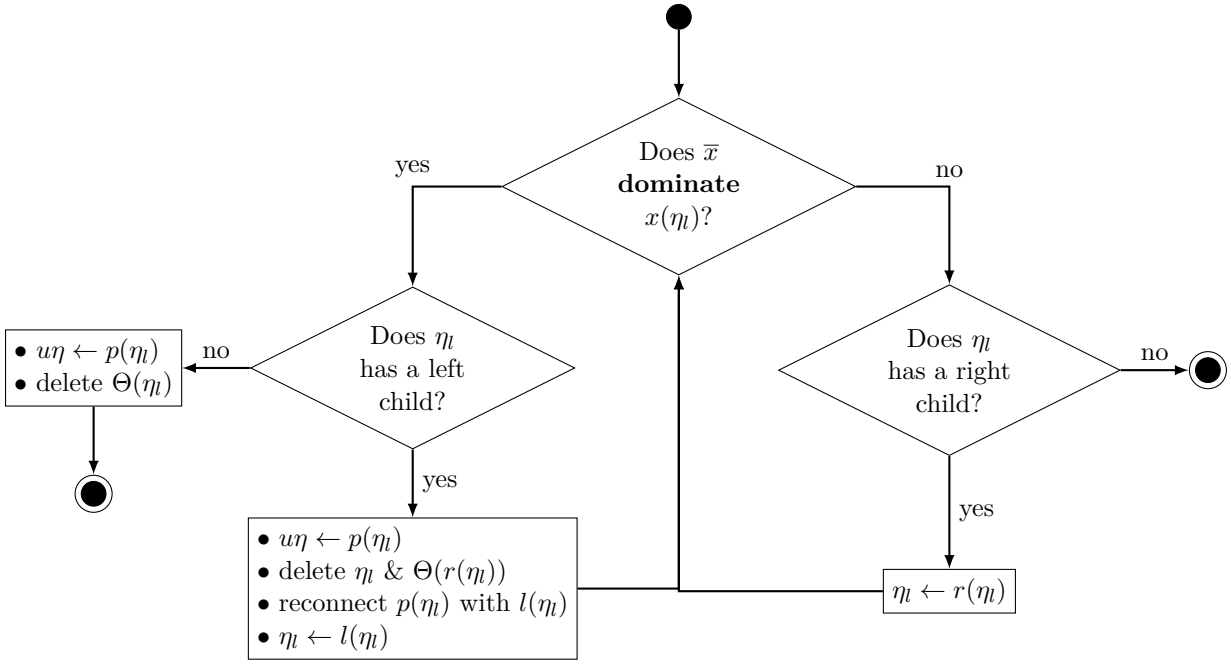


Figure 3.4 – The `LeftPruning` procedure of an AVL-Archive called from a node $\eta_l \in \Theta(l(\eta))$ with a candidate solution $\overline{x} \in X$.

Algorithme 13 : `LeftPruning`

Input : calling node η_l, deepest node to re-balance $u\eta$, candidate solution $\overline{x}$
Output : the deepest node to re-balance

```
1  if f1(x̄) ≤ f1(x(η_l)) then
       // x̄ dominates both x(η_l) & Θ(r(η_l))
2      uη ← p(η_l)
3      if η_l has a left child then
4          replace η_l by l(η_l) as child of p(η_l)
5          delete η_l and Θ(r(η_l))
6          return LeftPruning(l(η_l), uη, x̄)
7      else
8          delete Θ(η_l)
9          return uη
10 else
       // x̄ does not dominate both x(η_l) & Θ(l(η_l))
11     if η_l has a right child then
12         return LeftPruning(r(η_l), uη, x̄)
13     else
14         return uη
```

It returns the deepest node to re-balance. It uses the result pointed out by Proposition 3.1.3 to remove efficiently from $\Theta(l(\eta))$ all nodes dominated by $\overline{x}$.

More precisely, the procedure explores $\Theta(l(\eta))$ by following a *single top-down path* from the root of the subtree to a leaf node. Given Proposition 3.1.3, at each current node $\eta_l \in \Theta(l(\eta))$, only two cases are possible:

- If $f_1(\overline{x}) \leq f_1(x(\eta_l))$, then $\overline{x}$ dominates η_l and all nodes in $\Theta(r(\eta_l))$, thus they are removed
  - if η_l has a left child, then the subtree is recomposed by replacing η_l by $l(\eta_l)$ as child of $p(\eta_l)$, and the search continues down in $\Theta(l(\eta_l))$;
  - otherwise, the search stops.

  In both cases, the currently deepest node to re-balance is $p(\eta_l)$ as the right subtree of η_l has just been removed and thus can cause an imbalance from $p(\eta_l)$.

- Otherwise, $\overline{x}$ does not dominate η_l and any node in $\Theta(l(\eta_l))$, thus:
  - if η_l has a right child, then the search continues in $\Theta(r(\eta_l))$;
  - otherwise, the search stops.

To summarize, at the end of the left and right pruning processes (independently performed), both left and right subtrees of η have been emptied of all dominated nodes and for each subtree, the deepest node to re-balance has been detected. All that remains to do is to re-balance both subtrees from their respective detected nodes (if necessary), then reconnect the subtrees to η, and finally re-balance the whole tree from η (if necessary).

3.1.1.3 Re-balancing process

The re-balancing process is called by the `Add` procedure in two distinct cases:

- once a new leaf node is inserted;
- once a node/subtree is deleted.

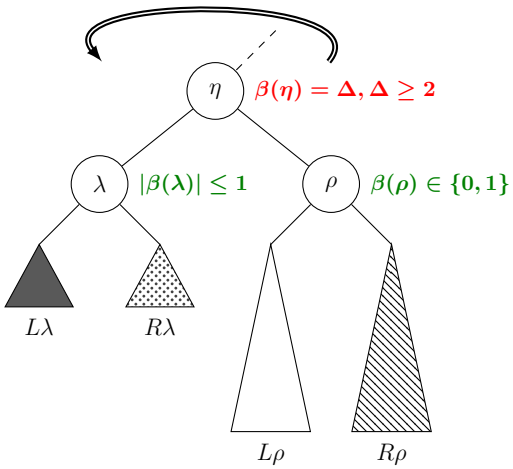
In any case, the deepest node affected by the modification is detected and re-balanced with a rotation if necessary. Because a rotation may propagate the imbalance on the ancestors of the node *only* (cf. [Adelson-Velskii, 1962, Knuth, 1998]), the tree is traveled up from this node and all imbalanced ancestors are re-balanced with rotations.

Next, the four different rotations introduced in [Adelson-Velskii, 1962] are detailed. There are four different types of rotations: simple left, simple right, double left-right and double right-left.

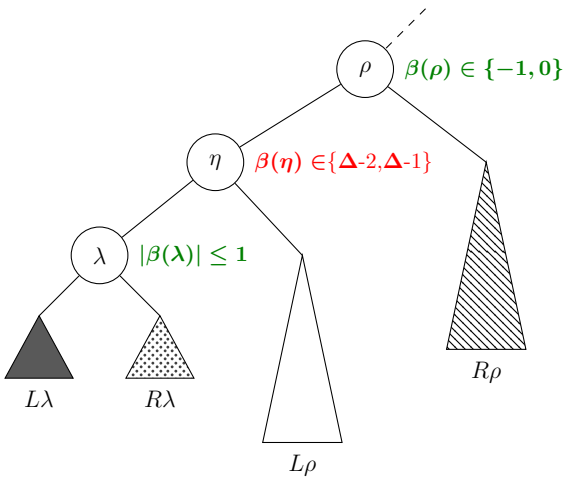
Definition 3.1.2. Let η be a node and ρ its right child. A **simple left rotation** rooted at η consists in ascending ρ such that η becomes the left child of ρ, and the left child of ρ becomes the right child of η. The new root of the subtree affected by the rotation is ρ.

Figure 3.5 illustrates a simple left rotation.

Definition 3.1.3. Let η be a node and λ its left child. A **simple right rotation** rooted at η consists in ascending λ such that η becomes the right child of λ, and the right child of λ becomes the left child of η. The new root of the subtree affected by the rotation is λ.



(a) An unbalanced subtree rooted at η. Subtrees $\Theta(\lambda)$, $L\rho$ and $R\rho$ are balanced. A left rotation rooted at η is performed.



(b) The final balanced tree rooted at ρ.

Figure 3.5 – Illustration of a simple left rotation.

Definition 3.1.4. Let η be a node, ρ its right child and μ the left child of ρ. A **double left-right rotation** rooted at η is a two-step rotation. First, it consists in applying a right rotation rooted at ρ, then a left rotation rooted at η. The new root of the subtree affected by the rotation is μ.

Figure 3.6 illustrates a double left-right rotation.

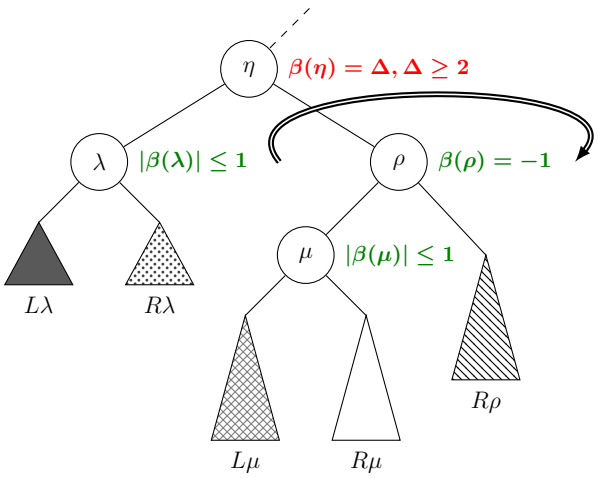
Definition 3.1.5. Let η be a node, λ its left child and μ the right child of λ. A **double right-left rotation** rooted at η is a two-step rotation. First, it consists in applying a left rotation rooted at λ, then a right rotation rooted at η. The new root of the subtree affected by the rotation is μ.

Algorithm 14 depicts a regular rotation rooted at a node η. The procedure performs a simple or a double rotation, if necessary, and returns the root of the affected subtree and a boolean indicating if a rotation has effectively been performed or not. It uses the `LeftRotation` and `RightRotation` procedures (the pseudo-code of these procedures is straightforward and thus not given), which proceed to a simple left and right rotation, respectively, update the height and balance attributes, and return the root of the affected subtree.

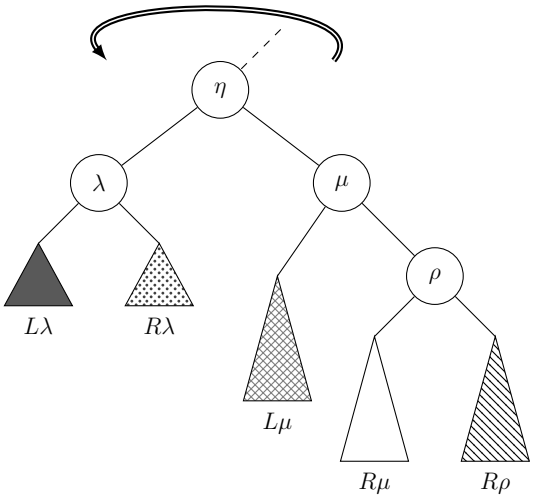
The general situation faced by an AVL-Archive when a node is unbalanced, is introduced by the following proposition and illustrated by Figure 3.7.

Proposition 3.1.4. Let Θ be a subtree and η its root, $\lambda = l(\eta)$ and $\rho = r(\eta)$. The node η is unbalanced s.t. $\beta(\eta) = \Delta$ where $|\Delta| \geq 2$. Let us assume that the rest of Θ is balanced: $|\beta(\lambda)| \leq 1$, $|\beta(\rho)| \leq 1$ and the subtrees $L\lambda = \Theta(l(\lambda))$, $R\lambda = \Theta(r(\lambda))$, $L\rho = \Theta(l(\rho))$ and $R\rho = \Theta(r(\rho))$ are balanced.

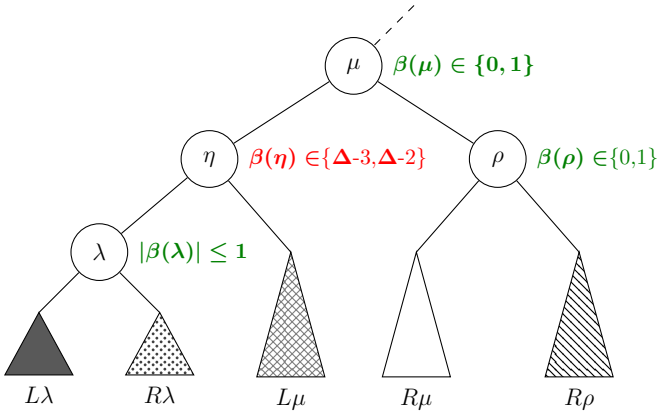
1. If $\Delta \geq 2$ and $\beta(\rho) \in \{0, 1\}$, then after a **simple left rotation** rooted at η, the root of Θ is ρ, $|\beta(\rho)| \leq 1$, $\beta(\eta) < \Delta$ and the height of Θ is unchanged or has strictly decreased.
2. If $\Delta \leq -2$ and $\beta(\lambda) \in \{-1, 0\}$, then after a **simple right rotation** rooted at η, the root of Θ is λ, $|\beta(\lambda)| \leq 1$, $\beta(\eta) > \Delta$ and the height of Θ is unchanged or has strictly decreased.
3. Let μ be the root of $L\rho$.
   If $\Delta \geq 2$ and $\beta(\rho) = -1$, then after a **double left-right rotation** rooted at η, the root of Θ is μ, $|\beta(\mu)| \leq 1$, $|\beta(\rho)| \leq 1$, $\beta(\eta) < \Delta$, and the height of Θ has strictly decreased.



(a) An unbalanced subtree rooted at η.
Subtrees $\Theta(\lambda)$, $L\mu$, $R\mu$ and $R\rho$ are balanced.
A right rotation rooted at ρ is performed.



(b) The temporary tree.
A left rotation rooted at η is performed.



(c) The final balanced tree rooted at μ.

Figure 3.6 – Illustration of a double left-right rotation.

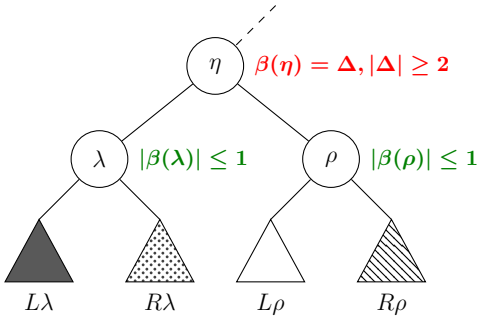


Figure 3.7 – Illustration of the general situation faced when a node η is unbalanced.

Algorithme 14 : `Rotation`

Input : calling node η
Output : root of the subtree, boolean

```
update h(η) and β(η)
if β(η) ≥ 2 then
    // Left rotation:
    if β(r(η)) = −1 then
        RightRotation(r(η))
    return (LeftRotation(η), true)
else if β(η) ≤ −2 then
    // Right rotation:
    if β(l(η)) = 1 then
        LeftRotation(l(η))
    return RightRotation(η), true)
else
    // No rotation performed:
    return (η, false)
```

4. Let μ be the root of $R\lambda$.
If $\Delta \leq -2$ and $\beta(\lambda) = 1$, then after a **double right-left rotation** rooted at η, the root of Θ is μ, $|\beta(\mu)| \leq 1$, $|\beta(\lambda)| \leq 1$, $\beta(\eta) > \Delta$ and the height of Θ has strictly decreased.

In any case (1)-(4), after the rotation:

(I) If $\Delta = 2$, then Θ is balanced, i.e. Property 3.1.2 is satisfied.

(II) the archive is still a binary search tree, i.e. Property 3.1.1 is still satisfied.

Proof. (1) The proof is illustrated by Figure 3.5. Let $\delta \in \{0, 1\}$ such that $h(R\rho) = h(L\rho) + \delta$.

Before the rotation, the height of Θ is determined by the right subtree since the balance factor of the root is positive. Thus the height of Θ is equal to $h(R\rho) + 2$.

After the rotation, the height of Θ is equal to $h(L\rho)+2 = h(R\rho)+2-\delta$, so the height of Θ has decreased or is unchanged. We have $\beta(\eta) = h(L\rho)-h(\lambda) = h(L\rho)-(h(L\rho)+\delta+1-\Delta) = \Delta-1-\delta$, so $\beta(\eta) \in \{\Delta-2, \Delta-1\}$. We have $\beta(\rho) = h(R\rho) - h(\eta) = \delta - 1$, so $\beta(\rho) \in \{-1, 0\}$.

(2) This case is symmetrical to (1).

(3) The proof is illustrated by Figure 3.6. Let $\mu = l(\rho)$, $L\mu = \Theta(l(\mu))$, $R\mu = \Theta(r(\mu))$, $max := \max\{h(L\mu); h(R\mu)\}$ and $min = \min\{h(L\mu); h(R\mu)\}$.

Before the rotation, the height of Θ is equal to $max + 3$. There is no assumption on the balance factor of μ, so $(max - min) \in \{0, 1\}$. We have $h(\mu) = max + 1$ and given that $\beta(\rho) = -1$, $h(R\rho) = h(\mu) - 1 = max$. Given that $\beta(\eta) = \Delta$, we have $h(\lambda) = h(\rho) - \Delta = max + 2 - \Delta$ thus $h(\lambda) \leq max$ since $\Delta \geq 2$.

After the rotation, the height of Θ is equal to $max + 2 < max + 3$, so the height of Θ has decreased. Two cases appear:

- If we assume that $h(L\mu) = max$, then $h(R\mu) = min$ and $\beta(\rho) = h(R\rho) - h(R\mu) = max - min$ since $h(R\rho) = max$, so $\beta(\rho) \in \{0, 1\}$. We have also $\beta(\eta) = h(L\mu) - h(\lambda) = \Delta - 2$ since $h(\lambda) = max + 2 - \Delta$. Moreover, $\beta(\mu) = h(\rho) - h(\eta) = (max+1) - (max+1) = 0$ since $h(\eta) = 1 + \max\{h(L\mu); h(\lambda)\} = max+1$ because $h(\lambda) \leq max$.
- Otherwise, if $h(L\mu) = min$, then $h(R\mu) = max$ and $\beta(\rho) = max - max = 0$ since $h(R\rho) = max$. We have also $\beta(\eta) = h(L\mu) - h(\lambda) = min - max - 2 + \Delta$ since $h(\lambda) = max + 2 - \Delta$, so $\beta(\eta) \in \{\Delta - 3, \Delta - 2\}$. Moreover, $\beta(\mu) = h(\rho) - h(\eta) = (max + 1) - (1 + \max\{h(L\mu); h(\lambda)\}) \in \{0, 1\}$ since $\max\{h(L\mu); h(\lambda)\} \in \{min, max\}$ because $h(\lambda) \leq max$.

So $\beta(\eta) \in \{\Delta - 3, \Delta - 2\}$ and $\beta(\mu) \in \{0, 1\}$ in both cases.

(4) This case is symmetrical to (3).

(I) Obvious from proofs of (1) and (3). See [Adelson-Velskii, 1962] for another proof.

(II) Obvious from Definitions 3.1.2, 3.1.3, 3.1.4 and 3.1.5.

□

Proposition 3.1.4 distinguishes four distinct cases of imbalance, and to each case is assigned a specific rotation.

Let us first consider the **insertion case**, in which a new node η_{new} is inserted in the tree after a candidate solution incomparable with all the solutions of the archive has been accepted. This case is already supported by the original AVL-tree algorithm [Adelson-Velskii, 1962] and the pseudo-code is depicted in Algorithm 15:`ReBalanceAfterInsert`. This method travels up the tree starting from the parent node $\eta = p(\eta_{new})$ and searches the first unbalanced descendant of η_{new}. As the absolute balance factor of any node is maintained inferior or equal to 1, the absolute balance factor of such a node is 2. A single rotation rooted at this node is then performed and re-balances the whole affected subtree, given Proposition 3.1.4. Given [Adelson-Velskii, 1962], the insertion of a new node unbalances *at most one descendant,* thus the search stops once such a descendant has been re-balanced. Moreover, the search can be prematurely stopped if the balance factor of the current visited descendant is null.

Algorithme 15 : `ReBalanceAfterInsert`

Input : parent η of newly inserted node
Output : $\emptyset$

```
1 rotated ←false
2 repeat
3 |   (ϱ, rotated) ←Rotation(η)
4 |   η ← p(ϱ)
5 until η = none or β(η) = 0 or rotated;
```

Now consider the **deletion case**. As previously suggested in Section 3.1.1.2, when a candidate solution dominates a node η:

- Both left and right subtrees of the dominated node $\Theta(l(\eta))$ and $\Theta(r(\eta))$ are first disconnected to η, then pruned, and the deepest node to re-balance of each subtree is identified. For each subtree: if

one node at least has been removed, then the deepest node to re-balance $u\eta$ and any of its ascendants might be unbalanced; thus a re-balancing procedure is called from $u\eta$.

- Once these subtrees have been reconnected to η, if one node at least has been removed from one of the subtrees, then η and any of its ascendants might be unbalanced; thus a re-balancing procedure is called from η.

Because whole subtrees can be removed during the pruning process, the absolute balance factor of a number of nodes may exceed 2. To our knowledge, such serious imbalance has not been taken into account by the original re-balancing procedure of AVL-tree, thus we generalized it.

The **generalized re-balancing procedure** works as follows. Let η be the node to re-balance. Given Proposition 3.1.4 and as illustrated by Figures 3.5 and 3.6, applying a rotation on η *pushes down the imbalance* by descending η and *strictly improving its absolute balance factor* but kept unbalanced if $|\beta(\eta))| > 2$, and such that the rest of the nodes are still balanced. From this observation, we propose the concept of *rotation chain*, which consists in performing successive rotations *on the same node* η until $|\beta(\eta)| \leq 1$.

Algorithme 16 : `RotationChain`

Input : calling node η
Output : new root of the subtree

```
1 (ϱ, ·) ←Rotation(η)
2 while |β(η)| ≥ 2 do
3 |   Rotation(η)
4 return ϱ
```

Algorithm 16 presents a rotation chain starting from a node η. First, the root ϱ of the subtree *after the first rotation* is memorized as it is the root of the subtree once the rotation chain is finished. Then, the imbalance is iteratively pushed down. The method returns the new subtree root ϱ. Note that if η is initially balanced, the subtree is unchanged.

Proposition 3.1.5. Let $\Theta(\eta)$ be a tree rooted at η such that $|\beta(\eta)| = \Delta$, $\Delta \geq 2$ and both $\Theta(l(\eta))$ and $\Theta(r(\eta))$ are balanced. If a rotation chain is performed from η, then $\Theta(\eta)$ is balanced after applying at most $\Delta - 1$ rotations (i.e. Property 3.1.2 is satisfied).

Proof. This result is a direct consequence of Proposition 3.1.4. As a rotation strictly improves the absolute balance factor of η of at least 1, then a rotation chain always terminates after at most $\Delta - 1$ rotations so that $|\beta(\eta)| \leq 1$ and the whole tree is balanced. □

Algorithme 17 : `ReBalanceAfterDelete`

Input : calling node η
Output : root of the subtree

```
1 repeat
2 |   ϱ ←RotationChain(η)
3 |   η ← p(ϱ)
4 until η = none;
5 return η
```

Algorithm 17:`ReBalanceAfterDelete` implements the generalized re-balancing procedure starting from a node η. The procedure travels up the tree from η and applies a rotation chain to each ascendant. *Any*

ascendant can be unbalanced after a deletion, thus the search cannot be prematurely stopped (as in the insertion case) until reaching the root of the tree.

Figure 3.8 illustrates an example of the insertion of a candidate solution into an AVL-Archive.

Proposition 3.1.6. The complexity in time of the `Add` procedure is in $O(\log_2^2 s)$, where s is the number of nodes in the AVL-Archive.

Proof. Given that an AVL-Archive is a balanced binary search tree, its height is in $O(\log_2 s)$. Let $\overline{x}$ be the candidate solution presented to the archive.

First, a dichotomic search is performed from the root to a node η. For each traversed node, constant time operations are performed.

• If $x(\eta)$ weakly dominates $\overline{x}$, then the procedure is stopped. Therefore in this case, the complexity is in $O(\log_2 s)$.

• Otherwise, if $\overline{x}$ is incomparable with all traversed node, then a new node is created with $\overline{x}$ as solution. During the re-balancing process, the tree is traveled up from η and at most one rotation is performed, which is a constant time operation. Thus in this case, the complexity is in $O(\log_2 s)$.

• Otherwise, $\overline{x}$ dominates $x(\eta)$. Both left and right subtrees of η are pruned. During the pruning process, a single path is followed and for each traversed node, the operations are made in constant time, including the deletion operation. So the pruning process is also in $O(\log_2 s)$.

Concerning the re-balancing process, let us consider the worst case, which consists in re-balancing a tree formed by a path of $O(\log_2 s)$ nodes. Now assume that the deepest node to re-balance $u\eta$ is the last node, i.e. the unique leaf of the tree. The re-balancing process consists in traveling up the tree from $u\eta$ to the root, and thus traversing $O(\log_2 s)$ nodes. For each traversed node: a rotation chain is applied, performing at most $\Delta - 1$ rotations, where $\Delta \in O(\log_2 s)$ is the balance factor of the starting node of the rotation chain. Thus the re-balancing process is in $O(\log_2^2 s)$. □

3.1.2 Self-adjusting version

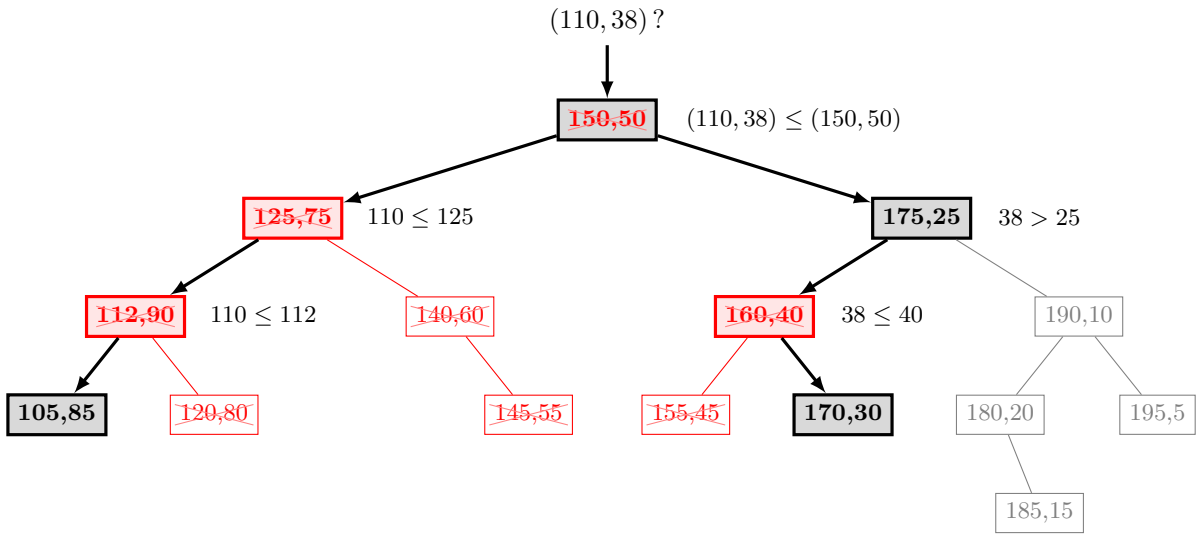
The Self-Adjusting version of an AVL-Archive, called SAAVLA, has a number of additional features compared to the vanilla version.

First, in addition to its usual attributes, a node η is associated with the local ideal $z^*(\eta)$ of the subtree rooted at η defined such that:

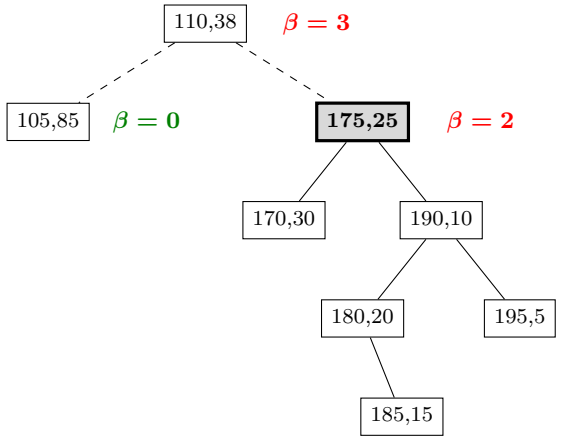
$$z_j^*(\eta) = \min\{f_j(x(\eta')) : \eta' \in \Theta(\eta)\} \text{ for } j = 1, 2$$

In fact, $z_1^*(\eta)$ (resp. $z_2^*(\eta)$) is the first (resp. second) objective value of the solution contained in the *left-most* (resp. *right-most*) *node* of $\Theta(\eta)$. The local ideal $z^*(\eta)$ is updated online and we do not mention these updates in pseudo-code as they are trivial. Indeed, each time a leaf is created or deleted:

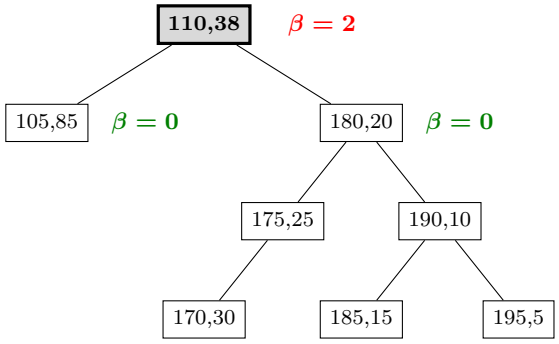
- if this leaf is a left child: the first objective value of its associated solution is propagated upward and is used to update $z_1^*(\eta')$, for each one of its ascendant η'.
- if this leaf is a right child: the second objective value of its associated solution is propagated upward and is used to update $z_2^*(\eta')$, for each one of its ascendant η'.



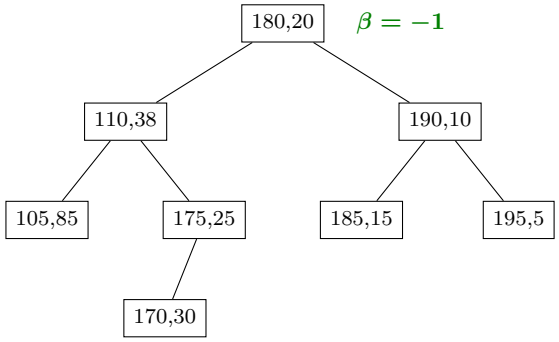
(a) $\overline{x}$ dominates the solution in the root $\Rightarrow$ $\overline{x}$ replaces the solution contained in the root; both left and right subtrees are pruned s.t. all dominated nodes are removed. The subtree rooted at $(190, 10)$ is not explored, and only 6 comparisons are necessary to remove 8 nodes.



(b) The subtree rooted at $(175, 25)$ is unbalanced $\Rightarrow$ a double left-right rotation is performed.

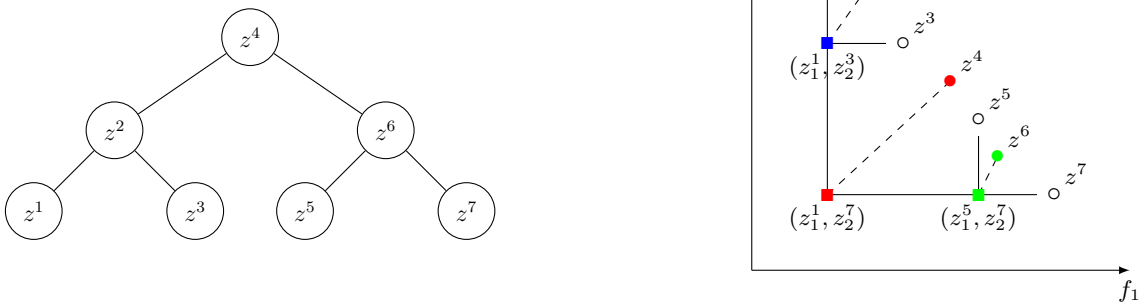


(c) The subtree rooted at $(110, 38)$ is unbalanced $\Rightarrow$ a simple left rotation is performed.



(d) The final balanced tree.

Figure 3.8 – Example of insertion of a candidate $\overline{x} \in X$ s.t. $f(\overline{x}) = (110, 38)$ into an AVL-Archive.



(a) The archive (stored points are indicated in nodes).

(b) View in the objective space of the archive. Points are indicated with circles and local ideals are indicated with squares. An internal point and its associated local ideal are linked by a dashed line.

Figure 3.9 – Example of a SAAVLA

Note that these updates are done in $O(\log_2(s))$, where s is the size of the archive.

Figure 3.9 illustrates an example of a SAAVLA.

A SAAVLA maintains a so-called *cache node*, which is:

- either the node which has rejected the last candidate solution;
- or the parent of the new node containing the last candidate.

When a new candidate solution is presented to the archive, instead of performing a dichotomic search from the root, it starts from the cache node. The general idea is that if a locality assumption holds, then two consecutive candidates will probably be either accepted or rejected in the same subtree. Starting from the cache node prioritizes the previously explored subtree.

However, an issue occurs with this mechanism. Let η be the cache node and $\overline{x} \in X$ be the current candidate solution. Thus the search starts from the cache node, but it may happen that $\overline{x}$ is weakly dominated or even dominates a node *outside* the subtree rooted at the cache node.

Next we introduce important notions to address this issue.

Definition 3.1.6. *A relevant subtree* $\Theta(\eta)$ w.r.t. a solution $\overline{x} \in X$ is a subtree such that:

1. $\overline{x}$ can not dominate any node outside the subtree;
2. and if a node outside the subtree weakly dominates $\overline{x}$, then a node inside the subtree also weakly dominates $\overline{x}$ for sure.

A relevant subtree corresponds to a subtree from which a dichotomic search can be effectively started, because the acceptance/rejection process is not influenced by nodes outside the relevant subtree. Let us now present some propositions supervising the notion of relevant subtree.

Proposition 3.1.7. Let η, η' be two nodes. $\eta' \in \Theta(\eta)$ iff. $z^*(\eta) \leqq f(x(\eta'))$.

Proof. Obvious from the combination of Property 3.1.1 and Proposition 3.1.1. □

Proposition 3.1.8. Let $\overline{x} \in X$ be a candidate solution external to the archive and η be a node. If $z^*(\eta) \leqq f(\overline{x})$, then $\forall \eta' \notin \Theta(\eta)$, $f(\overline{x}) \not\leq f(x(\eta'))$.

Proof. Let us assume that the proposition is incorrect. So there is a node $\eta' \notin \Theta(\eta)$ such that $f(\overline{x}) \leq f(x(\eta'))$, thus $z^*(\eta) \leq f(x(\eta')$ since $z^*(\eta) \leqq f(\overline{x})$. This is contradictory with the fact that $\eta' \notin \Theta(\eta)$ given Proposition 3.1.7. □

Proposition 3.1.9. Let $\overline{x} \in X$ be a candidate solution external to the archive and η be a node. If $z^*(\eta) \leqq f(\overline{x})$ and $\exists \eta' \notin \Theta(\eta)$ s.t. $f(x(\eta')) \leqq f(\overline{x})$, then $\exists \eta'' \in \Theta(\eta)$ s.t. $f(x(\eta'')) \leqq f(\overline{x})$.

Proof. Let $z^L \in Y$ be the point stored in the left-most point of $\Theta(\eta)$, i.e. st. $z_1^*(\eta) = z_1^L$, and $z^R \in Y$ the point stored in the right-most node of $\Theta(\eta)$, i.e. st. $z_2^*(\eta) = z_2^R$.

(i) If $f_2(x(\eta')) < z_2^R$, thus $z_1^R < f_1(x(\eta'))$ since $f(x(\eta')) \parallel z^R$. Moreover, $z_1^R < f_1(\overline{x})$ since $f(x(\eta')) \leqq f(\overline{x})$. Thus $z^R \leqq f(\overline{x})$ since $z_2^R = z_2^*(\eta) \leqslant f_2(\overline{x})$.

(ii) Otherwise, $f_1(x(\eta')) < z_1^L$ since $\eta' \notin \Theta(\eta)$. The remaining of the proof is similar to (i) and we finally obtain that $z^L \leqq f(\overline{x})$. □

Instead of starting immediately a dichotomic search from the cache node, we travel up the tree until reaching an ascendant η of the cache node (including itself) such that $z^*(\eta) \leqq f(\overline{x})$. In this case, given Propositions 3.1.8 and 3.1.9, we are located in a so-called *relevant subtree* w.r.t. $\overline{x}$, thus a dichotomic search is started from η.

Algorithme 18 : `SAAVLA::Add`

Input : candidate solution $\overline{x}$
Output : boolean

```
1 η ← CacheNode()
2 while p(η) ≠ ∅ do
3     if z*(η) ≦ f(x̄) then
4         return Add(η, x̄)
5     η ← p(η)
6 return Add(η, x̄)
```

Algorithm 18 describes the revised `Add` procedure of SAAVLA. First we search for the relevant subtree w.r.t. the candidate solution $\overline{x}$. Once this node is found, we proceed to the vanilla version of the `Add` procedure.

3.2 NDR*-Archive

A R*-tree [Guttman, 1984, Beckmann et al., 1990] is a *spacial indexing* tree data structure used for indexing spatial objects (like hyper-rectangles, hyper-spheres or more complex objects) in a multi-dimensional space. This data structure is extensively based on the notion of MBB. First, by ease of simplicity and to avoid costly geometrical computations, each object stored in a R*-tree is instead represented by its *own MBB*, i.e. the MBB completely covering its particular shape. Second, a R*-tree can be seen as a hierarchy of MBBs, such that:

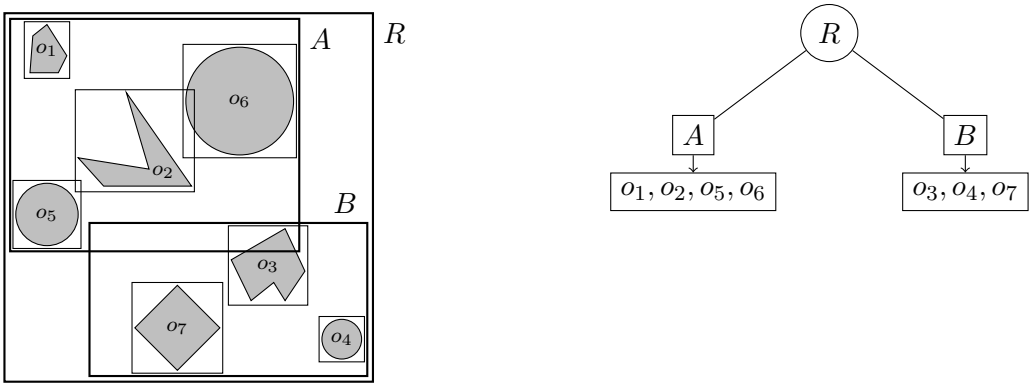


Figure 3.10 – Example of R*-tree in two-dimensional space containing seven spatial objects distributed into two leaves A and B, themselves contained in the root node R.

- A leaf node contains a list of objects. All the objects stored in the R*-tree are partitioned among the leaves, such that nearby objects tend to be in the same leaf node. To a leaf is attached a MBB including all the objects stored by this leaf.

- An internal node contains a list of child nodes. In the same way as a leaf, to an internal node is attached a MBB including all the MBB of its children.

Figure 3.10 shows a R*-tree. As a spatial indexing data structure, a R*-tree aims at providing fast answers to online queries such as searching, inserting or removing a number of indexed objects.

Since all the objects contained in a node lie within the attached MBB, a query that does not intersect the MBB cannot intersect any of the contained objects. This way, when a query is performed, a Depth First Search (DFS) is executed from the root of the tree and explores only the children whose MBB is concerned by the query.

A R*-tree has two main additional properties. First, it is a *balanced k-ary tree*, meaning that at any time, all leaves are at the same depth. Second, the nodes have a fixed minimum and a fixed maximum children/objects list size. The data structure is managed such that three main criteria are minimized: the total hypervolume, the margin (corresponding to the sum of the length of the edges) of the MBBs, and the overlap between the MBBs.

When an internal node (resp. leaf) is *overfilled* after the insertion of a new object, two strategies are successively considered. The first one, called *extraction strategy*, is to extract from the children (resp. list) of the overfilled node a number of nodes (resp. objects) and reinsert them like regular new objects. The idea behind this strategy is that the extracted objects might be re-inserted in a more suitable node of the tree. The second one, called *splitting strategy*, aims at splitting the overfilled node into two new nodes. Globally, the splitting strategy is used on a node when the extraction strategy has failed.

Once a node is *underfilled*, it is removed from the tree and its children/objects are reinserted.

R*-trees do not guarantee good worst-case performance, but generally perform well with real-world data [Hwang et al., 2003].

Unfortunately, we found that adapting directly the R*-tree structure for archiving task provides extremely poor results compared to the state-of-the-art archive ND-Tree. This first failure led us to the idea of combining these two structures into a new one attempting to retain the strengths of both structures and trying to lessen their weaknesses. We call this new archive NDR*-Archive.

3.2.1 Design of NDR*-Archive

The NDR*-Archive combines features of ND-tree and R*-tree. Each one of these two data structures has its own strengths and weaknesses:

- ND-tree has been specially designed for archiving, as outlined in Section 2.4.4. On the other hand, it is not a balanced tree, allowing a low structure maintenance cost.
- R*-tree has not been designed for archiving. On the other hand, it is a balanced tree at the expense of a high structure maintenance cost. Another drawback of a R*-tree is that it is complex to implement, in particular because it is itself an upgrade of R-tree [Guttman, 1984] to which several new features have been added.

NDR*-Archive is, globally, a simplified and modified, archiving-adapted R*-tree using the theoretical properties of ND-tree (reviewed in Section 2.4.4) to manage efficiently a set of incomparable points, but including also new features.

NDR*-Archive works as follows. Each node η maintains:

- an approximate local ideal $ideal(\eta)$ and nadir $nadir(\eta)$ points, defining a bounding box including all the points stored in the subtree;
- a list of its children $children(\eta)$ if it is internal, and a list of solutions $solutions(\eta)$ if it is a leaf.

Let an *object* be either a node or a solution. Like R*-tree, the nodes have a fixed minimum and a fixed maximum children/solutions list size. The process of NDR*-Archive is driven by two main procedures: the *Add solution procedure* and the *Restructure tree procedure*.

3.2.1.1 The Add solution procedure

Algorithme 19 : `Add`

Input : candidate solution x
Output : boolean

```
1  dominated ←CheckDominance(root, x)
2  if dominated then
3      return false
4  N_in ← ∅; X_in ← {x}
5  Prune(root, x, N_in, X_in)
6  foreach η_in ∈ N_in do
7      ReInsert(root, η_in, prevDepth(η_in) − 1, 0)
8  foreach x_in ∈ X_in do
9      ReInsert(root, x_in,leafDepth() − 1, 0)
10 return true
```

The *Add solution procedure* is the core of NDR*-Archive. It is described with a diagram in Figure 3.11 and detailed in pseudo-code by Algorithm 19: `Add`. This procedure is activated when a new candidate solution x is presented to the archive. It works as follows:

(1) First, one checks if x is dominated by exploring the archive in DFS starting from the root (Algorithm 20: `CheckDominance`). For each node met, if x is not weakly dominated by the local ideal, then x is not

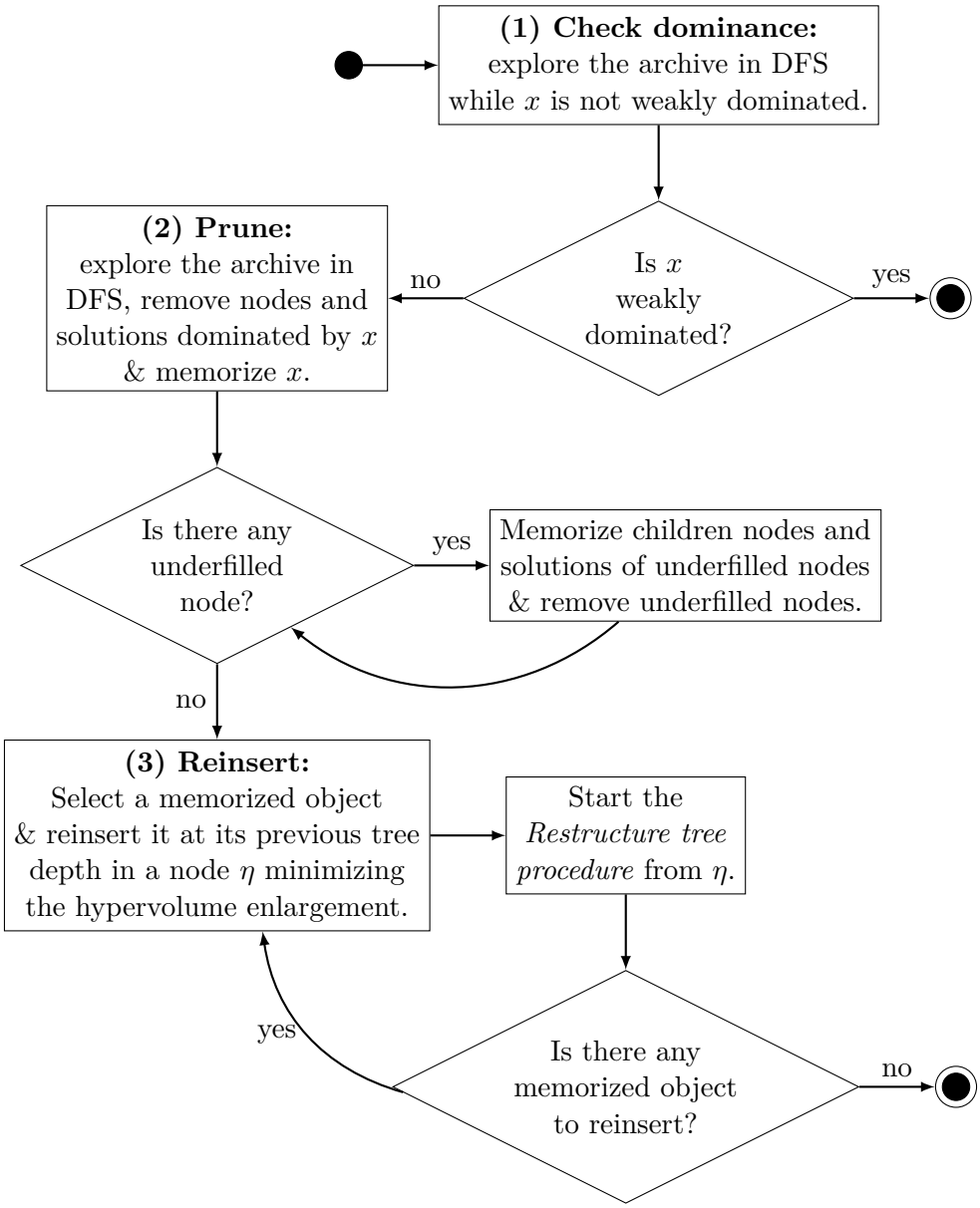


Figure 3.11 – The *Add solution procedure* of NDR*-Archive with a candidate solution x.

Algorithme 20 : `CheckDominance`

Input : calling node η, candidate solution x
Output : boolean

```
1  if ideal(η) ⋦ x then
2      return false
3  if nadir(η) ≦ x then
4      return true
5  if η is internal then
6      foreach cη ∈ children(η) do
7          if CheckDominance(cη, x) then
8              return true
9  else // η is a leaf
10     foreach x' ∈ solutions(η) do
11         if f(x') ≦ f(x) then
12             return true
13 return false
```

dominated by any point in the subtree; otherwise, if the local nadir does not weakly dominate x, then the search continues down in the subtree while x is not proved to be dominated. Note that this procedure just checks if x is dominated, so it makes only one-sided comparisons between x and the objects of the tree.

(2) If x is accepted, then no solution in the archive weakly dominates x. In this case, the tree is pruned by removing dominated nodes and solutions through the exploration of the archive in DFS starting from the root (Algorithm 21: `Prune`).

More precisely, for each node met, if x does not weakly dominates the local nadir, then x does not dominate any point in the subtree; otherwise, if the local ideal is weakly dominated by x then all the points in the subtree are dominated by x, so the node is removed. Otherwise, the search continues down in the subtree and any dominated object or empty node is removed. If a node is underfilled after the deletion of some of its children/solutions, then its children/solutions are memorized in a list (called the *memorized objects* for the rest of this section), then the node is removed.

When a child node $c\eta$ is removed, the `UpdateIdealNadir`$(\eta, \{c\eta\}, -)$ procedure is called from its parent η.

The procedure `UpdateIdealNadir`$(\eta_{src}, N_{modif}, option)$ updates the approximate local ideal and local nadir of the node η_{src} considering the deletion of $(option = -)$, the inclusion of $(option = +)$ or the reset with $(option = *)$ the set of nodes N_{modif}. The procedure also updates accordingly the local ideal and local nadir of the ascendants of η_{src}. In order to reduce the maintenance cost, this procedure is not called when a solution is removed.

(3) When all dominated objects have been removed, then the candidate solution and the memorized objects are (re)inserted in the tree (Algorithm 22:`ReInsert`).

Let the *hypervolume of a node* be the hypervolume of the bounding box defined by its approximate local ideal and local nadir points.

To reinsert each memorized object, the tree is explored by following a simple top-down path starting at the root; for each node met, the child for which the insertion of the object minimizes the hypervolume enlargement is selected. Then, the procedure is recursively called from this child. If the object is a solution, then it is (re)inserted in a leaf; otherwise, if the object is a node, then it is reinserted at the same tree depth

Algorithme 21 : `Prune`

Input : calling node η, candidate solution x, list of nodes to reinsert N_{in}, list of solutions to reinsert X_{in}
Output : boolean

```
1  if x ⋦ nadir(η) then
2      return false
3  if x ≦ ideal(η) then
4      return true
5  if η is internal then
6      foreach cη ∈ children(η) do
7          if Prune(cη, x, N_in, X_in) then
8              children(η) ← children(η) \ {cη}
9              UpdateIdealNadir(η, {cη}, -)
10             delete cη
11     if η is underfilled then
12         N_in ← N_in + children(η)
13         children(η) ← ∅
14 else // η is a leaf
15     foreach x' ∈ solutions(η) do
16         if f(x) ≤ f(x') then
17             solutions(η) ← solutions(η) \ {x'}
18             delete x'
19     if η is underfilled then
20         X_in ← X_in + solutions(η)
21         solutions(η) ← ∅
22 return (η is empty)
```

Algorithme 22 : `ReInsert`

Input : calling node η, node/solution to reinsert ω, reinsertion depth $r\delta$, current depth δ
Output : $\emptyset$

```
1  if δ < rδ then
2      cη* ← arg min{Hypervolume(cη + ω) − Hypervolume(cη) : cη ∈ children(η)}
3      ReInsert(cη*, ω, δ + 1)
4  else // δ = rδ
5      if η is internal then
6          children(η) ← children(η) + ω
7      else // η is a leaf ⇒ ω is a solution
8          solutions(η) ← solutions(η) + ω
9      UpdateIdealNadir(η, {ω}, +)
10     RestructureTree(η)
```

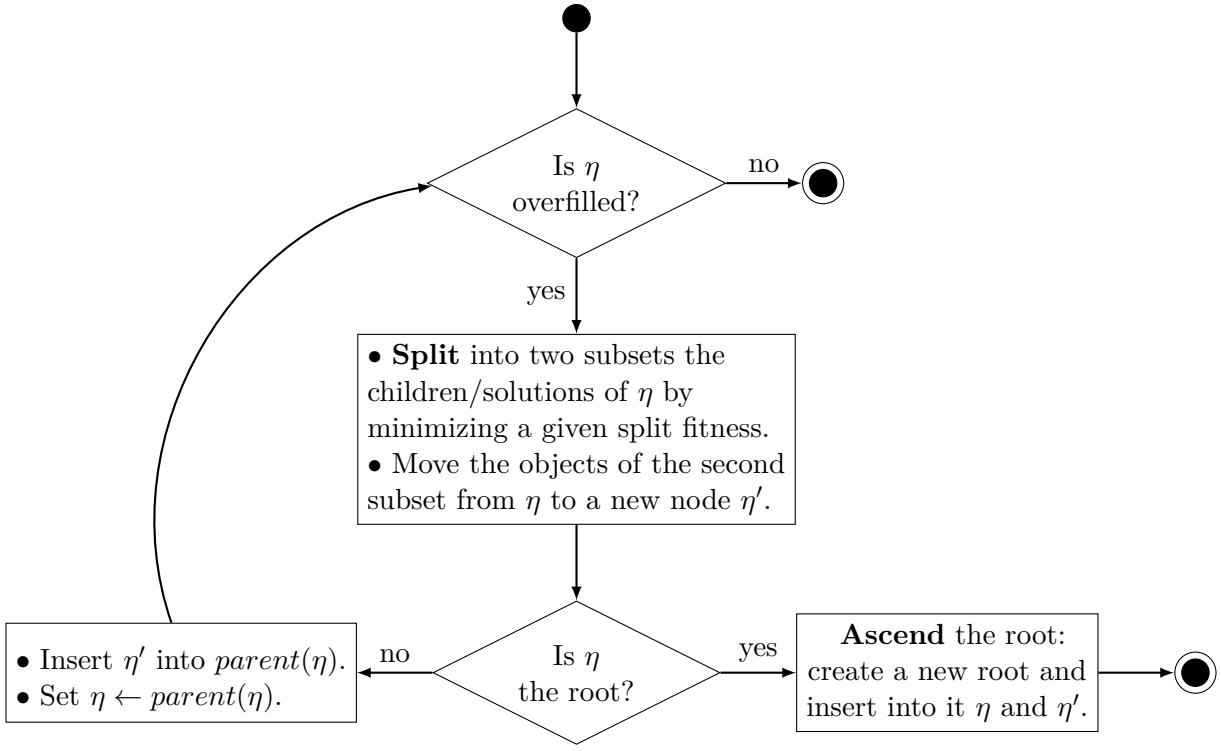


Figure 3.12 – The *Restructure tree procedure* of NDR*-Archive called from a node η.

than before its memorization (in order to preserve the structure of the tree). At the end, the approximate local ideal and local nadir of the parent node are updated. Note that the minimization of the hypervolume at each level of the tree is crucial in the sense that it corresponds to a minimization of the dead space between the MBBs of the nodes; and reducing this dead space helps reducing the height of the tree. Finally, the *Restructure tree procedure* is activated to re-structure the tree from the parent node.

3.2.1.2 Restructure tree procedure

Algorithme 23 : `RestructureTree`

Input : calling node η
Output : $\emptyset$

```
1  if η is overfilled then
2      η' ←Split(η)
3      if η ≠ root then
4          pη ← parent(η)
5          children(pη) ← children(pη) + η'
6          RestructureTree(pη)
7      else // η = root ⇒ ascend root
8          root ←new node()
9          children(root) ← η + η'
10         UpdateIdealNadir(root, η + η', *)
```

The *Restructure tree procedure* is described with a diagram in Figure 3.12 and detailed in pseudo-code by Algorithm 23: `RestructureTree`. Let η be the node from which the procedure is called. If η is overfilled,

then it is split. The split procedure is a key component of NDR*-Archive. It aims at partitioning into two subsets the children/solutions of a node.

R*-tree split manages MBBs of spatial objects while ND-tree split manages points. On the other hand, NDR*-Archive faces both cases: it has to split internal nodes, such that children are nodes represented by their bounding boxes; and it also has to split leaves, containing points. Two different splits are considered for each case. The main difference between the internal node splits and the leaf splits is the *split fitness* to optimize.

Algorithme 24 : `Split (I1)`

Input : calling internal node η
Output : new node η'

```
1  max, min ←maximum, minimum storage capacity of children(η)
2  SplitFitness(·,··) ← lex {Overlap(·,··), Hypervolume(·) + Hypervolume(··)}
   // Compute the best partition:
3  (N1*, N2*) ← ∅
4  for j = 1,...,p do
5      foreach optimum ∈ {ideal(η), nadir(η)} do
6          (cη1,...,cη_{max+1}) ←Sort children(η) in ascending order by the j-th objective value of their optimum
7          for k = min,...,max − min + 1 do
8              N1 ← (cη1,...,cηk)
9              N2 ← (cη_{k+1},...,cη_{max+1})
10             (N1*, N2*) ← arg min{SplitFitness(N1*, N2*); SplitFitness(N1, N2)}
   // Update η:
11 η' ←new node()
12 children(η) ← N1*;   UpdateIdealNadir(η, N1*, *)
13 children(η') ← N2*;   UpdateIdealNadir(η', N2*, *)
14 return η'
```

The **first internal node split** (Algorithm 24: `Split (I1)`) we proposed, called ***split I1***, is a simplified version of the R*-tree split. Globally, given an internal node to split, the idea is to select a partition of its children depending on a splitting objective which minimizes the overlap between the two resulting subsets.

More precisely, let η be the internal node to split. The computation of the best partition of the children of η iterates both on: (i) the objectives ($j = 1,...,p$) considered alternately for sorting the children of η; (ii) the local optima of any child node (*ideal*, *nadir*) used as sorting key. The *split axis* is the current objective selected for sorting.

Let $j \in \{1,...,p\}$ be the index of the current split axis, and $optimum \in \{ideal, nadir\}$ the current optimum. First, the children are sorted by their *optimum*'s j-th objective value. Any partition such that both subsets contain only consecutive children given the current sorting, and is neither underfilled nor overfilled is considered. We look for the partition minimizing the overlap between the bounding boxes of its two subsets; ties are resolved by choosing the partition such that the hypervolume sum of the bounding boxes of its subsets is minimum.

Note that minimizing the overlap is of prime importance because it has a deep impact on the number of paths to be traversed when a solution is presented to the archive.

The **first leaf node split** we designed, called ***split L1***, works similarly than the split I1 by choosing the axis and the corresponding partition minimizing a given fitness; except for two differences: the objective values of the solutions are considered for sorting, and the fitness used to evaluate a partition is the distance between the center of its two subsets (to maximize), instead of the overlap. We still search for minimizing the hypervolume in case of tie. We found that using either the Manhattan or the Euclidean distance provides similar results. Globally, the idea behind this split is to maximize the gap between the two resulting subsets.

Algorithme 25 : `Split (L2)`

Input : calling leaf node η
Output : new node η'

$max \leftarrow$maximum storage capacity of $solutions(\eta)$
`// Compute the best partition:`
$(x_1, x_2) \leftarrow \arg\max\{\texttt{Distance}(x, x') : x, x' \in solutions(\eta)\}$
$solutions(\eta) \leftarrow solutions(\eta) \setminus \{x_1, x_2\}$
$\overline{X}_1 \leftarrow \{x_1\}$; $\overline{X}_2 \leftarrow \{x_2\}$
while $solutions(\eta) \neq \emptyset$ **and** $\left|\overline{X}_1\right| < max$ **and** $\left|\overline{X}_2\right| < max$ **do**
  $x \leftarrow \arg\min\{\min\{\texttt{Distance}(x', \texttt{center}(\overline{X}_1)); \texttt{Distance}(x', \texttt{center}(\overline{X}_2))\} : x' \in solutions(\eta)\}$
  $solutions(\eta) \leftarrow solutions(\eta) \setminus \{x\}$
  if $\texttt{Distance}(x, \texttt{center}(\overline{X}_1) < \texttt{Distance}(x, \texttt{center}(\overline{X}_2)\}$ **then**
    $\overline{X}_1 \leftarrow \overline{X}_1 + x$
  else
    $\overline{X}_2 \leftarrow \overline{X}_2 + x$
if $\left|\overline{X}_1\right| < max$ **then**
  $\overline{X}_1 \leftarrow \overline{X}_1 + solutions(\eta)$
else
  $\overline{X}_2 \leftarrow \overline{X}_2 + solutions(\eta)$
`// Update` η`:`
$\eta' \leftarrow$**new node**()
$solutions(\eta) \leftarrow \overline{X}_1$; `UpdateIdealNadir(`$\eta$`,` $\overline{X}_1$`, *)`
$solutions(\eta') \leftarrow \overline{X}_2$; `UpdateIdealNadir(`$\eta'$`,` $\overline{X}_2$`, *)`
return η'

The **second internal node split**, called ***split I2***, is mainly based from the split used by R-trees. Contrary to I1, it considers all the objectives at the same time and try to minimize the total hypervolume of the new partition.

More precisely, it first selects the pair of child nodes such that the dead space between their MBBs is maximal, each one is extracted from the split node and inserted in two different subsets. Then, while the split node has a child and the two subsets are not overfilled, the child node minimizing the hypervolume enlargement through its insertion in any of the two subsets is selected. This node is extracted from the list of children of the split node and inserted in the best subset. Finally, if a subset is full, the remaining children of the split node are inserted in the other subset.

The **second leaf node split**, called ***split L2***, works in a similar way as *split I2* except that the fitness used is the (Euclidean) distance between a point and the center of a subset (to minimize). Indeed, using the

hypervolume enlargement as fitness did not provide good results for this leaf split. Algorithm 25 describes *split L2*. Globally, this split has some similarities with the split used in ND-Tree.

Independently to the split used, once the best partition has been found, the nodes of the second group are extracted (from the split node η) to a new node η'.

Finally, the same *restructure tree procedure* is restarted from the parent of η. Note that if the overfilled node η is the root of the archive (cf. lines 7-10 of Algorithm 23), the root is *ascended*, meaning that a new root is created and takes as children the old root and the new node created by the split. This mechanism increases by one the height of the tree.

This last step concludes the description of the NDR*-Archive. Figure 3.13 illustrates an example of insertion of a non-dominated candidate solution in a NDR*-Archive.

3.2.2 Self-adjusting version

Algorithme 26 : Add: modifications for self-adjustment.

```
// Replace the instruction line 1 in Algorithm 19 by the following
// instructions to implement a self-adjusted NDR*-Archive:
1 η ← cacheNode(root)
2 repeat
3 |   dominated ← CheckDominance(η, x)
4 |   η ← parent(η)
5 until dominated or η =none;
```

The proposed upgrade for self-adjustment of NDR*-Archive is general and goes beyond the archiving case. Indeed, it enables any R-like tree to use locality assumption, even for regular indexing tasks. Two main features characterize this upgrade:

1. We memorize the so-called *cache node*, from which the last candidate solution has been either inserted or rejected. Then, when a new candidate solution is presented to the archive, we start the `CheckDominance` procedure from the *cache node* instead of the root of the archive. When the subtree has been explored and the candidate solution not rejected, then the `CheckDominance` procedure is called from the parent of the cache node. This process continues until the candidate is rejected or the root is reached. Note that we forbid a subtree to be explored twice. Algorithm 26 indicates the instructions replacing the first instruction of the `Add` procedure (Algorithm 19):

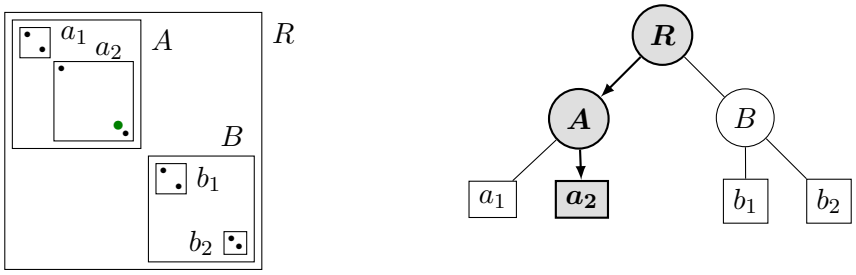
   $$dominated \leftarrow \texttt{CheckDominance}(root,x)$$

   to handle self-adjustment, where *root* is the root of the archive and x is the current candidate solution.

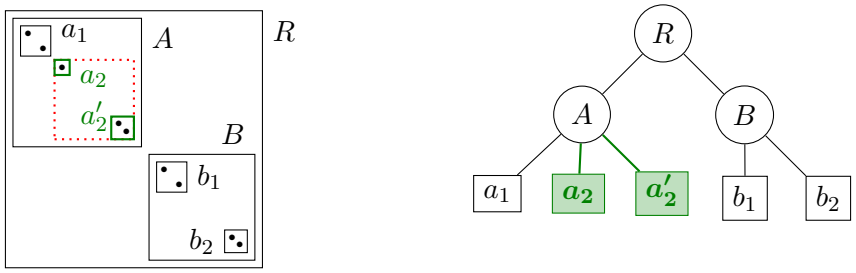
2. When the candidate is either inserted or rejected, all the nodes and the single solution (in a potential leaf) on the insertion/rejection path are then positioned in first position.

We call this archive *SANDRA*, standing for *Self-Adjusting NDR*-Archive*. As shown by our experiments reported in the next section, these simple modifications lead to substantial computational time improvements.

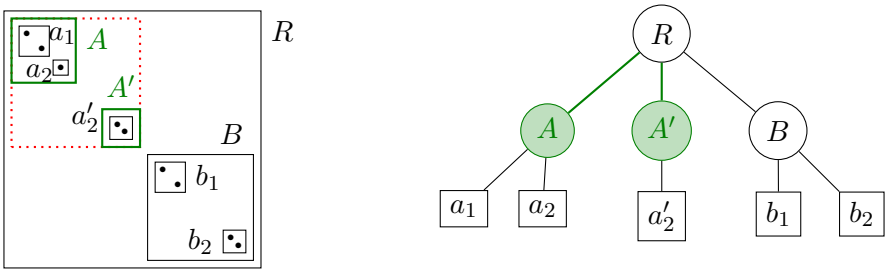
Figure 3.14 describes two use case examples of *SANDRA*.



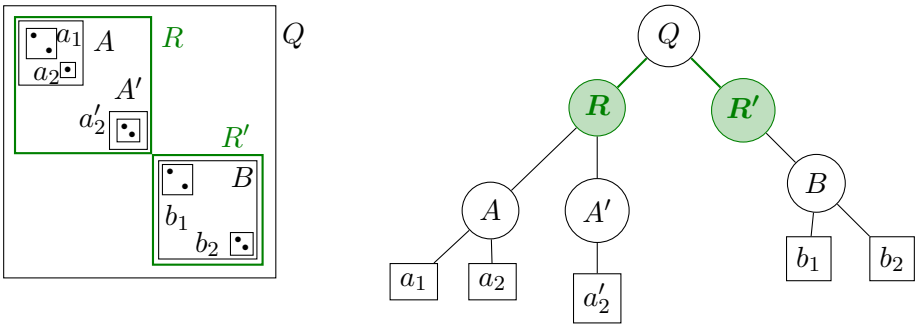
(a) Step 1: The new solution is inserted into the leaf a_2 by calling `ReInsert` from the root node R.



(b) Step 2: After the insertion, the leaf a_2 is overfilled, so it is split by calling `Split` from a_2. A new node a'_2 containing two solutions is created and inserted into A.

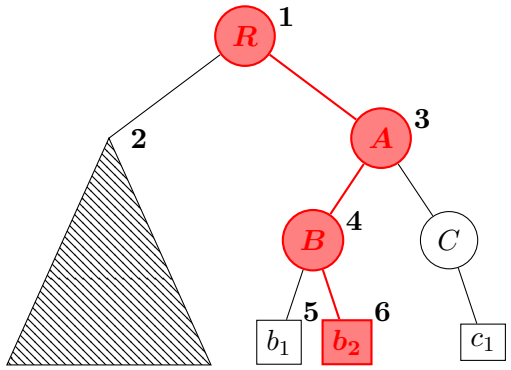


(c) Step 3: After the split of a_2, A is overfilled. So A is split and a new node A' containing a'_2 is created and inserted into R.

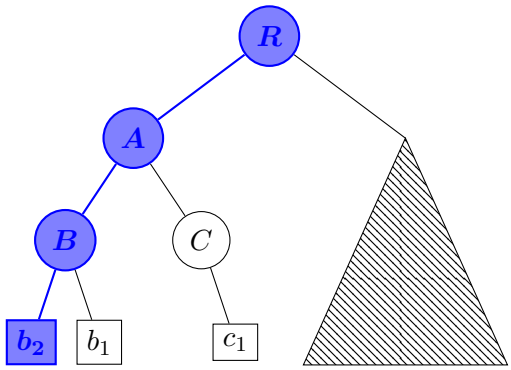


(d) Step 4: After the split of A, R is overfilled. So R is ascended: it is split and a new node R' containing B is created. A new root Q is created and both R and R' are inserted into Q.

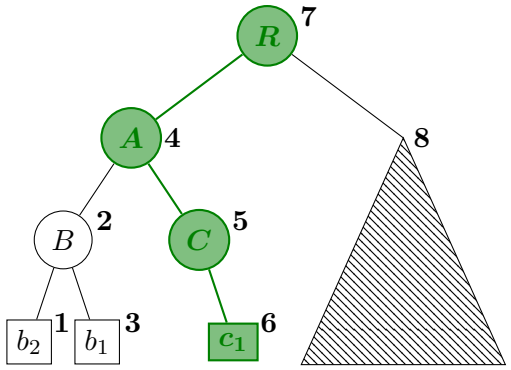
Figure 3.13 – Example of insertion of a new candidate solution x in a NDR*-Archive in the bi-objective case. We assume that x has already been checked for non dominance; and that internal and leaf nodes can contain at most two elements and at least one element.



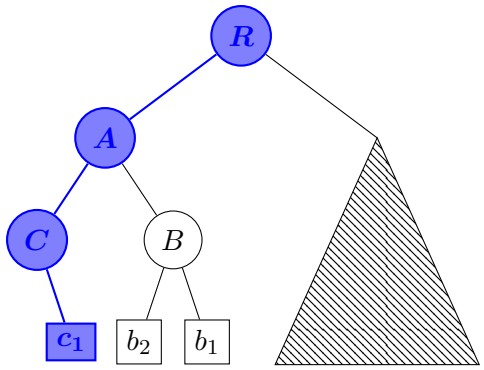
(a) The tree and the rejection path related to a candidate **dominated** by node b_2.



(b) The resulting ajusted tree in order to have the path $R - A - B - b_2$ in first position.



(c) The tree and the insertion path related to an **accepted** candidate.



(d) The resulting ajusted tree in order to have the path $R - A - C - c_1$ in first position.

Figure 3.14 – Illustration of the self-adjusting feature of SANDRA with an example of a rejection of a candidate (first row) *followed by* an acceptation of another candidate (second row). The numbers close to the nodes indicate the exploration order of the nodes.

3.3 Experiments

AVL-Archive, NDR*-Archive and their self-adjusting versions (SAAVLA, SANDRA) are experimentally compared to the current best known archives: ND-Tree [Jaszkiewicz and Lust, 2016] (for any number of objectives p) and sorted list (for $p = 2$ only). In particular:

1. AVL-Archive, NDR*-Archive, ND-Tree and sorted list are compared on a benchmark of several artificial sets simulating the generation of points during the run of a meta-heuristic;

2. AVL-Archive, SAAVLA, NDR*-Archive, SANDRA, ND-Tree and sorted list are applied within PLS on MOTSP.

All experiments presented were performed on a 3.4 GHz computer with 16Gb of volatile memory (RAM) on a Linux OS. All algorithms are written in C/C++. We use the C/C++ ND-Tree implementation of Jaszkiewicz and Lust and a slightly improved version of the sorted list they implemented. The linear list has been implemented with the Move-to-Front heuristic [Bentley et al., 1993].

Splits	C_{max}^{child}	$C_{max}^{solution}$	R_{min}
I1+L1	$p+6$	20	45%
I1+L2	$p+1$		
I2+L1	$p+6$		30%
I2+L2	$p+2$		25%

Table 3.1 – Good parameter settings for NDR*-Archive with different split combinations.

3.3.1 Artificial sets

Given p objectives to minimize, an artificial set is composed of N_{nd} non-dominated points, and $N_{dom} = \lfloor \phi_{dom} \times N_{nd} \rfloor$ dominated points, where $\phi_{dom} \geq 0$. Let us now describe how an artificial set of points is created. Any point $y = (y_1, ..., y_p) \in \mathbb{N}^p$ is generated uniformly at random in $\{0, ..., R\}^p$ inside a hyperball of center $(R, ..., R)$ and radius R, such that $\sum_{j=1}^{p} (R - y_j)^2 \leq R^2$. The idea is to obtain points near the corresponding hypersphere and not near the point $(0, ..., 0)$. In order to control the dispersion of the points generated, the constraint $\sum_{j=1}^{p} (R - y_j)^2 \geq (1 - \varepsilon) \times R^2$ is added, where $\varepsilon \in [0, 1]$. With a small ε, we obtain a dense set of points close to the hull of the hypersphere; while a large ε generates a scattered set of points.

This way of generating a set of points proposed in [Jaszkiewicz and Lust, 2016] simulates well the behavior of a multi-objective heuristic. Unfortunately, with this methodology, we can not control a fundamental parameter: the proportion of dominated points ϕ_{dom}, computed as the number of dominated points divided by the number of non-dominated points. As we will see later, the archiving time greatly depends on this parameter. We proceed in two steps to generate in the hyperball, a set of points with N_{nd} non-dominated points and N_{dom} dominated points.

During the first step, we successively generate points uniformly at random in the hyperball and present each one to an archive, until N_{nd} incomparable points are finally obtained.

The second step produces the dominated points. To do so, we successively generate points uniformly at random in the hyperball and each one is accepted only if it is dominated by a point in the archive. We stop when N_{dom} dominated points have been produced. Contrary to the first step, this second step is computationally expensive in the sense that it is relatively similar to the approximation with Monte Carlo sampling of the hypervolume indicator value [Zitzler, 1999] of the archive obtained in the first step. Thus, we speed-up the second step using a kd-tree [Bentley, 1975] we have modified to be dynamic and able to handle uniform sampling. The resulting list of solutions can then be presented to an archive.

To summarize, given that the hyperball radius is set to $R = 1000000$, an artificial set is defined by: the number of objectives p, the parameter ε controlling the dispersion of the points, the number of non-dominated points N_{nd}, and a proportion of dominated points ϕ_{dom} controlling the number of dominated points (in function of the number of non-dominated points).

3.3.2 Split and parameter setting of NDR*-Archive

NDR*-Archive has three parameters:

- the *maximum* storage capacities of the list of children of internal nodes C_{max}^{child} and of the list of solutions of leaf nodes $C_{max}^{solution}$

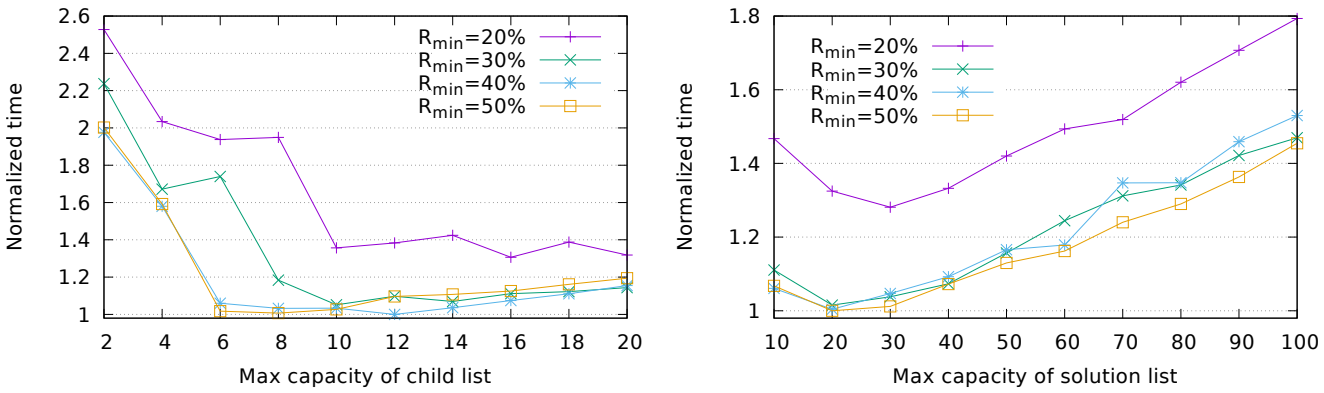


Figure 3.15 – Influence of the parameters C_{max}^{child} (left), $C_{max}^{solution}$ (right) and R_{min} on the computational time of NDR*-Archive with $I1 + L1$ splits ($p = 4$, $N_{dom} = 30000$, $\phi_{dom} = 10$, $\varepsilon = 15\%$).

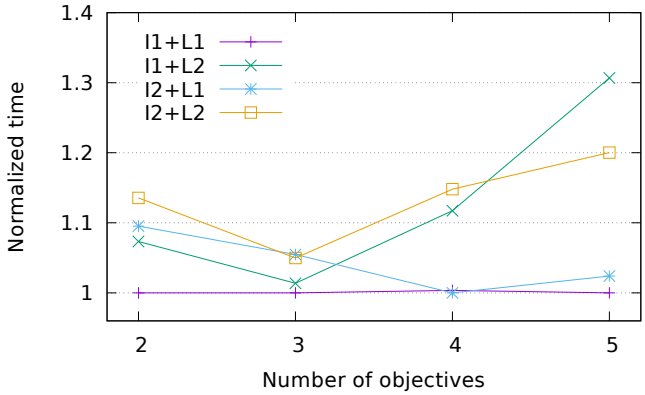
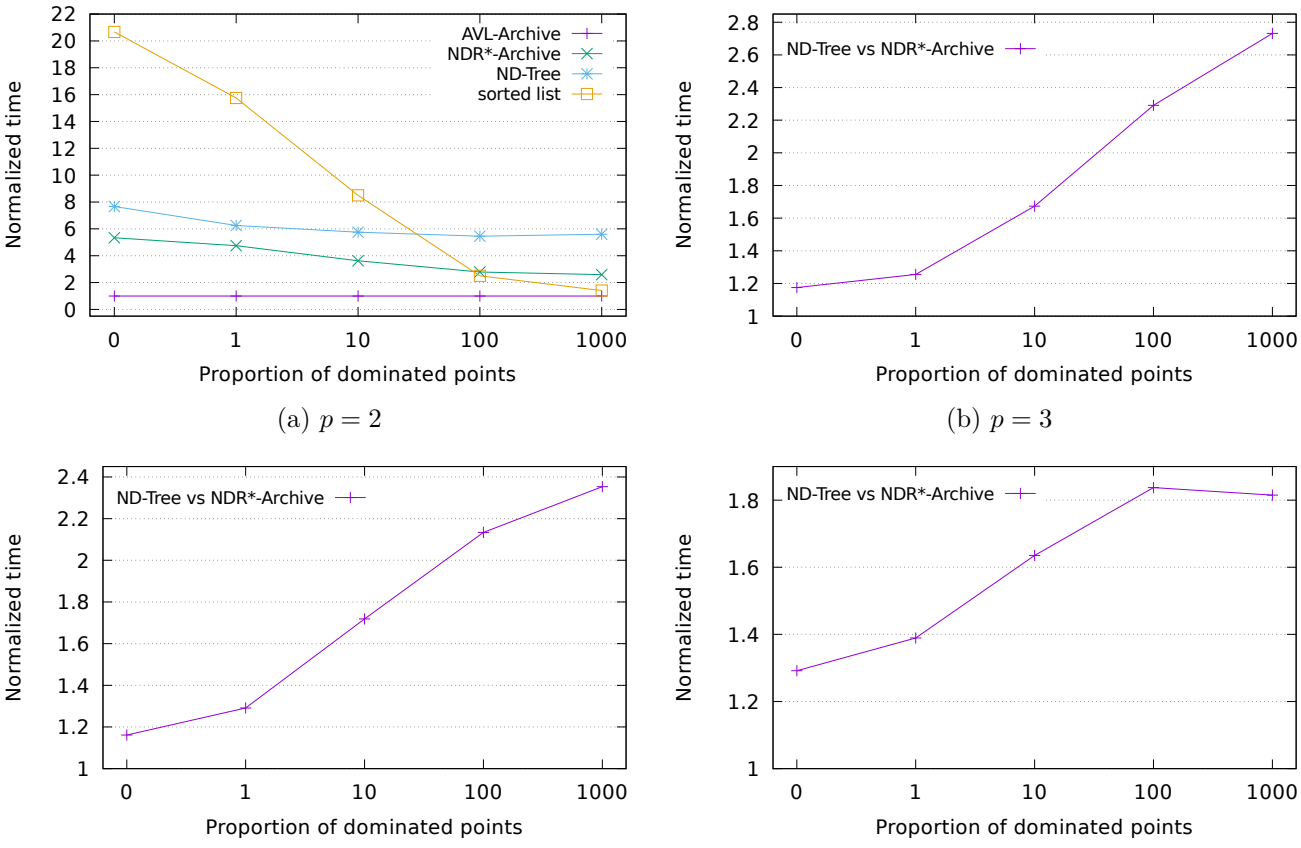


Figure 3.16 – Evolution of the running time of different combinations of splits in function of the number of objectives.

- the rate $R_{min} \in [0, 1]$ controlling the *minimum* storage capacity of both child and solution lists such that $C_{min}^{child} = \left\lfloor R_{min} \times C_{max}^{child} \right\rfloor$ and $C_{min}^{solution} = \left\lfloor R_{min} \times C_{max}^{solution} \right\rfloor$, where C_{min}^{child} (resp. $C_{min}^{solution}$) is the minimum storage capacity of child (resp. solution) list.

Four different splits have been proposed for NDR*-Archive. A good parameter setting has been found for each one of the four possible combinations of splits I1+L1, I1+L2, I2+L1 and I2+L2. The chosen parameter settings are indicated in Table 3.1 and Figure 3.15 shows the influence of different parameter settings for NDR*-Archive with the split combination I1+L1. Globally, the split I1 needs a high R_{min} while I2 needs a low R_{min}; and L1 works better when C_{max}^{child} is large, whereas L2 seems to favor a low C_{max}^{child}. For all combinations and number of objectives, a $C_{max}^{solution}$ set to 20 obtains good results.

For each number of objectives $p = 2, 3, 4, 5$, we tested the different combinations of splits (with their previously proposed settings) by comparing them on a benchmark of artificial sets with intermediate parameter values: a spread of points $\varepsilon = 15\%$, $N_{nd} = 30000$ non-dominated points and a proportion of dominated points set to $\phi_{dom} = 10$. Results are indicated in Figure 3.16. The split L1 seems to be particularly efficient and the best combination of splits appears to be I1+L1.



(a) $p = 2$ (b) $p = 3$

(c) $p = 4$ (d) $p = 5$

Figure 3.17 – Computational time comparison between the different archives in function of the proportion of dominated points ϕ_{dom} for $p = 2, 3, 4, 5$ ($N_{nd} = 10000$, $\varepsilon = 25\%$).

3.3.3 Experiments on artificial sets

The benchmark of artificial sets we propose for comparing AVL-Archive, NDR*-Archive, ND-Tree and sorted list is composed of 1600 ($= 80 \times 20$) instances of 80 different types where each instance contains between 10 thousands and 50 million points. Each type is defined by the following instance parameters:

- number of objectives $p = 2, 3, 4, 5$
- dispersion of points $\varepsilon = 5\%, 25\%$
- number of non-dominated points $N_{nd} = 10000$, 50000
- number of dominated points $N_{dom} = \lfloor \phi_{dom} \times N_{nd} \rfloor$ where $\phi_{dom} = 0, 1, 10, 100, 1000$

For each type of instance (i.e. for each combination of instance parameters), 20 different instances are randomly generated.

The case $\phi_{dom} = 0$, i.e. when there is no dominated points, generally does not occur in a real-world application, but it allows us to compare how efficient in time are the building mechanisms of the different archives. Note that the upper bound of the interval proposed for ϕ_{dom} is still realistic since, as we will see later when running a meta-heuristic, the number of generated dominated points is generally much larger than the number of non-dominated points.

Results are presented in Tables 3.2, 3.3, 3.4 and 3.5. The different archives (second columns) are compared in terms of number of comparisons (third column) and computational time (fourth column) in function of the proportion of dominated points (first column). The *number of comparisons* (third column) indicates the *average number of dominance comparisons* made between a candidate and the solutions/nodes of an archive so as to accept or reject the candidate. Figure 3.17 shows a computational time comparison between the different archives in function of the proportion of dominated points, for any number of objectives.

For $p = 2$, AVL-Archive outperforms all the other archives and NDR*-Archive performs better than ND-Tree, particularly when the number of dominated points is a hundred times or more larger than the number of non-dominated points; in this case, NDR*-Archive is often at least twice faster than ND-Tree. The sorted list remains slower than AVL-Archive, even when ϕ_{dom} is high. NDR*-Archive and particularly ND-Tree, are outperformed by the sorted list and AVL-Archive in terms of average number of dominance comparisons.

For $p \in \{3, 4\}$, NDR*-Archive performs better than ND-Tree, especially when ϕ_{dom} grows. Globally, when there is as many dominated points than non-dominated points, NDR*-Archive is 27% faster in average than ND-Tree, while NDR*-Archive is at least twice faster than ND-Tree when $\phi_{dom} = 1000$.

For $p = 5$, NDR*-Archive remains better than ND-Tree, and particularly when ϕ_{dom} is high, but a stagnation in terms of performance improvement is observed. Indeed, in average NDR*-Archive is approximately 50% faster than ND-Tree.

This stagnation can be explained by the following analysis. When a candidate is presented to NDR*-Archive, the tree is explored a first time and comparisons are oriented to check if the candidate is dominated. If it is not dominated, the tree is explored a second time and comparisons are made with the sole aim to remove dominated nodes or solutions. We say that NDR*-Archive makes *one-sided* comparisons. Thus, with such a separation of processes, some parts of the tree will be explored twice if the candidate is finally accepted. On the other hand, computationally light comparisons are made if the candidate is dominated. This is one of the new features we introduced. On the contrary, ND-Tree makes complete comparisons for checking if the candidate is either dominated or dominating, during the same exploration of the tree. Yet, for fixed N_{nd}, ϕ_{dom} and ε, the number of accepted candidates tends to increase considerably when p grows (see Figure 3.18(a) as illustration). Consequently, the number of dominance comparisons induced by accepted candidates increases much more for NDR*-Archive than for ND-Tree. Figure 3.18(b) illustrates this trend by indicating the number of comparisons made by NDR*-Archive and ND-Tree for rejected and accepted candidates, in function of the number of objectives p. In order to prevent this issue, a modification of NDR*-Archive would be to explore only once the tree, like ND-Tree. However, this would make more expensive the checking of dominated candidate. We expect the same trend for PLS (Section 3.3.4).

Despite all that, NDR*-Archive remains more efficient than ND-Tree, even when all candidates are non-dominated ($\phi_{dom} = 0$), and particularly for $p \leq 4$. This efficiency is due to the following reasons:

1. Comparisons made by NDR*-Archive are one-sided and thus low-cost, contrary to ND-Tree.

2. According to Section 3.3.2, the splits I1 and L1 used by the NDR*-Archive are more efficient than the other splits generally used in other structures.

3. The dynamic structure of NDR*-Archive allows it to replace previously removed parts of the tree to re-balance it (in other words, the archive can *fill the holes*), which is a mechanism absent in ND-Tree, making NDR*-Archive more compact than ND-Tree. Indeed, an interesting quality criterion measuring the compactness of structures used in the indexing structure research field is the *storage utilization* of nodes computed at the end of the candidate insertions. The storage utilization of a tree is measured by computing the ratio between the current number of child nodes and solutions stored in the tree, and the total number of available slots of child nodes or solutions. Globally, the storage utilization of NDR*-Archive is around 50% while for ND-Tree it is around 30%.

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**AVL-Archive**	**12**		**2**	
	NDR*-Archive	73	+500%	16	+558%
	ND-Tree	101	+735%	23	+858%
	Sorted list	**12**		64	+2,558%
1	**AVL-Archive**	**9**		**4**	
	NDR*-Archive	43	+405%	19	+390%
	ND-Tree	70	+711%	26	+579%
	Sorted list	**9**		64	+1,584%
10	**AVL-Archive**	**6**		**9**	
	NDR*-Archive	19	+242%	35	+278%
	ND-Tree	44	+677%	55	+502%
	Sorted list	**6**		73	+689%
100	**AVL-Archive**	**5**		**67**	
	Sorted list	**5**		146	+117%
	NDR*-Archive	15	+189%	176	+162%
	ND-Tree	40	+697%	347	+417%
1000	**AVL-Archive**	**5**		**629**	
	Sorted list	**5**		923	+47%
	NDR*-Archive	14	+176%	1,506	+140%
	ND-Tree	40	+706%	3,241	+416%

(a) $\varepsilon = 5\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**AVL-Archive**	**15**		**19**	
	NDR*-Archive	86	+493%	107	+474%
	ND-Tree	125	+763%	145	+682%
	Sorted list	**15**		1,523	+8,090%
1	**AVL-Archive**	**10**		**25**	
	NDR*-Archive	50	+412%	128	+405%
	ND-Tree	82	+739%	167	+559%
	Sorted list	**10**		1,522	+5,892%
10	**AVL-Archive**	**6**		**63**	
	NDR*-Archive	21	+250%	232	+270%
	ND-Tree	48	+710%	343	+447%
	Sorted list	**6**		1,569	+2,398%
100	**AVL-Archive**	**5**		**356**	
	NDR*-Archive	15	+191%	931	+161%
	ND-Tree	41	+713%	1,761	+394%
	Sorted list	**5**		1,950	+447%
1000	**AVL-Archive**	**5**		**3,217**	
	Sorted list	**5**		5,845	+82%
	NDR*-Archive	14	+183%	8,120	+152%
	ND-Tree	41	+721%	16,454	+411%

(b) $\varepsilon = 5\%$, $N_{nd} = 50000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**AVL-Archive**	**12**		**3**	
	NDR*-Archive	73	+501%	16	+486%
	ND-Tree	99	+719%	23	+714%
	Sorted list	**12**		62	+2,129%
1	**AVL-Archive**	**8**		**4**	
	NDR*-Archive	42	+448%	19	+428%
	ND-Tree	65	+749%	25	+606%
	Sorted list	**8**		63	+1,650%
10	**AVL-Archive**	**4**		**8**	
	NDR*-Archive	16	+316%	29	+282%
	ND-Tree	35	+803%	46	+508%
	Sorted list	**4**		68	+800%
100	**AVL-Archive**	**3**		**46**	
	Sorted list	**3**		115	+152%
	NDR*-Archive	11	+263%	128	+180%
	ND-Tree	31	+879%	251	+448%
1000	**AVL-Archive**	**3**		**414**	
	Sorted list	**3**		582	+41%
	NDR*-Archive	11	+251%	1,072	+159%
	ND-Tree	30	+891%	2,318	+460%

(c) $\varepsilon = 25\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**AVL-Archive**	**15**		**19**	
	NDR*-Archive	86	+493%	114	+502%
	ND-Tree	125	+762%	141	+642%
	Sorted list	**15**		1,516	+7,881%
1	**AVL-Archive**	**9**		**21**	
	NDR*-Archive	48	+451%	124	+501%
	ND-Tree	77	+782%	154	+648%
	Sorted list	**9**		1,521	+7,282%
10	**AVL-Archive**	**4**		**47**	
	NDR*-Archive	18	+329%	189	+306%
	ND-Tree	38	+830%	272	+483%
	Sorted list	**4**		1,555	+3,237%
100	**AVL-Archive**	**3**		**252**	
	NDR*-Archive	12	+272%	718	+185%
	ND-Tree	31	+889%	1,306	+419%
	Sorted list	**3**		1,803	+617%
1000	**AVL-Archive**	**3**		**2,079**	
	Sorted list	**3**		4,135	+99%
	NDR*-Archive	11	+259%	5,617	+170%
	ND-Tree	31	+912%	11,524	+454%

(d) $\varepsilon = 25\%$, $N_{nd} = 50000$

Table 3.2 – Performance of archives for $p = 2$.

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	111	+12%	**32**	
	ND-Tree	**99**		43	+33%
1	**NDR*-Archive**	**78**		**46**	
	ND-Tree	82	+5%	63	+36%
10	**NDR*-Archive**	**48**		**140**	
	ND-Tree	70	+46%	241	+72%
100	**NDR*-Archive**	**37**		**1,003**	
	ND-Tree	68	+83%	2,200	+120%
1000	**NDR*-Archive**	**33**		**9,744**	
	ND-Tree	70	+110%	25,690	+164%

(a) $\varepsilon = 5\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	112	+12%	**40**	
	ND-Tree	**101**		47	+18%
1	**NDR*-Archive**	**67**		**47**	
	ND-Tree	68		59	+25%
10	**NDR*-Archive**	**31**		**92**	
	ND-Tree	43	+41%	154	+67%
100	**NDR*-Archive**	**23**		**523**	
	ND-Tree	40	+78%	1,198	+129%
1000	**NDR*-Archive**	**21**		**4,417**	
	ND-Tree	42	+102%	12,064	+173%

(b) $\varepsilon = 5\%$, $N_{nd} = 50000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	160	+12%	**279**	
	ND-Tree	**142**		328	+17%
1	**NDR*-Archive**	**102**		**351**	
	ND-Tree	103		435	+24%
10	**NDR*-Archive**	**50**		**906**	
	ND-Tree	75	+49%	1477	+63%
100	**NDR*-Archive**	**37**		**6,176**	
	ND-Tree	72	+94%	13,823	+124%
1000	**NDR*-Archive**	**34**		**57,559**	
	ND-Tree	73	+113%	146,893	+155%

(c) $\varepsilon = 25\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	159	+12%	**274**	
	ND-Tree	**142**		319	+17%
1	**NDR*-Archive**	90		**297**	
	ND-Tree	**88**		371	+25%
10	NDR*-Archive	**35**		**510**	
	ND-Tree	49	+40%	886	+74%
100	**NDR*-Archive**	**23**		**2,549**	
	ND-Tree	43	+87%	6,248	+145%
1000	**NDR*-Archive**	**22**		**23,170**	
	ND-Tree	44	+103%	62,815	+171%

(d) $\varepsilon = 25\%$, $N_{nd} = 50000$

Table 3.3 – Performance of archives for $p = 3$.

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	169		**50**	
	ND-Tree	**145**	-14%	65	+30%
1	**NDR*-Archive**	143		**86**	
	ND-Tree	**131**		109	+27%
10	**NDR*-Archive**	**127**		**461**	
	ND-Tree	141	+11%	659	+43%
100	**NDR*-Archive**	**129**		**4,288**	
	ND-Tree	173	+34%	8,087	+89%
1000	**NDR*-Archive**	**112**		**42,338**	
	ND-Tree	192	+72%	104,332	+146%

(a) $\varepsilon = 5\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	295	+23%	**472**	
	ND-Tree	**239**		554	+18%
1	**NDR*-Archive**	227		**737**	
	ND-Tree	**214**		928	+26%
10	**NDR*-Archive**	**190**		**3,649**	
	ND-Tree	223	+18%	5,833	+60%
100	**NDR*-Archive**	**196**		**41,979**	
	ND-Tree	262	+33%	78,857	+88%
1000	NDR*-Archive	25		**434,130**	
	ND-Tree	**24**		908,101	+109%

(b) $\varepsilon = 5\%$, $N_{nd} = 50000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	181	+19%	**62**	
	ND-Tree	**152**		72	+15%
1	**NDR*-Archive**	126		**79**	
	ND-Tree	**123**		102	+30%
10	**NDR*-Archive**	**82**		**231**	
	ND-Tree	103	+26%	397	+71%
100	**NDR*-Archive**	**79**		**1,908**	
	ND-Tree	114	+45%	4,071	+113%
1000	**NDR*-Archive**	**84**		**22,351**	
	ND-Tree	127	+51%	52,605	+135%

(c) $\varepsilon = 25\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (ms)	
0	**NDR*-Archive**	293	+20%	**511**	
	ND-Tree	**244**		575	+12%
1	**NDR*-Archive**	188		**639**	
	ND-Tree	**177**		772	+21%
10	**NDR*-Archive**	**100**		**1,599**	
	ND-Tree	124	+24%	2,603	+63%
100	**NDR*-Archive**	**89**		**11,585**	
	ND-Tree	127	+42%	24,626	+113%
1000	**NDR*-Archive**	**9**		**148,460**	
	ND-Tree	11	+27%	329,107	+122%

(d) $\varepsilon = 25\%$, $N_{nd} = 50000$

Table 3.4 – Performance of archives for $p = 4$.

ϕ_{dom}	Archive	Number of comparisons		Time (s)	
0	**NDR*-Archive**	207	+10%	**0.07**	
	ND-Tree	**188**		0.08	+29%
1	**NDR*-Archive**	180		**0.12**	
	ND-Tree	**166**		0.14	+23%
10	**NDR*-Archive**	212	+20%	**0.80**	
	ND-Tree	**177**		0.94	+18%
100	**NDR*-Archive**	305	+26%	**12**	
	ND-Tree	**242**		15	+20%
1000	**NDR*-Archive**	290		**153**	
	ND-Tree	290		220	+43%

(a) $\varepsilon = 5\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (s)	
0	**NDR*-Archive**	436	+30%	**0.73**	
	ND-Tree	**335**		0.83	+14%
1	**NDR*-Archive**	352	+17%	**1.23**	
	ND-Tree	**302**		1.45	+18%
10	**NDR*-Archive**	416	+21%	**9.10**	
	ND-Tree	**344**		11.47	+26%
100	**NDR*-Archive**	588	+26%	**169**	
	ND-Tree	**467**		204	+17%
1000	**NDR*-Archive**	**46**		**200**	
	ND-Tree	49		275	+37%

(b) $\varepsilon = 5\%$, $N_{nd} = 50000$

ϕ_{dom}	Archive	Number of comparisons		Time (s)	
0	**NDR*-Archive**	240	+12%	**0.07**	
	ND-Tree	**214**		0.09	+29%
1	**NDR*-Archive**	**201**		**0.11**	
	ND-Tree	205		0.16	+40%
10	**NDR*-Archive**	**183**		**0.58**	
	ND-Tree	222	+21%	0.95	+64%
100	**NDR*-Archive**	**255**		**9**	
	ND-Tree	305	+20%	17	+84%
1000	**NDR*-Archive**	**319**		**149**	
	ND-Tree	367	+15%	271	+81%

(c) $\varepsilon = 25\%$, $N_{nd} = 10000$

ϕ_{dom}	Archive	Number of comparisons		Time (s)	
0	**NDR*-Archive**	469	+28%	**0.74**	
	ND-Tree	**367**		0.86	+17%
1	**NDR*-Archive**	351	+12%	**1.1**	
	ND-Tree	**313**		1.4	+27%
10	**NDR*-Archive**	**242**		**5**	
	ND-Tree	288	+19%	8	+67%
100	**NDR*-Archive**	**306**		**70**	
	ND-Tree	357	+17%	123	+77%
1000	**NDR*-Archive**	38		1,245	
	ND-Tree	**36**		1,887	+52%

(d) $\varepsilon = 25\%$, $N_{nd} = 50000$

Table 3.5 – Performance of archives for $p = 5$.

Note that the RAM utilization of both structures is, in average over all tested instances, linear in the number of solutions stored.

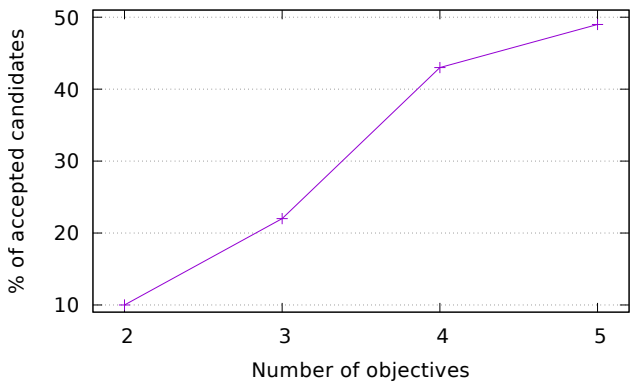
3.3.4 Experiments on PLS

The second bench of experiments tests the use of archives inside PLS, applied to MOTSP. As previously noted in Section 2.2.3.5, the quality of the starting solution set is of prime importance for the convergence speed of PLS. Thus, instead of producing a starting set of solutions randomly generated, we follow the 2PPLS framework (Algorithm 7) which is standard nowadays:

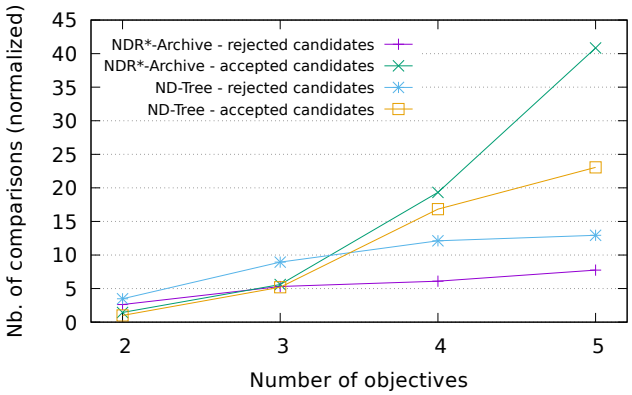
- We first generate an initial approximation of the efficient set by solving a number of single-objective TSP through a linear aggregation of the objectives and a pre-defined set of weights using MDW [Steuer, 1986]. Each single-objective TSP is optimized using the chained Lin-Kernighan heuristic [Applegate et al., 2003]. Then we start PLS from this initial set of solutions.

- The neighborhood used in PLS is a 2-opt with complete exploration strategy and candidate lists, such that an edge is candidate only if it composes a solution of the starting set of PLS.

Let us first introduce the MOTSP instances considered, then describe the experimental methodology and the different archives tested.



(a) Evolution of the proportion of accepted candidates.



(b) Evolution of the number of dominance comparisons in NDR*-Archive and ND-Tree.

Figure 3.18 – Evolution of the proportion of accepted candidates (a) and number of dominance comparisons in NDR*-Archive and ND-Tree (b) in function of the number of objectives ($N_{nd} = 10000$, $\phi_{dom} = 10$, $\varepsilon = 5\%$).

3.3.4.1 Benchmark instances

The benchmark is composed of 240 ($= 3 \times 4 \times 20$) MOTSP instances of 3 different sizes n for each number of objectives $p = 2, ..., 5$. Each instance has been generated with the same construction processes as in the literature ([Reinelt, 1991, Lust and Jaszkiewicz, 2010]):

- An instance is composed of p single-objective Euclidean instances
- For each objective, the coordinates of each city are integers that are uniformly and independently generated in the range $[0; 3163]$, and the costs between the edges correspond to the Euclidean distance between two cities on the plane.

For each combination of size and number of objectives, 20 instances are randomly generated.

Note that the size of the instances has been substantially reduced according to the number of objectives because the number of incomparable points generally increase significantly with a growing number of objectives.

3.3.4.2 Experimental methodology and results

As indicated in Section 2.2.3.5, **two sub-versions of PLS** have been proposed:

- the original version (without local archive);
- the version with local archive.

We propose to compare these two sub-versions, as to our knowledge, it has never been done and might be interesting as we will see. We also intend to compare different combinations of archives. We considered as **local archives** the sorted list for $p = 2$, and the simple list with Move-to-Front heuristic (denoted *List-MF*) for $p \geq 3$.

As **main archives**, we compared:

n	$\lvert X_{all}\rvert$	ϕ_{dom}	Archive	Local archive	Number of comparisons		PLS time (s)	
100	1730	494	**SAAVLA**	-	4.0	+62%	**0.13**	
			AVL-Archive	-	5.0	+106%	0.13	
			AVL-Archive	Sorted list	2.5		0.14	+14%
			SAAVLA	Sorted list	2.5		0.14	+15%
			Sorted list	Sorted list	2.5		0.14	+15%
			Sorted list	-	4.8	+96%	0.15	+16%
			SAAVLA	AVL-Archive	**2.4**		0.15	+20%
			SANDRA	-	5.5	+123%	0.15	+21%
			AVL-Archive	AVL-Archive	2.5		0.15	+22%
			SANDRA	Sorted list	3.3	+33%	0.16	+23%
			NDR*-Archive	Sorted list	4.1	+66%	0.16	+24%
			ND-Tree	Sorted list	5.8	+138%	0.16	+30%
			NDR*-Archive	-	23.4	+855%	0.20	+63%
			ND-Tree	-	43.7	+1,682%	0.31	+148%
500	33504	5262	**SAAVLA**	**-**	3.8	+86%	**29**	
			SAAVLA	AVL-Archive	**2.0**		30	
			AVL-Archive	AVL-Archive	2.1		30	
			AVL-Archive	Sorted list	2.1		31	
			SANDRA	Sorted list	2.3	+10%	31	
			SAAVLA	Sorted list	2.1		31	
			NDR*-Archive	Sorted list	2.6	+26%	31	
			Sorted list	Sorted list	2.1		32	+11%
			ND-Tree	Sorted list	4.0	+97%	32	+12%
			AVL-Archive	-	7.0	+244%	33	+14%
			SANDRA	-	4.6	+125%	33	+16%
			Sorted list	-	6.7	+229%	38	+30%
			NDR*-Archive	-	34.3	+1,577%	53	+84%
			ND-Tree	-	88.6	+4,232%	95	+229%
1000	113696	17105	**SAAVLA**	**AVL-Archive**	**2**		**311**	
			SAAVLA	-	3.8	+88%	312	
			AVL-Archive	AVL-Archive	2		316	
			SANDRA	Sorted list	2.1		326	
			AVL-Archive	Sorted list	2.1		326	
			SAAVLA	Sorted list	2		330	
			NDR*-Archive	Sorted list	2.3	+17%	329	
			ND-Tree	Sorted list	3.3	+67%	342	+10%
			Sorted list	Sorted list	2.1		353	+13%
			SANDRA	-	4.5	+124%	359	+15%
			AVL-Archive	-	7.9	+296%	375	+20%
			Sorted list	-	7.6	+282%	435	+40%
			NDR*-Archive	-	37.7	+1,790%	614	+97%
			ND-Tree	-	111	+5,467%	1,303	+319%

Table 3.6 – Comparison between the different archive types on PLS with $p = 2$.

n	$\lvert X_{all}\rvert$	ϕ_{dom}	Archive	Local archive	Number of comparisons		PLS time (s)	
50	93590	677	**SANDRA**	**-**	**14.9**		**18**	
			SANDRA	List-MF	27.3	+84%	30	+67%
			NDR*-Archive	List-MF	34.4	+132%	32	+77%
			NDR*-Archive	-	73.9	+397%	35	+93%
			ND-Tree	List-MF	53.1	+257%	43	+140%
			ND-Tree	-	168.1	+1,031%	84	+366%
75	374393	1303	**SANDRA**	**-**	**13.9**		**136**	
			SANDRA	List-MF	26.7	+92%	227	+67%
			NDR*-Archive	List-MF	33.2	+139%	241	+77%
			NDR*-Archive	-	83.0	+497%	293	+116%
			ND-Tree	List-MF	56.5	+307%	335	+147%
			ND-Tree	-	236.7	+1,603%	809	+496%
100	817820	1955	**SANDRA**	**-**	**12.7**		**432**	
			SANDRA	List-MF	24.0	+89%	687	+59%
			NDR*-Archive	List-MF	29.5	+132%	726	+68%
			NDR*-Archive	-	86.3	+579%	973	+125%
			ND-Tree	List-MF	52.1	+310%	1058	+145%
			ND-Tree	-	273.2	+2,049%	3297	+662%

Table 3.7 – Comparison between the different archive types on PLS with $p = 3$.

- only for $p = 2$: AVL-Archive, SAAVLA (self-adjusting version of AVL-Archive), the sorted list;
- for $p \geq 2$: NDR*-Archive, SANDRA (self-adjusting version of NDR*-Archive), ND-Tree.

Results are shown in Tables 3.6 ($p = 2$), 3.7 ($p = 3$), 3.8 ($p = 4$), and 3.9 ($p = 5$). The first column of each table shows the instance size n, the second column indicates the average size of the final approximation set X_{all} found, the third column indicates the proportion of dominated points ϕ_{dom}, computed as the average number of dominated solutions presented to the archive divided by the number of (potentially) non-dominated points $|X_{all}|$, the fourth column indicates the main archive used, next are displayed: the local archive used, the number of dominance comparisons, finally the execution time of PLS.

Overall, for $p = 2$, results are pretty homogeneous in terms of computational time and number of comparisons when a local archive is used. More precisely, better performance are obtained by using a local archive, except for SAAVLA which performs similarly with or without it.

Globally, AVL-Archive and SAAVLA performs slightly better than the sorted list (with local archive), the latter being around 10-15% slower. Given these results, we tested AVL-Archive as main *and* local archive, and found better results than using the sorted list as local archive.

Finally, NDR*-Archive and SANDRA obtain slightly better results than ND-Tree with local archive, and similar or better results than the sorted list with local archive. ND-tree without local archive is completely outperformed by the other archives.

For $p \in \{3, 4\}$, SANDRA without local archive is clearly the most effective combination, followed by SANDRA with local archive and NDR*-Archive also with local archive. As usual, ND-Tree without local archive is outperformed by any other archive, and its version with local archive is approximately 150% slower than SANDRA. Concerning the number of comparisons, SANDRA with or without local archive outperforms the other archives.

For $p = 5$, SANDRA is still better than the other archives, but slightly better performance is obtained by using a local archive. The second best archive is still NDR*-Archive (with local archive). ND-Tree with local

n	$\lvert X_{all}\rvert$	ϕ_{dom}	Archive	Local archive	Number of comparisons		PLS time (s)	
20	50369	193	**SANDRA**	**-**	**60**		**9**	
			SANDRA	List-MF	67	+11%	12	+30%
			NDR*-Archive	List-MF	102	+69%	15	+64%
			NDR*-Archive	-	169	+180%	18	+97%
			ND-Tree	List-MF	139	+130%	19	+112%
			ND-Tree	-	266	+342%	31	+239%
25	187127	311	**SANDRA**	**-**	**64**		**60**	
			SANDRA	List-MF	73	+14%	79	+32%
			NDR*-Archive	List-MF	119	+86%	105	+76%
			ND-Tree	List-MF	168	+163%	138	+131%
			NDR*-Archive	-	222	+246%	143	+141%
			ND-Tree	-	370	+477%	251	+322%
30	446172	432	**SANDRA**	**-**	**63**		**205**	
			SANDRA	List-MF	78	+20%	280	+36%
			NDR*-Archive	List-MF	132	+89%	389	+90%
			ND-Tree	List-MF	190	+162%	505	+146%
			NDR*-Archive	-	261	+253%	575	+180%
			ND-Tree	-	446	+488%	989	+383%

Table 3.8 – Comparison between the different archive types on PLS with $p = 4$.

n	$\lvert X_{all}\rvert$	ϕ_{dom}	Archive	Local archive	Number of comparisons		PLS time (s)	
10	1790	57	**SANDRA**	**-**	87	+26%	**0.11**	
			SANDRA	List-MF	**70**		0.12	
			NDR*-Archive	List-MF	87	+25%	0.12	
			NDR*-Archive	-	138	+99%	0.14	+22%
			ND-Tree	List-MF	111	+59%	0.18	+58%
			ND-Tree	-	198	+185%	0.26	+132%
15	81523	150	**SANDRA**	**List-MF**	**188**		**36**	
			SANDRA	-	260	+38%	41	+14%
			NDR*-Archive	List-MF	291	+55%	45	+26%
			ND-Tree	List-MF	307	+64%	52	+46%
			NDR*-Archive	-	547	+192%	76	+112%
			ND-Tree	-	596	+218%	95	+164%
20	480987	242	**SANDRA**	**List-MF**	**236**		**505**	
			SANDRA	-	307	+30%	539	
			NDR*-Archive	List-MF	404	+71%	662	+31%
			ND-Tree	List-MF	438	+85%	710	+41%
			NDR*-Archive	-	774	+227%	1041	+106%
			ND-Tree	-	899	+280%	1360	+170%

Table 3.9 – Comparison between the different archive types on PLS with $p = 5$.

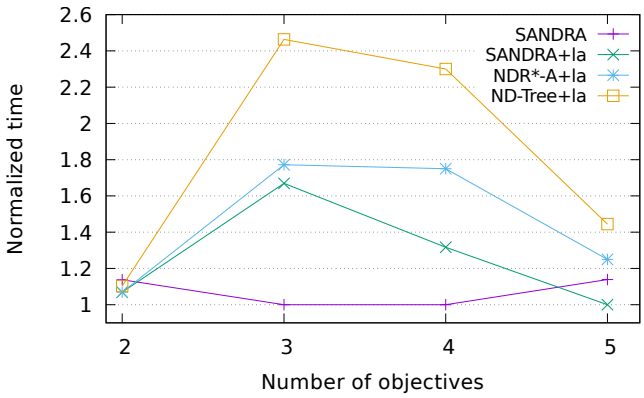


Figure 3.19 – Evolution of the computational time of NDR*-Archive and ND-Tree for PLS in function of the number of objectives ("la" stands for local archive).

archive is approximately 50% slower than SANDRA. The number of comparisons is still at the advantage of SANDRA.

Figure 3.19 summarizes the results by comparing the computational time of SANDRA, and NDR*-archive and ND-Tree with local archive in function of p.

3.4 Conclusion

We proposed two new archives, AVL-Archive specialized in the bi-objective case, and NDR*-Archive for the general case. Both structures are self-balancing trees and have two versions: a general version suited for any multi-objective optimization task; and a self-adjusting version, especially designed if a presumption of temporal and spatial locality exists between the solutions presented to the archive.

These archives have been experimentally tested and compared to other known state-of-the-art archives on a large benchmark of instances simulating the generation of points of a MO meta-heuristic, and inside a PLS for MOTSP up to 5 objectives.

AVL-Archive outperforms all the other archives in the bi-objective case on the artificial benchmark tested and its self-adjusting version performs slightly better than the sorted list when used in PLS.

NDR*-Archive performs better than competitors in the general case, particularly when $p \leq 4$ and the number of dominated points is larger than the number of non-dominated points. The self-adjusting version of NDR*-Archive (SANDRA) is much more effective than competitors when used in PLS, especially for $p = 3, 4$.

We hope that these two archives and their self-adjusting versions could be useful for future meta-heuristics, particularly those using PLS, MO dynamic programming or search zones-based methods.

Concerning the perspectives of NDR*-Archive, one could keep sorted the children of an internal node (or the solutions of a leaf) using their Hilbert key like in [Kamel and Faloutsos, 1993] (or any key provided by a space-filling curve, like the Peano curve). Like AVL-Archive which uses specific properties of the bi-objective case, it might be interesting to use the specific properties of objective spaces of fixed dimensions $p = 3, 4$ highlighted and practically applied in [Fonseca et al., 2006] and [Guerreiro et al., 2012]. More researches seem necessary to improve the efficiency of NDR*-Archive for $p \geq 5$. For example, a new split or even a new procedure detecting efficiently if a candidate and the solutions in the archive are incomparable could be necessary to obtain better results.

One possible improvement of AVL-Archive is to use splay moves [Sleator and Tarjan, 1985] to rearrange the tree so that the cache node is placed at the root of the tree. However, this modification could make the tree not balanced anymore.

Chapter 4

New optimization methods

This chapter proposes a number of new optimization methods and improvements of existing methods to tackle more efficiently MOCO problems. First are suggested some modifications of SO optimizers to improve their performance on MOCO problems. Based on these suggestions, Aggregation-based NMCS (A-NMCS) is a new version of NMCS especially designed for MOCO. Second, a more adaptive and generalized version of MDW called Adaptive MDW (A-MDW) is presented. Third, we introduce Partitioned Pareto Local Search (P-PLS) which aims at speeding-up PLS-VND. Then, we propose a new MO meta-heuristic called 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D). Finally, a new system based on the ϵ-archive concept is presented.

Introduction

First, Section 4.1 suggests a number of improvements for SO optimizers and a new version of NMCS adapted to MOCO, Aggregation-based Nested Monte Carlo Search (A-NMCS). Second, Section 4.2 proposes Adaptive Maximally Dispersed set of Weights (A-MDW), a more adaptive and generalized version of MDW. Third, Section 4.3 introduces Partitioned Pareto Local Search (P-PLS), a method consisting in reducing the computational overhead of PLS. Then, Section 4.4 proposes 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D), a MO meta-heuristic based on the concepts of 2-Phase PLS, Decomposition methods and IPLS and combining A-MDW, P-PLS and the suggested improvements for SO optimizers. Section 4.5 presents a methodology to bound the size of the generated approximation set while guaranteeing a good distribution in the objective space of the points generated. Finally, we summarize the chapter.

4.1 Single-objective optimizers: improvements for MOCO

We saw that state-of-the-art MO meta-heuristics presented in Section 2.2 often optimize aggregation functions during their optimization process. We are particularly interested by weighted sum aggregations, as SO optimizers are directly able to solve the related problem. Indeed, efficient domain-specific SO optimizers which have already proved their reliability are generally available, even for hard problems. This section suggests some simple modifications on SO optimizers to greatly improve their ability for finding efficient solutions of MOCO problem instances. There are three main suggestions:

1. The first suggestion is to *systematically* start/guide the search of the SO optimizer from an already found solution, if this option is made available by the SO optimizer or even simple modifications in implementation can make it possible. In fact, this option is often either available or implementable: naturally in local search methods (see [Paquete and Stützle, 2003] for example), but also in construction methods, like Monte Carlo Search methods as we will see later in this section. The aim is to start/guide the search process from a good quality solution instead of a randomly greedily generated one.

2. The second suggestion proposes the application of data perturbation on the cost function of the addressed weighted-sum problem. In fact, by optimizing weighted sum problems, the SO optimizer tends to focus its search toward supported solutions. Applying data perturbation on the cost function modifies the search direction of the SO optimizer, and potentially directs the search on any type of solutions (either supported or non-supported).

As previously indicated in Section 2.2.3, data perturbation has been introduced in MOCO optimization in [Lust and Teghem, 2010] for bi-objective TSP, but has not been reused after this paper, probably because it requires setting 3 parameters. We propose a simpler version of data perturbation (initially introduced in our work [Cornu et al., 2017]) which needs a single parameter.

Given a weight $\lambda \in \Lambda_0$ and a small real $\delta > 0$ controlling the maximum variation of the noise called *data perturbation coefficient*, data perturbation generates from the MO cost function $c : E \to \mathbb{R}^p$ the new SO *aggregated and perturbed cost function* $c^{\lambda,\delta} : E \to \mathbb{R}$ such that:

$$c^{\lambda,\delta}(e) = \nu \times \sum_{j=1}^{p} \lambda_j c_j(e) \ , \ \forall e \in E$$

where $\nu \sim \mathcal{U}(1-\delta, 1+\delta)$ is a real number taken from a uniform distribution in the range $[1-\delta, 1+\delta]$. The higher δ, the larger the perturbation is. Once perturbation has been performed, the SO optimizer solves the

data perturbed weighted sum problem $\lambda f^{\delta}(\cdot)$ derived from $c^{\lambda,\delta}$. Naturally, $\lambda f^{\delta}(\cdot)$ with $\delta = 0$ corresponds to a classical (non-data perturbed) weighted sum problem: $\lambda f^{0}(\cdot) = \lambda f(\cdot)$, therefore $\lambda f^{\delta}(\cdot)$ is a generalization of $\lambda f(\cdot)$.

3. The last suggestion is, to our knowledge, a novel proposition (initially introduced in our work [Cornu et al., 2017]) which consists in memorizing all relevant generated solutions in an archive during the whole run of a SO optimizer, instead of returning the best found solution for the addressed weighted-sum problem. The idea behind this suggestion is that, in general a SO optimizer finds several different solutions during its optimization process. While these solutions might be of lowest quality than the best found solution for the addressed weighted-sum problem, they could be efficient for the MOCO problem. This suggestion involves minor (and non-problem specific) modifications on the SO optimizer, by contrast it enables a much larger exploration of the search space of the MOCO problem. As an example, if the SO optimizer is an ILS, then the idea is to memorize in an archive all local optima obtained after each LS descent phase.

4.1.1 Aggregation-based Nested Monte Carlo Search

Let us introduce a new version of NMCS called Aggregation-based Nested Monte Carlo Search (A-NMCS) implementing the three suggestions presented above for SO optimizers. A-NMCS is described in pseudo-code by Algorithm 27 and works like NMCS with three differences (indicated with red color):

1. It optimizes a data perturbed weighted sum problem $\lambda f^{\delta}(\cdot)$, $\delta \geq 0$.

2. The first call of A-NMCS can take as input parameter a sequence of actions seq_{best} and its related solution x_{best}. This allows the highest level of A-NMCS to follow, from the very beginning of the search, a sequence previously found by another run of A-NMCS, typically optimizing another but very similar data perturbed weighted sum problem, such that the starting sequence it provides is of good quality for both problems.

3. A global archive $\overline{X}_{all}$ is maintained and any new solution generated by random simulations is presented to $\overline{X}_{all}$.

4.2 Adaptive Maximally Dispersed set of Weights

This section presents a new initialization procedure for any MO meta-heuristic, aiming at generating a starting approximation of the efficient set. It uses a concept extensively used in initialization phases of a number of meta-heuristics described in Section 2.2, in particular in the first phase of 2-phase PLS methods (2PPLS and TP+PLS) or during the initialization of methods based on decomposition (MOEA/D and MoMad): first a number of weights are generated, then the corresponding weighted sum problems are solved with a SO optimizer, and the generated solutions are memorized in an archive. While an efficient and problem-specific SO optimizer is mandatory to generate good quality points *towards* the non-dominated set, the diversification of the weights in the weight space is fundamental as it allows to generate points in the objective space well dispersed *along* the non-dominated set. We are interested in the second point.

The new method we introduce, called Adaptive Maximally Dispersed set of Weights (A-MDW), proposes to generalize MDW and make it more adaptive.

Algorithme 27 : `A-NMCS`

Input : $level$, data perturbed weighted sum problem $\lambda f^{\delta}(\cdot)$, starting state s_{start}, best sequence of actions seq_{best}, best solution x_{best}

Output : seq_{best}, x_{best}, global archive $\overline{X}_{all}$

$s \leftarrow s_{start}$
if $level = 0$ **then**
  $(seq_{best}, x_{best}) \leftarrow$`RandomSimulation(`$s$`)`
  return $(seq_{best}, x_{best}, \{x_{best}\})$
$\overline{X}_{all} \leftarrow \emptyset$
$i \leftarrow 1$
repeat
  foreach $a \in$ `Actions(`s`)` **do**
    $s \leftarrow$`PerformAction(`a, s`)`
    $(seq, x, \overline{X}) \leftarrow$`A-NMCS(`$level - 1$, $\lambda f^{\delta}(\cdot)$, s, none, none`) // Like in (vanilla) NMCS, the best seq. and the best sol. are not given to lower levels.`
    $s \leftarrow$`UnperformAction(`a, s`)`
    $\overline{X}_{all} \leftarrow$ `AddAll(`$\overline{X}$, $\overline{X}_{all}$`)`
    if $\lambda f^{\delta}(x) < \lambda f^{\delta}(x_{best})$ **then**
      $x_{best} \leftarrow x$
      $seq_{best} \leftarrow seq$
  $a_{best} \leftarrow seq_{best}[i{+}{+}]$
  $s \leftarrow$`PerformAction(`a_{best}, s`)`
until `IsTerminal(`s`)`;
return $(seq_{best}, x_{best}, \overline{X}_{all})$

4.2.1 Motivations

MDW is largely used in the MO meta-heuristic research field as it is a *simple* and *easily implementable* method which generates *well dispersed weights* in the weight space. Unfortunately, MDW has two main drawbacks:

1. The parameter D, controlling the number of weights generated, has to be fixed. The value of this parameter is difficult to set because it depends on the instance characteristics of the addressed problem, like the number of objectives, the instance size and more complex features depending on the instance treated. In general, the user will set experimentally a D value for each tested instance of the addressed problem, which is a time-consuming offline task and mostly can not be the best setting.

2. A number of weights generated by MDW contain a 0 to some of their components, and thus do not evaluate the related objectives. This phenomenon is exacerbated as the number of objectives grows.

A-MDW circumvents the two issues of MDW depicted above by:

1. Replacing the original parameter D by a positive real α called the *minimum acceptance rate threshold*. The idea is to iteratively generate equally dispersed weights like in MDW with an increasing parameter number $D = 1, 2, ...$ controlling the number and the compactness of the generated weights, and stop once the solver does not generate enough new non-dominated solutions so that continuing to optimize weighted sum problems is no longer efficient.

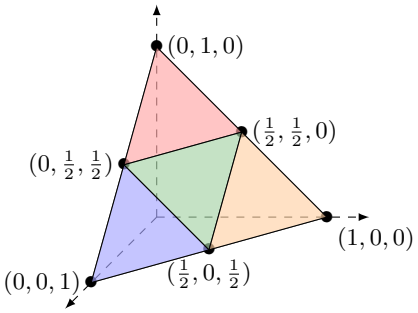


Figure 4.1 – Edgewise Simplex subdivision of Λ_0 with $b = 2$ ($p = 3$).

2. Contrary to MDW which generates the weights only from Λ_0, A-MDW can generate the weights from any simplex and thus generating weights with component values larger than or equal to $\varepsilon \geq 0$.

4.2.2 Simplex subdivision

A-MDW uses as routine the Edgewise Simplex Subdivision of Gonçalves et al. [Gonçalves et al., 2006] to generate its weights. Given a (p-1)-dimensional simplex Λ, the method subdivides Λ into b^{p-1} (p-1)-dimensional simplices with same volume and shape characteristics. The so-called *base* $b \geq 0$ is the integer number by which an edge of the simplex is subdivided. Next we describe the procedure with any b, but we will use it only with $b = 2$. The method works as follows: given Λ, each edge of the simplex is cut in b parts; then Λ is subdivided into b^{p-1} smaller simplices, such that each pair of smaller simplices is either disjoint or meets along a common face [Edelsbrunner and Grayson, 2000]. The decomposition process is independent of the input simplex. Figure 4.1 describes the subdivision of Λ_0 with $b = 2$ and for $p = 3$, where $2^{3-1} = 4$ new simplices are generated.

Algorithm 28 shows how the method works to subdivide a simplex Λ with any base b in a p-dimensional space. It returns the list $L\Lambda$ of new simplices resulting of the subdivision of Λ. Although concise, the algorithm may seem pretty hard to understand at first sight, thus we propose an illustrated example in Figure 4.2 detailing the subdivision of a simplex with $b = 2$ in a tri-dimensional space ($p = 3$).

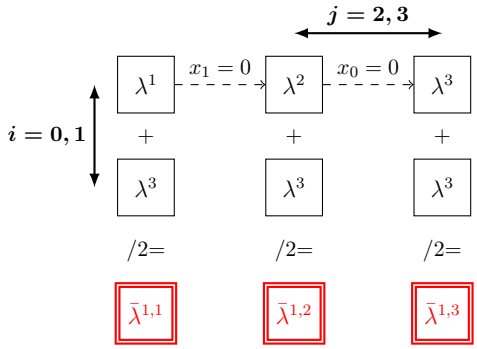
We assume that $\Lambda = \triangle(\lambda^1, ..., \lambda^p)$, where $\lambda^j \in \mathbb{R}^p, j = 1, ..., p$ are vertices of Λ. The algorithm builds the b^{p-1} new simplices *one at a time* (loop iterating on s, line 3).

Let $\bar{\lambda}^{s,1}, ..., \bar{\lambda}^{s,p} \in \mathbb{R}^p$ be the future vertices of the s^{th} simplex to build (line 4), $s \in \{1, ..., b^{p-1}\}$. The fundamental principle of the algorithm is that the vertices $\bar{\lambda}^{s,j}$ are built *simultaneously* (lines 5-14) such that the value assigned to each $\bar{\lambda}^{s,j}$ is a *unique linear combination of b vertices of* Λ, $j = 1, ...p$:

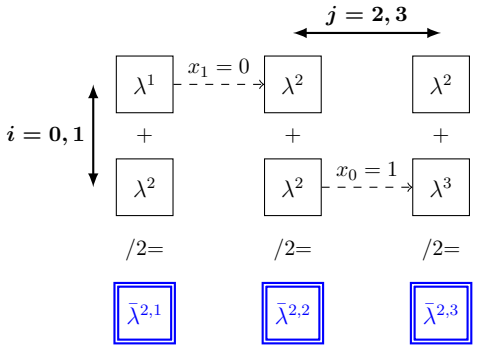
$$\bar{\lambda}^{s,j} = \frac{\sum_{i=0}^{b-1} \lambda^{\kappa(s,i,j)}}{b}, \; j = 1, ...p$$

where $\kappa(s, i, j) \in \{1, ...p\}$ is a so-called *color*, i.e. an index identifying a vertex of Λ. The values taken by $\kappa(s, i, j)$, $i = 0, ..., b-1$, $j = 1, ...p$ follow a so-called *color scheme*. A color scheme is a $b \times p$ matrix $K(s) = [\kappa(s, i, j)]$ such that each cell $\kappa(s, i, j) \in \{1, ...p\}$ ($i = 0, ..., b-1$, $j = 1, ...p$) is a color, and built such that the colors are of increasing value and the columns are pairwise different in such a way that from column $j - 1$ to column j, only one color is always increased by one.

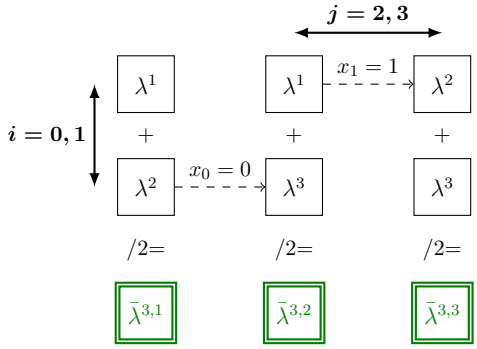
A color scheme $K(s)$ is created row-by-row starting from $\kappa(s, 0, 1) = 1$. To find out if the next entry will be kept or increased by one, it is necessary to represent the integer number $(s - 1)$ in base b with $p - 1$ digits: $x_{p-2}...x_0$ (line 5). The values of the digits x_{p-j}, $j = 2, ..., p$ determine which row i of column $j - 1$ will be



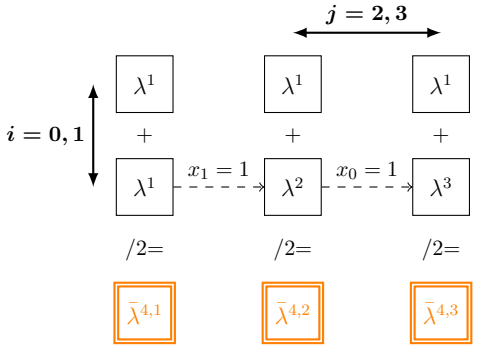
(a) 1^{st} new simplex: $s = 1 \Rightarrow (s-1)_{10} = 0_{10} = 00_2$.



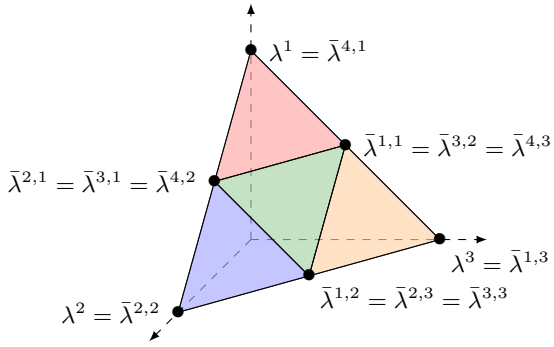
(b) 2^{nd} new simplex: $s = 2 \Rightarrow (s-1)_{10} = 1_{10} = 01_2$.



(c) 3^{nd} new simplex: $s = 3 \Rightarrow (s-1)_{10} = 2_{10} = 10_2$.



(d) 4^{th} new simplex: $s = 4 \Rightarrow (s-1)_{10} = 3_{10} = 11_2$.



(e) The resulting subdivided simplex with the new simplices.

Figure 4.2 – Illustration of the subdivision of a simplex with a base $b = 2$ and $p = 3$.

Algorithme 28 : `SimplexSubdivision`

Input : simplex to subdivide Λ, base b
Output : list of new simplices $L\Lambda$

1 Let $\Lambda = \triangle(\lambda^1, ..., \lambda^p)$
2 $L\Lambda \leftarrow \emptyset$
3 **for** $s \leftarrow 1, ..., b^{p-1}$ **do**
4 $\quad \bar{\lambda}^{s,1}, ..., \bar{\lambda}^{s,p} \leftarrow \overrightarrow{0}$
5 $\quad x_{p-2}...x_0 \leftarrow$ convert $(s-1)$ to base b
6 $\quad k \leftarrow 1$
7 $\quad$ **for** $i \leftarrow 0, ..., b-1$ **do**
8 $\quad\quad \bar{\lambda}^{s,1} \leftarrow \bar{\lambda}^{s,1} + \lambda^k$
9 $\quad\quad$ **for** $j \leftarrow 2, ..., p$ **do**
10 $\quad\quad\quad$ **if** $x_{p-j} = i$ **then**
11 $\quad\quad\quad\quad k \leftarrow k + 1$
12 $\quad\quad\quad \bar{\lambda}^{s,j} \leftarrow \bar{\lambda}^{s,j} + \lambda^k$
$\quad$ `// normalization:`
13 $\quad$ **for** $j \leftarrow 1, ..., p$ **do**
14 $\quad\quad \bar{\lambda}^{s,j} \leftarrow \bar{\lambda}^{s,j}/b$
15 $\quad L\Lambda \leftarrow L\Lambda \cup \triangle(\bar{\lambda}^{s,1}, ..., \bar{\lambda}^{s,p})$
`// returns the simplex resulting from the subdivision of` Λ`:`
16 **return** $L\Lambda$

increased by one to generate the column j (lines 10-11); and such that the following row starts with the last color in the preceding row, that is $\kappa(s, i, 1) = \kappa(s, i-1, p)$:

$$K(s) = \begin{bmatrix} \kappa(s,0,1) = 1 & \kappa(s,0,2) & \dots & \kappa(s,0,p) \\ \kappa(s,1,1) = \kappa(s,0,p) & \kappa(s,1,2) & \dots & \kappa(s,1,p) \\ \dots & \dots & \dots & \dots \\ \kappa(s,b-1,1) = \kappa(s,b-2,p) & \kappa(s,b-1,2) & \dots & \kappa(s,b-1,p) = p \end{bmatrix}$$

Finally, the j^{th} column of $K(s)$ defines the colors $\kappa(s, i, j)$, $i = 0, ..., b-1$ used to build $\bar{\lambda}^{s,j}$.

4.2.3 Design of A-MDW

The Edgewise Simplex Subdivision (with a base $b = 2$) is important for A-MDW as the *vertices of the simplices correspond to weights* if the input simplex is included in the weight space Λ_0. In addition, one can subdivide the simplex Λ, then all the newly generated simplices, etc. and repeat the process several times. We call this process an *iterative simplex subdivision*. As the Edgewise Simplex Subdivision is *independent of the input simplex*, one can start an iterative simplex subdivision from any simplex. We are particularly interested in starting an iterative simplex subdivision from a so-called *ε-restricted weight space*:

$$\Lambda_\varepsilon := \{\lambda \in \mathbb{R}^p : \lambda_j \geq \varepsilon,\ j = 1, ..., p,\ \sum_{j=1}^{p} \lambda_j = 1\},\ \varepsilon \geq 0$$

By definition, $\Lambda_\varepsilon \subseteq \Lambda_0$ for any $\varepsilon \geq 0$, and $\Lambda_0 = \Lambda_\epsilon$ with $\varepsilon = 0$. If $\varepsilon > 0$, then any weight λ generated by an iterative simplex subdivision starting from Λ_ε has strictly positive values in each objective. Figure 4.3 illustrates an ε-restricted weight space in the weight space.

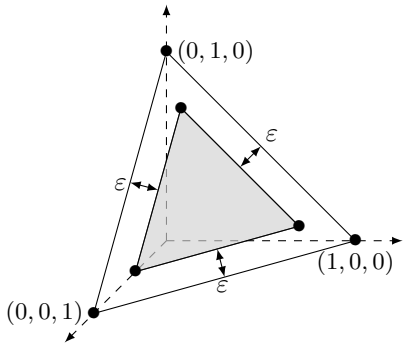


Figure 4.3 – An ε-restricted weight space in $p = 3$ s.t.
$\Lambda_\varepsilon = \triangle((1-2\varepsilon,\varepsilon,\varepsilon),\ (\varepsilon,1-2\varepsilon,\varepsilon),\ (\varepsilon,\varepsilon,1-2\varepsilon)),\ \varepsilon > 0.$

Let us now introduce the *generalized MDW* formula:

$$\begin{cases} \text{MDW}(D,\varepsilon) := \{\lambda = (\lambda_1, ..., \lambda_p) :\ \lambda \in \Lambda_\varepsilon,\ \lambda_j = \varepsilon + \frac{d}{D} \cdot (1 - p \cdot \varepsilon),\ d \in \{0, ..., D\},\ j = 1, ..., p\} \\ \text{such that} \min\limits_{\lambda^a, \lambda^b \in \Lambda_\varepsilon\ :\ \lambda^a \neq \lambda^b} \sum\limits_{j=1}^{p} \left|\lambda_j^a - \lambda_j^b\right| = \frac{2}{D} \cdot (1 - p \cdot \varepsilon) \\ \qquad 0 \leq \varepsilon < \frac{1}{p} \end{cases} \tag{4.1}$$

We can see that if $\varepsilon = 0$, then $\text{MDW}(D,\varepsilon) = \text{MDW}(D)$.

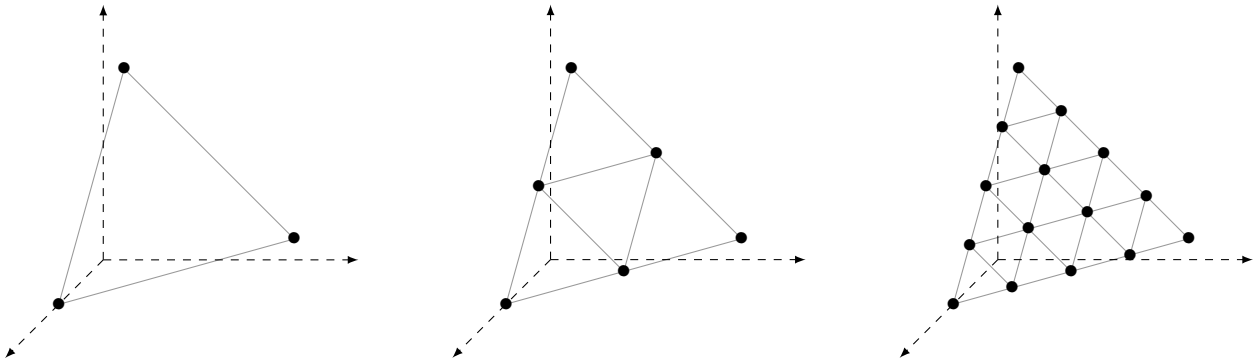
Proposition 4.2.1. Applying an iterative simplex subdivision starting from Λ_ε with base $b = 2$ and I iterations, generates the same set of weights as $\text{MDW}(2^{I-1}, \varepsilon)$.

Proof. By construction, the set of weights provided by $\text{MDW}(2^{I-1}, \varepsilon)$ is equivalent to the set of weights generated by an Edgewise Simplex Subdivision with $b = 2^{I-1}$ starting from Λ_ε [Edelsbrunner and Grayson, 2000, Gonçalves et al., 2006], which is itself equivalent to an iterative simplex subdivision with $b = 2$ and I iterations. □

Given a fixed $\varepsilon \in [0; \frac{1}{p}[$ and a number of iterations, Proposition 4.2.1 indicates that the weights generated by an iterative simplex subdivision are equally dispersed and follow the same scheme of dispersion as the generalized MDW.

The successive generation of weights provided by an iterative simplex subdivision is illustrated by Figure 4.4. Next we present in detail how our method A-MDW works.

A-MDW consists in applying an iterative simplex subdivision initially started from an ε-restricted weight space, with base $b = 2$ and $\varepsilon \in [0; \frac{1}{p}[$. At each iteration, the current simplices are subdivided, which generates a number of new weights. For each weight, the corresponding weighted sum problem is optimized and resulting solutions are stored in an archive. By construction, the new weights generated at a given iteration are built such that the minimum distance between all the weights generated so far decreases, which provides a set of weights more and more compact in the weight space. This leads to the generation in the objective space of an approximation of the non-dominated set of increasing quality through successive improvements. However, the improvement of quality induced by each iteration naturally tends to reduces when the number of iterations grows. Therefore, it is necessary to regularly use an *indicator* which evaluates the ability of the optimizer to improve the quality of the current approximation. As soon as the optimizer does not improve sufficiently the indicator value, then the procedure is stopped. Concerning the indicator to use, quality indicators, such as hypervolume or epsilon indicators, seem to be tools of choice for this task. Unfortunately, these indicators induce strong computational overhead and thus can not be used in practice.



(a) 1^{st} iteration. (b) 2^{nd} iteration. (c) 3^{rd} iteration.

Figure 4.4 – View in the weight space of weights generated by successive iterations of an iterative simplex subdivision ($p = 3, \varepsilon = 0$).

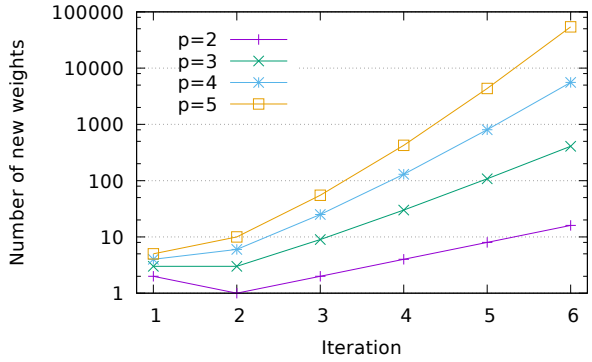


Figure 4.5 – Number of new weights generated by A-MDW in function of the number of objectives p and the total number of iterations (y-axis in log-scale). The formula computing the number of new weights generated at the I^{th} iteration in a p-dimensional weight space is $\binom{2^{I-1}+p-1}{p-1} - \binom{2^{I-2}+p-1}{p-1}$, $I \geq 2$.

Instead, we propose a simple yet efficient method which consists in collecting statistics about the total number of solutions generated by the optimizer and the total number of solutions accepted when presented to the archive. Once the percentage of accepted solutions reaches the minimum acceptance rate threshold $\alpha \in [0; 1]$ given as input parameter, then the optimizer is considered as inefficient, thus the process is stopped. We might be tempted to check it out between two iterations of the iterative simplex subdivision process only, unfortunately this option is not appropriate. Indeed, the total number of weights generated grows exponentially fast in function of both the number of iterations and the number of objectives, as it is equal to $\binom{2^{I-1}+p-1}{p-1}$, where I is the number of iterations performed. Consequently, the number of new weights generated at each iteration, and thus the number of weighted sum problems to optimize, also increases exponentially. Figure 4.5 illustrates this behavior for $p = 2, ..., 5$ during 6 iterations of A-MDW.

Instead, it seems more appropriate to check if the percentage of accepted solutions reaches the threshold α each time a weighted sum problem has been optimized. By considering this option, the procedure can be stopped during any iteration while a number of new weighted sum problems will not be optimized. Therefore, at each iteration, it is necessary to sort the new weights so as to prioritize the optimization of weighted sum problems the most promising to provide a maximum of new solutions accepted in the archive (and thus currently non-dominated). We thus propose to order the weights given the efficiency of the simplex they belong to. The efficiency of a simplex corresponds to the sum of the number of accepted solutions provided by the optimization of the weighted sum problems corresponding to the vertices of the simplex.

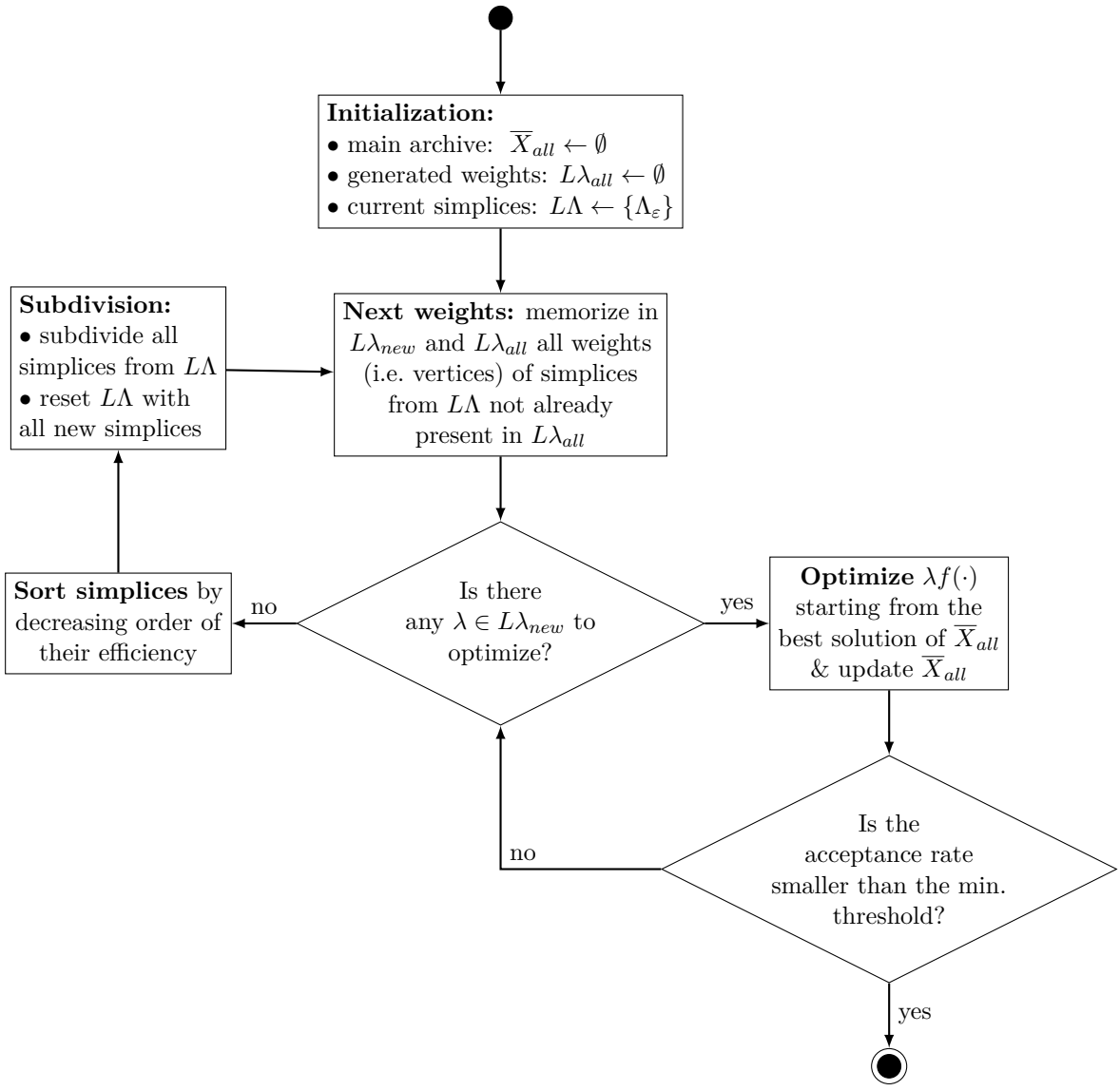


Figure 4.6 – The A-MDW procedure.

A-MDW is described in pseudo-code by Algorithm 29 and in diagram by Figure 4.6. It takes as input parameters a SO optimizer, a minimum acceptance rate threshold $\alpha \in [0;1]$, and a small real $\varepsilon \in [0;\frac{1}{p}[$ (typically $\varepsilon \leq 10^{-3}$) for defining the input ε-restricted weight space from which the iterative simplex subdivision starts. The method returns the list $L\lambda_{all}$ of all weights generated and an archive $\overline{X}_{all}$ of all incomparable solutions generated. It maintains the list of current simplices $L\Lambda$ (to be subdivided at the next iteration), initialized with the ε-restricted weight space Λ_ε.

The main loop works as follows. First, any vertex of all simplices memorized in $L\Lambda$ which has not already been considered is inserted in $L\lambda_{all}$ and $L\lambda_{new}$, the set of new weights. For each new weight, the corresponding weighted sum problem is optimized with the SO optimizer. We assume that the optimizer may provide several different solutions in output and is able to take a solution in input. Then the statistics concerning the sum η_{accept} of solutions accepted in the global archive $\overline{X}_{all}$ and the total sum η_{total} of solutions generated by the optimizer are updated. If the rate $\frac{\eta_{accept}}{\eta_{total}}$ has reached the minimum acceptance rate threshold α, then the method stops and returns the set of all weights $L\lambda_{all}$ and the archive $\overline{X}_{all}$.

Once all new weights have been considered, then the simplices stored in $L\Lambda$ are sorted given their efficiency. Finally, the simplices stored in $L\Lambda$ are subdivided, then $L\Lambda$ is reset with the new simplices. And the same process can continue.

4.3 Partitioned Pareto Local Search

This section presents a new iterative method based on PLS-VND called Partitioned Pareto Local Search (P-PLS). P-PLS consists in partitioning the archive to explore into a number of sub-archives, and building for each one a restricted neighborhood structure used by PLS-VND. The partitioning is made only once at initialization, and each sub-archive is made such that its solutions are neighbors in the objective space and its initial size is limited to a maximum number of solutions (defined by the user). Figure 4.7 illustrates the partitioning of an archive in a tri-objective space.

At each iteration of P-PLS, for each sub-archive, we extract the regularities found in the decision space between its solutions and build a model from these regularities. Typically, the model may consist in a statistical model or more simply a pool of rules on the elements composing the solutions. For example, it might be interesting to detect if the same element is present in all solutions of a sub-archive, or on the contrary, if some elements are completely absent from the sub-archive. The model is used to build a so-called *neighborhood restriction structure*. A neighborhood restriction structure is a function $\mathcal{R} : \mathcal{N} \longmapsto \mathcal{N}$ which modifies a neighborhood structure $\mathcal{N}$ by reducing the search space of $\mathcal{N}$ in order to direct the search of new solutions in a restricted but promising part of the neighborhood of a solution, and thus speeding-up PLS-VND with a minimal reduction of quality of the generated approximation. For instance, if we keep the example of the component present in all solutions of a sub-archive, it may be judicious to forbid or at least to avoid new solutions without this component.

PLS-VND is then conducted from the archive to explore (corresponding to the union of the sub-archives). During a PLS-VND run, the new solutions inherit the neighborhood restriction structure from their so-called *parent solution*. A solution x parent of another solution x' is a solution from which a LS move has been performed and has led to the creation of x'. Once PLS-VND is terminated, the archive to explore has been updated with new solutions, but not sub-archives, thus they are updated accordingly. Then the neighborhood restriction structure of each sub-archive $\overline{X}P$ is *widened*. A *widening phase* consists in rebuilding a neighborhood restriction structure by considering in addition to $\overline{X}P$, the solutions of sub-archives close to $\overline{X}P$ in the objective space. A widening phase aims at enlarging the neighborhood restriction structure and therefore increasing the search space exploration. Sub-archives used to build a neighborhood restriction structure are called its *sources*. Figure 4.8 illustrates an example of widening phases centered on a sub-archive.

Algorithme 29 : `A-MDW`

Input : SO optimizer `SO-Optimizer`, min. acceptance rate threshold α, real ε

Output : list of generated weights $L\lambda_{all}$, global archive $\overline{X}_{all}$

`// Initialization:`

$\eta_{total}, \eta_{accept} \leftarrow 0$

$N_{accept} : \mathbb{R}^p \longrightarrow \mathbb{N}$ `// maps to each weight (corresponding to a weighted sum pb. optimized), the number of solutions accepted in the archive`

$L\lambda_{all} \leftarrow \emptyset$

$\overline{X}_{all} \leftarrow \emptyset$

$L\Lambda \leftarrow \{\Lambda_\varepsilon\}$

while true **do**

  $L\lambda_{new} \leftarrow \emptyset$

  `// Next weights:`

  foreach $\Lambda = \triangle(\lambda^1, ..., \lambda^p) \in L\Lambda$ **do**

    for $j \leftarrow 1, ..., p$ **do**

      if $\lambda^j \notin L\lambda_{all}$ **then**

        $L\lambda_{all} \leftarrow L\lambda_{all} \cup \{\lambda^j\}$

        $L\lambda_{new} \leftarrow L\lambda_{new} \cup \{\lambda^j\}$

  `// Optimize weighted sum problems:`

  foreach $\lambda \in L\lambda_{new}$ **do**

    `// Optimize` $\lambda f(\cdot)$`:`

    $x \leftarrow \arg\min\{\lambda f(x) : x \in \overline{X}_{all}\}$

    $\overline{X} \leftarrow$`SO-Optimizer(`$\lambda f(\cdot)$`,` x`)`

    $\overline{X}_{all} \leftarrow$`AddAll(`$\overline{X}$`,` $\overline{X}_{all}$`)`

    `// Update statistics:`

    $\eta_{total} \leftarrow \eta_{total} + |\overline{X}|$

    $N_{accept}[\lambda] \leftarrow |\{x \in \overline{X} : x \text{ has been accepted in } \overline{X}_{all}\}|$

    $\eta_{accept} \leftarrow \eta_{accept} + N_{accept}[\lambda]$

    `// Is the acceptance rate has reached the min. threshold?`

    if $\eta_{accept} < \alpha \times \eta_{total}$ **then**

      return $(L\lambda_{all}, \overline{X}_{all})$

  `// Sort simplices:`

  sort $L\Lambda$ in decreasing order s.t. each simplex $\Lambda = \triangle(\lambda^1, ..., \lambda^p) \in L\Lambda$ has the following sorting key:

  $$key(\Lambda) = \sum_{j=1}^{p} N_{accept}[\lambda^j]$$

  $L\Lambda_{next} \leftarrow \emptyset$

  `// Subdivide simplices:`

  foreach $\Lambda \in L\Lambda$ **do**

    $L\Lambda_{next} \leftarrow L\Lambda_{next} \cup$ `SimplexSubdivision(`Λ`, 2)`

  $L\Lambda \leftarrow L\Lambda_{next}$

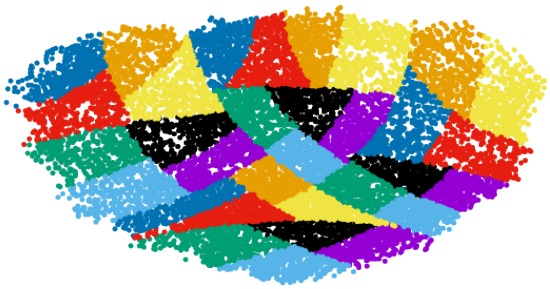


Figure 4.7 – View in a tri-objective space of the partitioning of an archive into 32 sub-archives (tri-objective TSP instance).

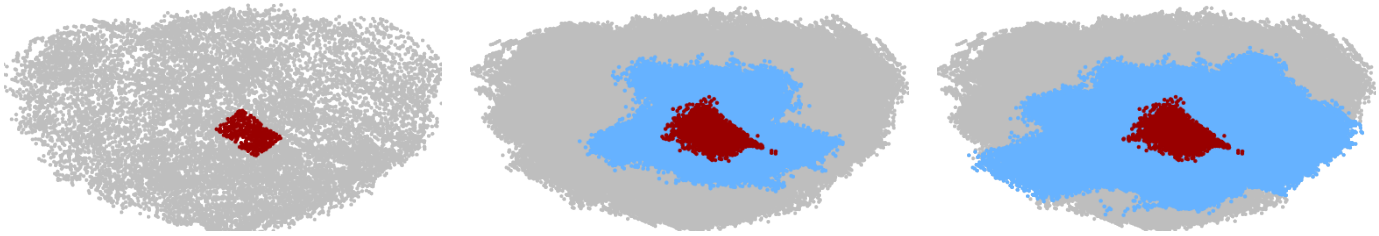


Figure 4.8 – Illustration of two widening phases related to a sub-archive (tri-objective TSP instance). The figure distinguishes the sub-archive of interest (in red), its source (in blue) and all the other sub-archives (in gray).

As it induces a computational overhead, we expect from widening of a neighborhood restriction structure that it leads to an improvement of the quality of the corresponding sub-archive during the next PLS-VND run. If no improvement of quality has been noted on a given sub-archive after a PLS-VND run, then this indicates that the extra informations gathered on solutions from new sources was useless for PLS-VND to find new solutions. Therefore, no more widening will be performed from its corresponding neighborhood restriction structure and thus no solutions belonging to this sub-archive will be explored anymore. Consequently, the widening of a given neighborhood restriction structure is performed only if its corresponding sub-archive has generated at least one new non-dominated solution during the previous PLS-VND run.

Once the widened neighborhood restriction structure has been built for each sub-archive, PLS-VND is conducted again from the archive to explore. During this new run, the (restricted) neighborhood of any new solution (i.e. newly generated since the previous run) is completely explored; while for old solutions, only newly accessible parts of the (restricted) neighborhood are explored.

This process of alternating PLS-VND with a widening phase at each iteration continues until a PLS-VND run does not generate any new non-dominated solution.

The notion of widening is related to the concept of progressive enlargement of neighborhood implemented in VND, and to the concept of progressive widening [Coulom, 2007] of MCTS, which consists in initially restricting the search of MCTS to the most promising actions only, and accept an increasing number of actions over time.

P-PLS is based on a fundamental assumption and a key observation:

1. P-PLS assumes the existence of *global convexity*. The concept of global convexity, introduced for MOCO in [Borges and Hansen, 2002], is not convexity in the strict sense, but may be used to denote the empirical observation that:

- efficient solutions of a MOCO problem instance are often concentrated in a small part of the decision space;
- solutions neighbors in the objective space tend to be also neighbors in the decision space.

In theory, global convexity and related notions such as connectedness of efficient solutions [Ehrgott and Klamroth, 1997] does not hold in general for a majority of classical MOCO problems [Ehrgott and Klamroth, 1997, Gorski et al., 2006]. By contrast, in practice, meta-heuristics extensively exploit this assumption with great success. For example, local search methods typically explore the search space close to current solutions, therefore local search is naturally favored by a concentration of efficient solutions; MO Estimation of Distribution Algorithms [Pelikan et al., 2006] build a statistical model which detects regularities of solutions of a population and produce new comparable solutions, MO Genetic Algorithms like Decomposition based methods [Zhang and Li, 2007] apply crossover on solutions close in the objective space to produce new similar solutions in the decision space, etc.
Assuming the existence of global convexity tends to legitimate the general idea of P-PLS. Indeed, as the solutions from a sub-archive are neighbors in the objective space, they are globally similar in the decision space by global convexity assumption, therefore the regularities extracted from this population might potentially be strong and thus useful for the efficiency of the search space reduction.

2. We observe in the literature that PLS is generally used with small neighborhood structures. As examples:

   - state-of-the-art methods for MOTSP use 2-opt [Lust and Teghem, 2010, Ke et al., 2014, Cornu et al., 2017];
   - for MO Multidimensional KP, a popular neighborhood move consists in removing a certain percentage of objects from the knapsack, then replace them by other objects. The percentage corresponds to the size of the neighborhood. The current best method for this problem uses a small (as admitted by the authors) neighborhood size of 2% [Ke et al., 2014];
   - concerning MOQAP (see [Drugan and Thierens, 2010, Dubois-Lacoste et al., 2015] as examples), the neighborhood move generally used consists in swapping the position of two facilities;
   - similar observations can be made for the MO set covering problem [Lust and Tuyttens, 2014], and the MO Flow-Shop Scheduling Problem [Dubois-Lacoste et al., 2013].

   As pointed out in [Ke et al., 2014] and [Lust and Tuyttens, 2014] among other studies, large neighborhood size induces strong computational overheads, thus only small neighborhood structures are employed. By contrast, it is well known that increasing the size of a neighborhood drastically increases the quality of results.
   In P-PLS the computational resources saved by the search space reduction can be reallocated to explore neighborhood of larger size.

P-PLS is described in diagram by Figure 4.9 and detailed in pseudo-code by Algorithm 30. P-PLS takes the same input parameters as PLS-VND, with a small modification: we assume that the starting set $\overline{X}$ (to explore) is an archive. As usual, $\overline{X}_{all}$ denotes the global archive. The other input parameters correspond to the set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$ used by PLS-VND and two boolean defining the different possible search strategies of PLS-VND: the first dominating exploration strategy already mentioned (parameter *first-dominating*), and the possibility of avoiding the exploration of the neighborhood of dominated solutions (parameter *explore-dominated*) which is a new feature (initially introduced in our work [Cornu et al., 2017]) we propose for PLS-VND and which will be discussed later. A last input parameter is the maximum authorized initial size σ of a sub-archive. P-PLS returns the global archive $\overline{X}_{all}$.

P-PLS manages a partition LP implemented as a set of parts. A part $P \in LP$ is a tuple composed of:

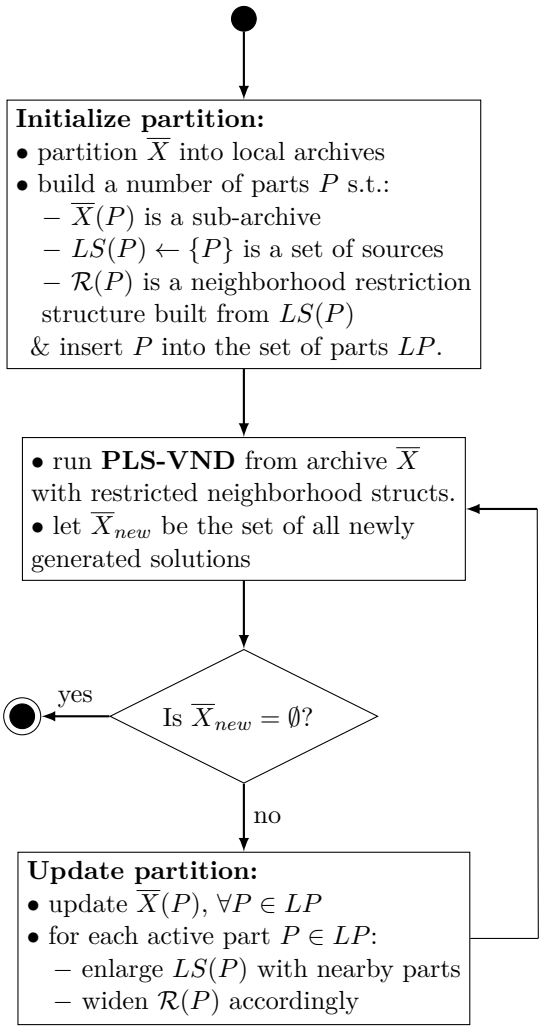


Figure 4.9 – The P-PLS procedure.

Algorithme 30 : `P-PLS`

Input : set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$, boolean *first-dominating*, boolean *explore-dominated*, maximum initial size of part σ, starting archive (to explore) $\overline{X}$, global archive $\overline{X}_{all}$

Output : $\overline{X}_{all}$

$LP \leftarrow$`InitializePartition(`σ, 1, $\overline{X}$`)`
repeat
  $\overline{X}_{all} \leftarrow$`PLS-VND(`$(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating, explore-dominated,* $\overline{X}$, $\overline{X}_{all}$`)`
  $\overline{X}_{new} \leftarrow \{x \in \overline{X} : x$ has been generated at current iteration$\}$
  $LP \leftarrow$ `UpdatePartition(`LP, $\overline{X}_{new}$`)`
until $\overline{X}_{new} = \emptyset$;
return $\overline{X}_{all}$

- a sub-archive $\overline{X}(P)$ which is a subset of $\overline{X}$;
- a set of parts called *sources* $LS(P) \subset LP$; initially, the only source is P itself ($LS(P) = \{P\}$ at first call of PLS-VND);
- a neighborhood restriction structure $\mathcal{R}(P)$ built from the solutions of the sub-archives of the *sources* $LS(P)$.

such that $\overline{X} = \bigcup_{P \in LP} \overline{X}(P)$. Each solution $x \in \overline{X}$ belongs to a unique sub-archive, and to any x is attached an attribute $P(x)$ indicating the part to which x is associated. This will allow to apply to the neighborhood of x the relevant neighborhood restriction structure during PLS-VND.

P-PLS starts by partitioning the archive to explore $\overline{X}$ and creates the parts of the partition by calling the `InitializePartition` procedure (Algorithm 31). At each iteration of the main loop of P-PLS (Algorithm 30), first PLS-VND (Algorithm 32) is conducted from $\overline{X}$, then the `UpdatePartition` procedure (Algorithm 34) is called: it updates sub-archives with the set of new solutions $\overline{X}_{new}$ found at this iteration and widens neighborhood restriction structures. P-PLS stops once $\overline{X}_{new}$ is empty and returns the global archive $\overline{X}_{all}$.

Algorithme 31 : `InitializePartition`

Input : maximum initial size of part σ, index of split objective j, archive $\overline{X}$
Output : set of partition LP

if $|\overline{X}| > \sigma$ **then**
  sort $\overline{X}$ given the split objective j so that $\overline{X} = \{x_1, ..., x_{|\overline{X}|}\}$
  $\overline{X}_1 \leftarrow \{x_1, ..., x_{\lfloor |\overline{X}|/2 \rfloor}\}$
  $\overline{X}_2 \leftarrow \{x_{\lfloor |\overline{X}|/2 \rfloor + 1}, ..., x_{|\overline{X}|}\}$
  $LP_1 \leftarrow$`InitializePartition(` σ, $(j \bmod p) + 1$, $\overline{X}_1$`)`
  $LP_2 \leftarrow$`InitializePartition(` σ, $(j \bmod p) + 1$, $\overline{X}_2$`)`
  return $LP_1 \bigcup LP_2$
else
  `//` $|\overline{X}| \leq \sigma \Rightarrow$ `Creation of a new part:`
  $P \leftarrow$**new** Part
  $\overline{X}(P) \leftarrow \overline{X}$
  $LS(P) \leftarrow \{P\}$
  $\mathcal{R}(P) \leftarrow$ `BuildRestriction(`$\overline{X}$`)`
  foreach $x \in \overline{X}(P)$ **do**
    $P(x) \leftarrow P$
  return P

The `InitializePartition` procedure (Algorithm 31) is a recursive method which shares the same partitioning process as a k-d tree [Bentley, 1975]. Each call consists in splitting the input archive in two subsets of equal size by considering the values of solutions on a single objective. The method cycles over the split objectives. As soon as the size of the input archive is lower than or equal to the maximum authorized size of a part σ (passed in input), then a new part P is created: the sub-archive $\overline{X}(P)$ corresponds to the input archive, the set of sources $LS(P)$ is initialized with P itself, the neighborhood restriction structure $\mathcal{R}(P)$ is then built from $\overline{X}(P)$. Finally, each solution of $\overline{X}(P)$ is associated to P. When the procedure terminates, the whole set of parts LP is returned.

Concerning PLS-VND, we propose a novel option to the original method presented in Section 2.2.3.5. In addition to the first dominating exploration strategy, we add the possibility not to explore the neighborhood

Algorithme 32 : `PLS-VND [generalized]`

Input : set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$, boolean *first-dominating*, boolean *explore-dominated*, set of solutions to explore $\overline{X}$, global archive $\overline{X}_{all}$

Output : $\overline{X}_{all}$

1 $\overline{X}_{all} \leftarrow$`AddAll(` $\overline{X}$, $\overline{X}_{all}$`)`
2 $j \leftarrow 1$
3 **while** $j \leq k$ **do**
4 **while** $\overline{X} \neq \emptyset$ **do**
5 $\overline{X}_{new} \leftarrow$`PLS-iteration(` $\mathcal{N}_j$, *first-dominating*, *explore-dominated*, $\overline{X}$, $\overline{X}_{all}$`)`
6 **foreach** $x \in \overline{X}$ **do** $j(x) = j$;
7 **if** $\overline{X}_{new} \neq \emptyset$ **then**
8 $j \leftarrow 1$
9 $\overline{X} \leftarrow \overline{X}_{new}$
10 $j \leftarrow j + 1$
11 $\overline{X} \leftarrow \{x \in \overline{X}_{all} : j(x) < j\}$
12 return $\overline{X}_{all}$

of dominated solutions. Indeed, as previously indicated, as the PLS-VND uses a temporary archive to store the solutions to explore, thus they are protected from a removal from the global archive $\overline{X}_{all}$.

Algorithme 33 : `PLS-iteration [generalized vanilla]`

Input : neighborhood structure $\mathcal{N}_j$, boolean *first-dominating*, boolean *explore-dominated*, set of solutions to explore $\overline{X}$, global archive $\overline{X}_{all}$

Output : archive of new solutions $\overline{X}_{new}$

1 $\overline{X}_{new} \leftarrow \emptyset$
2 **foreach** $x \in \overline{X}$ **do**
3 **if** *explore-dominated* **or** $x \in \overline{X}_{all}$ **then**
4 **foreach** $x' \in \mathcal{R}(P(x)) \circ \mathcal{N}_j(x)$ **do**
5 **if** $f(x) \nleq f(x')$ **then**
6 **if** `Add(`x', $\overline{X}_{all}$`)` **then**
7 `Add(`x', $\overline{X}_{new}$`)`
8 $P(x') \leftarrow P(x)$
9 **if** *first-dominating* **and** $f(x') \leq f(x)$ **then**
10 **break**
11 return $\overline{X}_{new}$

Algorithms 32 and 33 present the new version of PLS-VND with two simple modifications (highlighted in pseudo-code with red color) compared to the original version, consisting in:

1. The introduction of the input parameter *explore-dominated*, set to true if the exploration of the neighborhood of dominated solutions is permitted. Thus if *explore-dominated* is set to true, then it corresponds to the original version of PLS-VND. From now on, if *explore-dominated* is set to false, then before exploring a solution $x \in \overline{X}$, we check if x still belongs to $\overline{X}_{all}$. If not, then x is dominated by a solution newly accepted in $\overline{X}_{all}$ and thus the exploration is not performed (line 3 of Algorithm

33). Checking the presence of x in $\overline{X}_{all}$ is made in constant time by associating to each solution a flag turned off once another solution dominates it in the global archive.

2. The inclusion of P-PLS features: instead of exploring the whole neighborhood $\mathcal{N}_j(x)$ of a solution x, we first apply to $\mathcal{N}_j(x)$ the neighborhood restriction structure $\mathcal{R}(P(x))$ of the part associated to x (line 4 of Algorithm 33). Once a new solution x' has been accepted in $\overline{X}_{all}$, then we assign as part of x' the part of x (line 8 of Algorithm 33).

Algorithme 34 : `UpdatePartition`

Input : set of parts LP, archive of new solutions $\overline{X}_{new}$
Output : LP

`// Remove dominated solutions from archives of parts:`
1 **foreach** $P \in LP$ **do**
2 $\quad \overline{X}(P) \leftarrow \overline{X}(P) \setminus \{x \in \overline{X}(P) : \exists x' \in \overline{X}_{new},\ f(x') \leq f(x)\}$
`// Add new solutions into appropriate archives of parts:`
3 $LP_a \leftarrow \emptyset$
4 **foreach** $x \in \overline{X}_{new}$ **do**
5 $\quad$ `Add(` x, $\overline{X}(P(x))$`)`
6 $\quad LP_a \leftarrow LP_a + P(x)$
`// Enlarge the set of sources of active parts with nearby parts:`
7 **foreach** $P \in LP_a$ **do**
8 $\quad LS_{ext} \leftarrow LP \setminus LS(P)$
9 $\quad$ sort parts from LS_{ext} in increasing order of their distance with P
10 $\quad$ keep in LS_{ext} only the 2^{p-1} closest partition from P
11 $\quad LS(P) \leftarrow LS(P) + LS_{ext}$
`// Widen accordingly the neighborhood restriction structure of active parts:`
12 **foreach** $P \in LP_a$ **do**
13 $\quad \overline{X} \leftarrow \bigcup_{S \in LS(P)} \overline{X}(S)$
14 $\quad \mathcal{R}(P) \leftarrow$ `BuildRestriction(`$\overline{X}$`)`
15 **return** LP

Algorithm 34 describes the `UpdatePartition` procedure. It takes as input parameters the set of all parts LP and the set of new solutions $\overline{X}_{new}$ found by PLS-VND at current iteration.

First, sub-archives are updated with new solutions, thus dominated solutions from sub-archives are removed, and new solutions from $\overline{X}_{new}$ are added into relevant sub-archives (lines 1-6). A set of parts LP_a collects all *active part*s. A part is said to be active if at least one new solution has been inserted in its sub-archive.

Then, the set of sources of any active part is enlarged with nearby parts (lines 7-11). For a given active part, the idea is to add as sources the parts all around the current set of sources of the part. We define the distance between two parts as the Euclidean distance between the center of their respective sub-archives, where the center of a set of points is the average point of its local ideal and its local nadir. For each active part P, we add as new sources of P the 2^{p-1} closest parts from P.

Finally, the restriction structure of each active part is widened according to its new set of sources (lines 12-14).

Note that the set of active parts computed at each iteration of P-PLS by the `UpdatePartition` procedure is of non-increasing size. Indeed, once a part does not generate any solution at a given iteration, its neighborhood

restriction structure will not be widen, and thus none of its solutions will generate any new solution at the next PLS-VND iterations.

A related comment concerns PLS-VND. As mentioned, a new solution generated by PLS-VND inherits its part from its parent. Naturally, this algorithmic choice can lead to the situation where a new solution is actually closer from another part. This situation appears in Figure 4.8, for some solutions at southeast of the part of interest. In practice, this occurs marginally. Moreover, a first algorithmic choice had been instead to associate to any new solution the part from which the solution is the closer from the center of its sub-archive. This enabled to obtain more compact parts. Unfortunately, this also reactivated non-active parts through the `UpdatePartition` procedure and thus increased considerably the computational time of P-PLS for no great improvement of quality.

4.4 2-Phase Iterated Pareto Local Search with Decomposition

The new meta-heuristic we propose, 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D), combines the concepts of three successful methodologies: 2-Phase PLS (Section 2.2.3.6), Decomposition methods (Section 2.2.2) and IPLS (Section 2.2.3.7); and also implements the new features previously introduced in this section: the suggestions of improvement for SO optimizers, A-MDW and P-PLS. 2PIPLS/D is a 2-phase method.

The **first phase** consists in generating an initial high quality approximation of the efficient set. First, A-MDW is utilized, then P-PLS is conducted from the initial approximation set.

The **second phase** aims at iteratively refining the approximation set. As a Decomposition method, 2PIPLS/D keeps in memory all the weights generated in the first phase, and maintains for each weight $\lambda \in \Lambda_0$ a so-called incumbent $x \in X$, which is the best solution for the weighted-sum problem $\lambda f(\cdot)$ found so far. A so-called *sub-problem* (λ, x) is a pair composed of a weight $\lambda \in \Lambda_0$ and the associated incumbent $x \in X$. As 2PIPLS/D is an IPLS, the second phase alternates PLS-VND run with a global perturbation step generating new starting solutions for next PLS-VND run.

The global perturbation step consists in first choosing a number of sub-problems. For each selected sub-problem, the incumbent is perturbed and the related weighted-sum problem is data perturbed, then optimized. As the set of incumbents corresponds to an approximation of the supported efficient set, it is generally much smaller than the best-so-far approximation set and is composed of solutions present all along the best-so-far approximation set. Therefore, the perturbation mechanism allows us to generate new solutions dispersed all along the best-so-far approximation set.

Any run of PLS-VND (both in first and second phases) uses the neighborhood restriction structures built by P-PLS at first phase. The SO optimizer used by A-MDW and during the global perturbation step naturally implements the suggestions we have proposed: it memorizes all generated (or useful) solutions and it starts/guides its optimization from a starting solution.

In Decomposition algorithms like MOEA/D or MoMad, each sub-problem provides a unique search direction. We claim that focusing optimization on the same directions during the entire duration of the run may neglect other attractive areas of the search space. This is the reason why in 2PIPLS/D, the weighted-sum problems optimized during second phase are data perturbed in order to produce a stochastic change to the sub-problem search direction (as initially done in our work [Cornu et al., 2017]).

2PIPLS/D is described in pseudo-code by Algorithm 35 and with a diagram by Figure 4.10. It takes as input parameters a stopping criterion given by the user (a maximum time or number of iterations), a SO optimizer, a perturbation operator to perturb incumbents, a minimum acceptance rate threshold $\alpha \in [0; 1]$ for A-MDW, a maximum initial size $\sigma > 0$ of part for P-PLS, a set of neighborhood structures for PLS-VND,

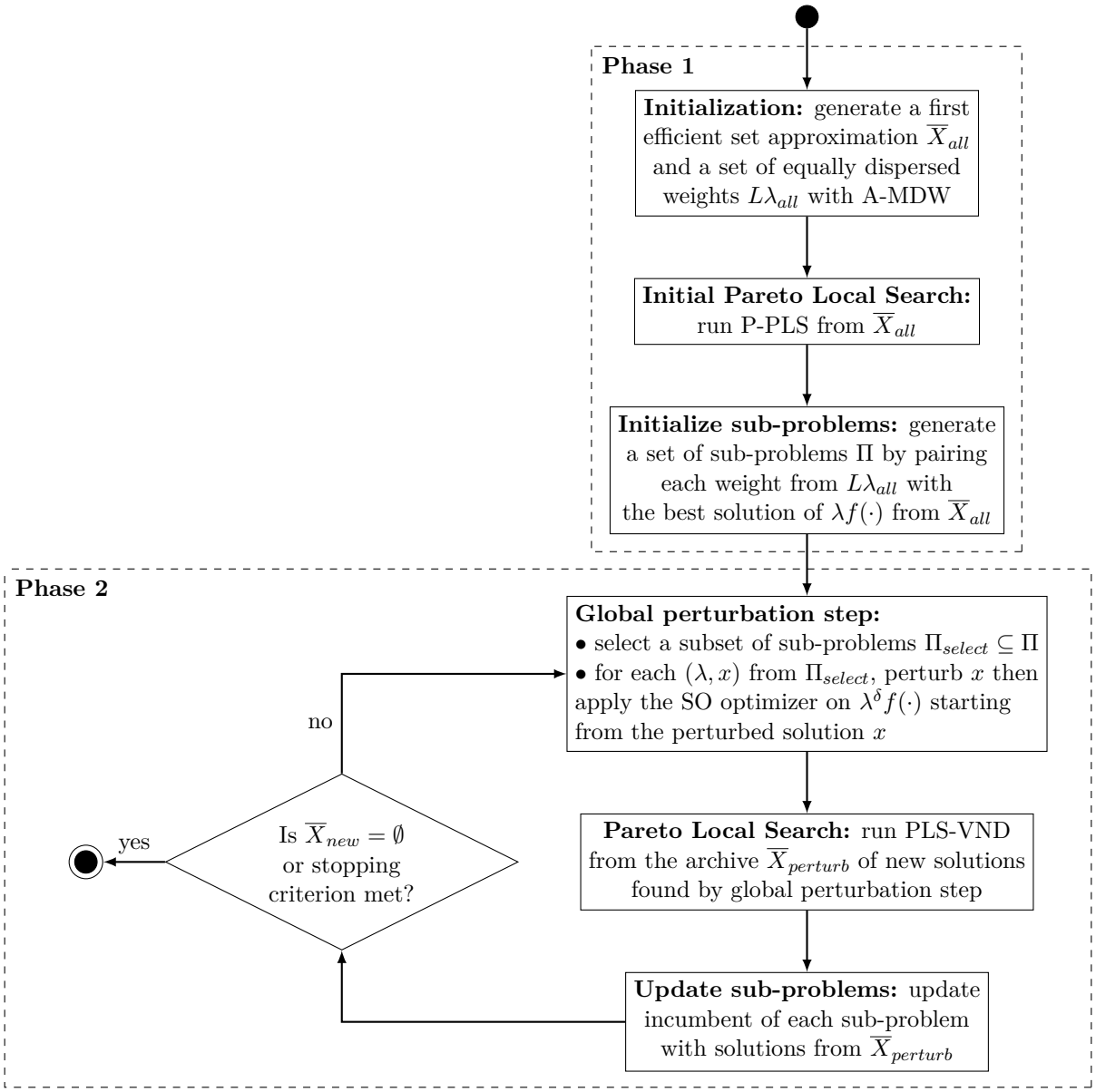


Figure 4.10 – The 2PIPLS/D procedure.

Algorithme 35 : `2PIPLS/D`

Input : *stopping criterion*, SO optimizer `SO-Optimizer`, perturbation operator `Perturb-Operator`, min. acceptance rate threshold α, maximum initial size of part σ, set of neighborhood structures $(\mathcal{N}_1, ..., \mathcal{N}_k)$, boolean *first-dominating*, boolean *explore-dominated*, boolean *independent-pls*, data perturbation coefficient δ

Output : global archive $\overline{X}_{all}$

`// Phase 1:`
`// Initialization:`
$(L\lambda_{all}, \overline{X}_{all}) \leftarrow$`A-MDW(SO-Optimizer,` α, 10^{-3}`)`
`// Initial Pareto Local Search:`
$\overline{X}_{all} \leftarrow$`P-PLS(` $(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating, explore-dominated,* σ, $\overline{X}_{all}$, $\overline{X}_{all}$`)`
$\Pi \leftarrow$`InitializeSubProblems(`$L\lambda_{all}$, $\overline{X}_{all}$`)`
`// Phase 2:`
repeat
  `// Global perturbation step:`
  $\overline{X}_{perturb} \leftarrow \emptyset$
  $\Pi_{select} \leftarrow$`SelectSubProblems(`Π`)`
  foreach $(\lambda, x) \in \Pi_{select}$ **do**
    $x' \leftarrow$`Perturb-Operator(`x`)`
    $\overline{X} \leftarrow$`SO-Optimizer(`$\lambda f^{\delta}(\cdot)$, x'`)`
    $\overline{X}_{perturb} \leftarrow$`AddAll(` $\overline{X}$, $\overline{X}_{perturb}$`)`
  `// Pareto Local Search step:`
  if *independent-pls* **then**
    `// PLS-VND in independent mod:`
    $\overline{X}_{pls} \leftarrow$`PLS-VND(` $(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating, explore-dominated,* $\overline{X}_{perturb}$, $\emptyset$`)`
    $\overline{X}_{all} \leftarrow$`AddAll(` $\overline{X}_{pls}$, $\overline{X}_{all}$`)`
  else
    `// PLS-VND in dependent mod:`
    $\overline{X}_{perturb} \leftarrow \{x \in \overline{X}_{perturb} : \nexists x' \in \overline{X}_{all},\ f(x') \leqq f(x)\}$
    $\overline{X}_{all} \leftarrow$`PLS-VND(` $(\mathcal{N}_1, ..., \mathcal{N}_k)$, *first-dominating, explore-dominated,* $\overline{X}_{perturb}$, $\overline{X}_{all}$`)`
  $\overline{X}_{new} \leftarrow \{x'' \in \overline{X}_{all} : x''$ has been found at current iteration$\}$
  $\Pi \leftarrow$`UpdateSubProblems(`Π, $\overline{X}_{new}$`)`
until $\overline{X}_{new} = \emptyset$ **or** *stopping criterion* is met;
return $\overline{X}_{all}$

Algorithme 36 : `InitializeSubProblems`

Input : set of weights $L\lambda_{all}$, global archive $\overline{X}_{all}$

Output : set of sub-problems Π

$\Pi \leftarrow \emptyset$
foreach $\lambda \in L\lambda_{all}$ **do**
  $x \leftarrow \arg\min\{\lambda f(x') : x' \in \overline{X}_{all}\}$
  $\Pi \leftarrow \Pi + (\lambda, x)$
return Π

three parameters corresponding to the different possible exploration strategies and mods of PLS-VND, and the data perturbation coefficient $\delta \geq 0$.

During the first phase, A-MDW is run from a restricted weight space Λ_ε with $\varepsilon \in [0; \frac{1}{p}[$ (typically $\varepsilon \leq 10^{-3}$) and generates a set of equally dispersed weights $L\lambda_{all}$ and a global archive $\overline{X}_{all}$. Then, P-PLS is conducted from $\overline{X}_{all}$, generating a set of neighborhood restriction structures and enriching the global archive $\overline{X}_{all}$. Finally, the set of sub-problems Π is initialized (Algorithm 36) such that each sub-problem $(\lambda, x) \in \Pi$ is composed of a weight $\lambda \in L\lambda_{all}$ to which is associated an incumbent $x \in \overline{X}_{all}$.

Algorithme 37 : `SelectSubProblems`

Input : set of sub-problems Π
Output : subset of selected sub-problems Π_{select}

```
1 X̄_se ← {x ∈ X : (·, x) ∈ Π}
2 Π_select ← ∅
3 foreach x ∈ X̄_se do
4 |   λ ← select uniform {λ ∈ Λ_0 : (λ, x) ∈ Π}
5 |   Π_select ← Π_select + (λ, x)
6 return Π_select
```

Let us now describe the second phase. At each iteration:

1. Firstly, a global perturbation step is performed:
   • A subset of the sub-problems Π_{select} is selected from Π (Algorithm 37). A-MDW may generate a large number of weights, so that a single solution may be the incumbent of several sub-problems. The aim is to optimize a single data perturbed weighted sum problem for each unique incumbent. Thus, for each unique incumbent $x \in X$, a weight $\lambda \in \Lambda_0$ is selected uniformly at random from the set of sub-problems Π such that (λ, x) is a sub-problem.
   • For each selected sub-problem $(\lambda, x) \in \Pi_{select}$, the incumbent is perturbed providing a new solution $x' \in X$, then the weighted-sum problem is data perturbed with the coefficient $\delta \geq 0$ and the SO optimizer is used to optimize $\lambda f^\delta(\cdot)$ starting from x'. The archive $\overline{X}_{perturb}$ memorizes all incomparable solutions generated by the SO optimizer at current iteration.

2. Secondly, PLS-VND (Algorithm 32 using neighborhood restriction structures) is performed from $\overline{X}_{perturb}$. In addition to the first dominating exploration strategy (*first-dominating* parameter) and the option consisting in exploring or not the neighborhood of dominated solutions (*explore-dominated* parameter), 2PIPLS/D gives the possibility to use PLS-VND in independent mod (*independent-pls* parameter). In independent mod, PLS-VND is run from $\overline{X}_{perturb}$ from scratch in such a way that $\overline{X}_{all}$ is completely ignored and any solution from $\overline{X}_{all}$ can thus be rediscovered during the process. To our knowledge, this mod is used by all known proposed implementations of IPLS [Drugan and Thierens, 2010, Drugan and Thierens, 2012, Inja et al., 2014]. In dependent mod, $\overline{X}_{all}$ is considered, thus any solution from $\overline{X}_{perturb}$ weakly dominated by a solution from $\overline{X}_{all}$ is removed, then PLS-VND is conducted from $\overline{X}_{perturb}$ with $\overline{X}_{all}$ as global archive. To our knowledge, this mod is only used by MoMad [Ke et al., 2014]. At the expense of a much higher computational resources need, PLS-VND in independent mod has a superior exploration power than PLS-VND in dependent mod.
   After PLS-VND, the incumbents of sub-problems are updated with new solutions from $\overline{X}_{all}$ found at current iteration (Algorithm 38).

2PIPLS/D stops as soon as either no new solution has been accepted in the global archive $\overline{X}_{all}$ at the current iteration, or the stopping criterion given by the user is met.

Algorithme 38 : `UpdateSubProblems`

Input : set of sub-problems Π, new archive $\overline{X}_{new}$
Output : Π

```
foreach (λ, x) ∈ Π do
    x ← arg min{λf(x') : x' ∈ X̄_new ⋃ {x}}
return Π
```

Figure 4.11 shows the main steps of a single iteration of 2PIPLS/D, and in particular illustrates the difference between the two PLS-VND mods.

4.5 Preservation of a good distribution of points in the objective space

In addition to the difficulty for finding a good quality approximation of the non-dominated set, meta-heuristics have to provide a set of points well-distributed along the non-dominated set. An issue arises when the addressed problem is intractable, making impossible the memorization of all the non-dominated points for some instances of the problem.

In order to maintain a set of well-distributed points in the objective space and bound the size of this set during the optimization process, Laumanns et al. [Laumanns et al., 2002] have introduced the concept of ϵ-*archive*. They propose to place a hyper-grid that discretizes the objective space into regions called *boxes*. A box can contain at most one point. An ϵ-archive takes as parameter a tolerance $\epsilon > 0$, which controls the dispersion of the points and implicitly determines the maximal size of the approximation. The larger ϵ, the larger the dispersion of the points and the smaller the size of the approximation will be. Guarantees on the good distribution of points in the ϵ-archive and on the bound on its size are given in [Laumanns et al., 2002].

We propose a modified version of the original ϵ-archive. The aim of the proposed ϵ-archive is not to maintain an ϵ approximation of the non-dominated set, but rather to bound the size of the archive while guaranteeing a good distribution of the points of the approximation in the objective space.

Let $\epsilon > 0$ be a fixed tolerance parameter. Let $z = (z_1, ..., z_p) \in Z$ be a point and $\text{box}(z) = (\text{box}_1(z), ..., \text{box}_p(z))$ the vector of the box z belongs to, such that $\text{box}_j(z) = \left\lfloor \frac{\log z_j}{\log(1+\epsilon)} \right\rfloor$ is the coordinate of the box on the j-th objective, for $j = 1, ..., p$.

Let $z \in Z$ be a point. The *local ideal point* $\overline{z}^* \in Z$ related to z is the point sharing the same box as z and dominating all the points in this box, i.e. $\overline{z}^*$ is such that for any $z' \in Z$: if $\text{box}(z') = \text{box}(z)$, then $\overline{z}^* \leqq z'$.

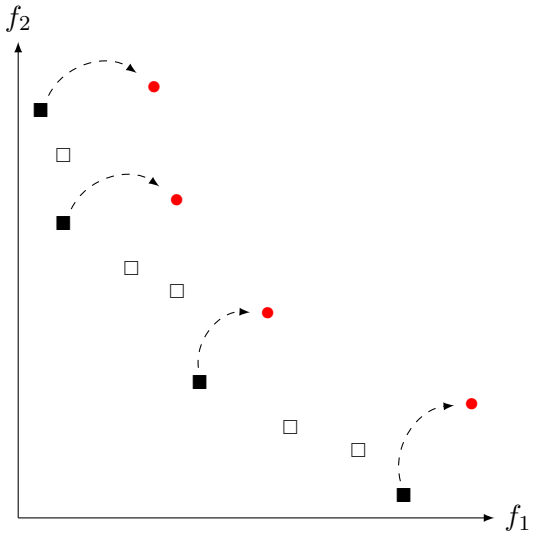
Let $z, \overline{z}^* \in Z$ be two points such that $\overline{z}^*$ is the local ideal point of z. Let

$$\text{D}(z, \overline{z}^*) := \text{lex} \left\{ \max_{j=1,...,p} \{z_j - \overline{z}^*_j\};\ \sum_{j=1}^{p} (z_j - \overline{z}^*_j) \right\}$$

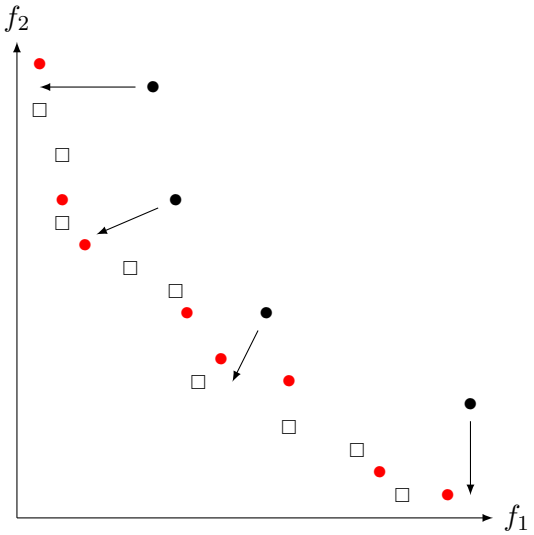
be the distance between z and $\overline{z}^*$.

When an ϵ-archive is used instead of a regular unbounded archive, the following procedure replaces the original `Add` procedure (Algorithm 2). When a solution $x \in X$ s.t. $z = f(x) \in Y$ enters the ϵ-archive $\overline{X}$:

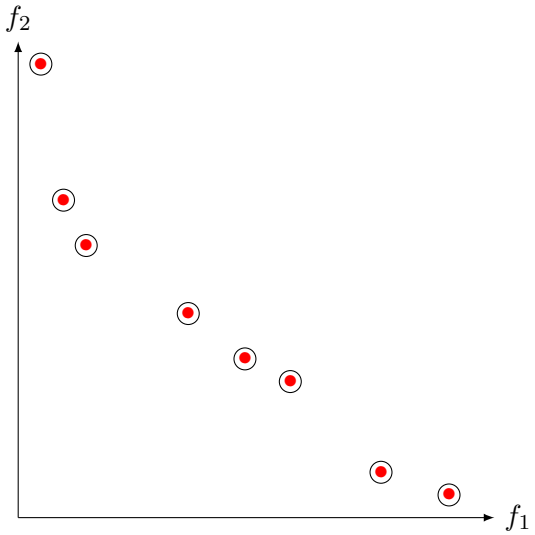
- if there exists $x' \in \overline{X}$ st. $z' = f(x') : \text{box}(z') < \text{box}(z)$, then x is *not* added to $\overline{X}$;
- otherwise, if there exists $x' \in \overline{X}$ st. $z' = f(x') : \text{box}(z) < \text{box}(z')$, then x is added to $\overline{X}$ and x' is removed from $\overline{X}$;



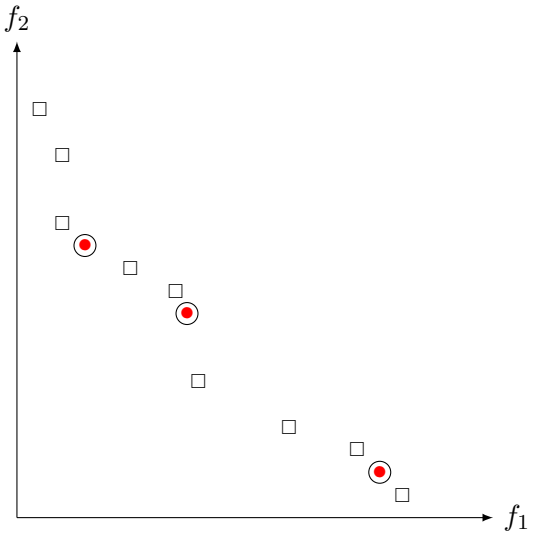
(a) Global perturbation step (1/2): Application of perturbation moves on incumbents of selected sub-pbs.



(b) Global perturbation step (2/2): Optimization of data perturbed weighted sum problems of selected sub-pbs starting from perturbed incumbents.



(c) Pareto Local Search step (first option): Set of starting solutions of PLS-VND run in **independent mod**.



(d) Pareto Local Search step (second option): Set of starting solutions of PLS-VND run in **dependent mod**.

Figure 4.11 – Main steps of a 2PIPLS/D iteration. Newly generated solutions are represented by bullets while the other solutions are represented by squares. Starting solutions of PLS-VND are circled.

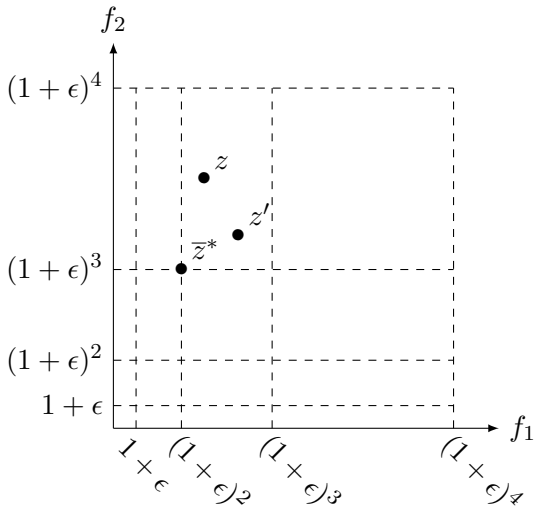


Figure 4.12 – Hyper-grid of an ϵ-archive represented in the objective space. $z, z' \in Y$ share the same box.

- otherwise, if there exists $x' \in \overline{X}$ st. $z' = f(x') : \text{box}(z) = \text{box}(z')$, then:
  - If $\text{D}(z, \overline{z}^*) < \text{D}(z', \overline{z}^*)$ where $\overline{z}^* \in Z$ is the local ideal point related to z, then x is added to $\overline{X}$ and x' is removed from $\overline{X}$
  - Otherwise, x is *not* added to $\overline{X}$
- otherwise, z is added to $\overline{X}$.

Figure 4.12 illustrates an hyper-grid of an ϵ-archive in a bi-objective space.

4.6 Conclusion

We proposed in this chapter a number of improvements of existing methods and new optimization methods to tackle more efficiently MOCO problems.

First we suggested three modifications of SO solvers to improve their ability to find efficient solutions in MOCO problems: systematically starting/guiding the search from an already found solution, memorizing the solutions generated during the optimization process and enable the data perturbation of a weighted sum problem. Based on these specifications, we proposed a new version of NMCS, called Aggregation-based Nested Monte Carlo Search (A-NMCS) better suited for MOCO than vanilla NMCS.

Second, we presented a new method generalizing the concept of MDW called Adaptive MDW (A-MDW) and making it more adaptive such that the method continues generating weights until the SO solver optimizing the related weighted sum problems is no longer efficient.

Third, we introduced Partitioned Pareto Local Search (P-PLS), which aims at speeding-up PLS-VND through the partitioning of the set of solutions to explore, and a smart restriction of the PLS-VND neighborhood structure based on the presumption of global convexity on the addressed problem.

Then, we proposed a new MO meta-heuristic called 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D) based on three successful methodologies: 2-Phase meta-heuristics, Decomposition methods and IPLS, and combining A-MDW, P-PLS, as well as the suggested modifications for SO solvers.

Finally, a new system based on the ϵ-archive concept is presented, allowing to bound the size of an archive while guaranteeing a good distribution of the points in the objective space.

Chapter 5

Application of 2PIPLS/D to MOTSP

In this chapter, we apply 2PIPLS/D to MOTSP. We first detail the implementation of the different algorithmic components of 2PIPLS/D, then propose an empirical evidence of global convexity on MOTSP in order to legitimate the use of the P-PLS partitioning. After a sensitivity analysis of 2PIPLS/D on its parameters, we propose a final parameter setting. Finally, we compare the method with the best known algorithms on a large benchmark of bi-objective and tri-objective instances.

Introduction

This chapter proposes the application of 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D) to MOTSP. Section 5.1 first presents the benchmarks of bi-objective and tri-objective instances used for the experiments, then Section 5.2 describes how 2PIPLS/D has been implemented for MOTSP, in particular it details the SO optimizer(s) and the perturbation move selected, the PLS-VND neighborhood structure utilized and the neighborhood restriction structure used in P-PLS. Then Section 5.3 proposes an empirical evidence of existence of global convexity on MOTSP, in order to justify the use of P-PLS (as it is suggested in Section 4.3). After a sensitivity analysis of 2PIPLS/D on its parameters, a good final parameter setting is proposed in Section 5.4. A comparison of 2PIPLS/D with state-of-the-art methods on MOTSP is made in Section 5.5. Finally, we summarize the chapter and propose a number of perspectives concerning MOTSP.

5.1 MOTSP benchmarks and experimental design

5.1.1 Types of instances

The MOTSP instances used in our experiments either come directly from the literature or have been generated with exactly the same process as in the literature. They are of different types:

- **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.
  Most of Euclidean instances comes from the TSPLIB[1] library [Reinelt, 1991] or have been generated with the same process as in TSPLIB. More precisely, on each objective, the coordinates of each city are integers that are uniformly and independently generated in the range [0,3163]. These instances are denoted as Krolak/Felts/Nelson (abbreviated as "kro") instances. Other instances have been produced by the DIMACS instance generator[2].

- **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 distances.
  All clustered instances have been produced with the DIMACS instance generator, such that any instance of size strictly lower than 100 is composed of 12 clusters, and any instance of size greater than or equal to 100 is composed of 25 clusters.

- **Random instances**: the costs between the edges are randomly generated from a uniform distribution. All random instances have been produced with the DIMACS instance generator, such that 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]. This range was chosen in order to have a range similar to the one of the *kro* instances (note that $\left\lfloor \sqrt{2 \times 3163^2} + 0.5 \right\rfloor = 4473$).

- **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.

[1] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
[2] http://dimacs.rutgers.edu/Challenges/TSP/

5.1.2 Benchmarks of MOTSP instances

Two different benchmarks are utilized. The *learning benchmark* is first used to make an empirical evidence of the global convexity on MOTSP (Section 5.3), then to propose a good parameter setting for 2PIPLS/D (Section 5.4). The *test benchmark* is employed for the comparison of 2PIPLS/D with its competitors (Section 5.5).

Learning benchmark

The learning benchmark is composed of 80 bi-objective and tri-objective instances of different sizes and types:

- 10 bi-objective Euclidean instances of size 100, and 10 of size 500;
- 10 bi-objective random instances of size 100, and 10 of size 500;
- 10 tri-objective Euclidean instances of size 100, and 10 of size 500;
- 10 tri-objective random instances of size 100, and 10 of size 500.

Euclidean instances have been generated with the same process as *kro* instances mentioned above, and random instances have been produced with the DIMACS instance generator.

Test benchmark

The benchmark used for the comparison between 2PIPLS/D and its competitors on MOTSP is described in Table 5.1 and is composed of 55 instances: 30 bi-objective instances with a size ranging from 100 to 1000, and 25 tri-objective instances with a size ranging from 50 to 500 such that:

- 40 instances come from the literature [Jaszkiewicz, 2002, Paquete and Stützle, 2003, Paquete et al., 2004, Angel et al., 2004, Lust and Teghem, 2010, Lust and Jaszkiewicz, 2010] (30 bi-objective, 10 tri-objective). The Euclidean *kro* instances come either from the TSPLIB library or have been generated by Lust and Teghem[3] [Lust and Jaszkiewicz, 2010]. The rest of instances has been produced with the DIMACS instance generator.
- 15 additional tri-objective instances we have generated with the same process as *kro* instances or with the DIMACS instance generator.

5.1.3 Experimental design

For all bi-objective instances and small-size ($n < 100$) tri-objective instances, the sets of points of all tested methods (i.e. 2PIPLS/D and its competitors) are managed as regular unbounded archives (i.e. with a tolerance for dominance relations $\epsilon = 0\%$).

For tri-objective instances with $n \geq 100$, the size of the non-dominated sets is very large, in consequence the archives of all tested methods are managed as ϵ-archives with the technique preserving a good distribution of the points described in Section 4.5, with a tolerance $\epsilon = 1\%$. Note that the computational cost of the log function is negligible because the required log values are precomputed and stored in a look-up table before the beginning of each instance optimization.

All experiments were performed on a 3.4 GHz computer with 16Gb of volatile memory (RAM) on a Linux OS.

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

Type	Quantity	Size (n)	Source	Composition	Name
Euclidean	17	100	TSPLIB library	2 Euclidean SO instances	kroAB100, kroAC100, kroAD100, kroBC100, kroBD100, kroCD100
			DIMACS generator	2 Euclidean SO instances	euclidAB100, euclidCD100, euclidEF100
		200	TSPLIB library	2 Euclidean SO instances	kroAB200
		300	DIMACS generator	2 Euclidean SO instances	euclidAB300
			[Lust and Jaszkiewicz, 2010]	2 Euclidean SO instances	kroAB300
		400	[Lust and Jaszkiewicz, 2010]	2 Euclidean SO instances	kroAB400
		500	DIMACS generator	2 Euclidean SO instances	euclidAB500
			[Lust and Jaszkiewicz, 2010]	2 Euclidean SO instances	kroAB500
		750	[Lust and Jaszkiewicz, 2010]	2 Euclidean SO instances	kroAB750
		1000	[Lust and Jaszkiewicz, 2010]	2 Euclidean SO instances	kroAB1000
clustered	3	100	DIMACS generator	2 clustered SO instances	clusteredAB100
		300	DIMACS generator	2 clustered SO instances	clusteredAB300
		500	DIMACS generator	2 clustered SO instances	clusteredAB500
random	5	100	DIMACS generator	2 random SO instances	rdAB100, rdCD100, rdEF100
		300	DIMACS generator	2 random SO instances	rdAB300
		500	DIMACS generator	2 random SO instances	rdAB500
mixed	5	100	DIMACS generator	1 random + 1 Euclidean instances	mixedGG100, mixedHH100, mixedII100
		300	DIMACS generator	1 random + 1 Euclidean instances	mixedAB300
		500	DIMACS generator	1 random + 1 Euclidean instances	mixedAB500

(a) Bi-objective instances ($p = 2$).

Type	Quantity	Size (n)	Source	Composition	Name
Euclidean	12	50	Personal generator (*new*)	3 new Euclidean SO instances	euclidA-3-50, euclidB-3-50, euclidC-3-50
		100	TSPLIB library	3 Euclidean SO instances	kroABC100
			DIMACS generator	3 Euclidean SO instances	euclidABC100, euclidDEF100
		300	DIMACS generator	3 Euclidean SO instances	euclidABC300, euclidDEF300
			Personal generator (*new*)	2 Euclidean + 1 new Euclidean SO instances	euclidG-3-300
		500	DIMACS generator	3 Euclidean SO instances	euclidABC500, euclidDEF500
			Personal generator (*new*)	3 new Euclidean SO instances	euclidG-3-500
clustered	4	50	DIMACS generator (*new*)	3 new clustered SO instances	clusteredA-3-50
		100	DIMACS generator (*new*)	clusteredAB100 + 1 new clustered SO instances	clusteredABC100
		300	DIMACS generator (*new*)	clusteredAB300 1 new clustered SO instances	clusteredABC300
		500	DIMACS generator (*new*)	clusteredAB500 1 new clustered SO instances	clusteredABC500
random	9	50	DIMACS generator (*new*)	3 new random SO instances	rdA-3-50, rdB-3-50, rdC-3-50
		100	DIMACS generator	3 random SO instances	rdABC100
			DIMACS generator (*new*)	3 new random SO instances	rdD-3-100
		300	DIMACS generator	3 random SO instances	rdABC300
			DIMACS generator (*new*)	3 new random SO instances	rdD-3-300
		500	DIMACS generator	3 random SO instances	rdABC500
			DIMACS generator (*new*)	3 new random SO instances	rdD-3-500

(b) Tri-objective instances ($p = 3$).

Table 5.1 – MOTSP *test* benchmark.

5.2 Implementation of 2PIPLS/D to address MOTSP

2PIPLS/D is written in C/C++. Different algorithmic components of 2PIPLS/D have to be chosen and implemented for the addressed problem: the SO optimizer, the perturbation move, the neighborhood structure used by PLS-VND and the neighborhood restriction structure utilized in P-PLS.

5.2.1 SO optimizer

Concerning the SO optimizers, we compare two different methods: the local search-based method Chained Lin-Kernighan [Applegate et al., 2003] (C-LK), and the Monte Carlo Search-based method NMCS (Section 2.2.5.3) and A-NMCS (Section 4.1.1), its version especially designed for MOCO. C-LK is the solver used in the last three best methods on bi-objective TSP (2PPLS [Lust and Teghem, 2010], MoMad [Ke et al., 2014] and PDA [Cornu et al., 2017]). As in the literature, we use the C-LK implementation of the Concorde package[4]. As suggested in Section 4.1, we have implemented a modified version of C-LK so that it memorizes all incomparable locally minimum solutions generated. We call this version *improved C-LK*, while the non-modified version is called *vanilla C-LK*.
On the other hand, NMCS has never been adapted to this problem. We chose NMCS because it has been successfully applied to a number of difficult SO problems (as mentioned in Section 2.2.5.3). Consequently, it appears to us interesting to analyze the performance of 2PIPLS/D when using such a promising method which is NMCS (and A-NMCS) with a problem-dependent and efficient ILS method such as C-LK. The idea is also to gauge the dependence of 2PIPLS/D on the SO optimizer used.

Note that we have also tested two other SO optimizers: the Helsgaun implementation of Lin-Kernighan, called LKH [Helsgaun, 2000]; and the Genetic Algorithm using Edge Assembly Crossover (GA-EAX) [Nagata and Kobayashi, 2013]. Internal tests have shown that C-LK and LKH provide globally similar performance, as already observed in [Lust and Teghem, 2010] while GA-EAX provided surprisingly poor results. For these reasons, we do not consider these two solvers in the rest of this document.

Implementation of (A-)NMCS for MOTSP

In our implementation of (A-)NMCS, the construction of a sequence consists in building a tour by iteratively adding an edge to a path. More precisely, a starting city is first selected at initialization of (A-)NMCS, such that all generated sequences will share this city as starting city. By ease of simplicity, we select as starting city the first city of the addressed instance. A partial sequence represents a path, and technically it corresponds to a pair composed of:

- a current city, initialized with the starting city;
- and an ordered list of actions, where an action consists first in the insertion of a *valid edge* present in the candidate edge list of the current city, then the change of current city to the newly introduced city. A valid edge consists in an edge whose insertion does not forbid the partial sequence to finally be a valid tour.

The concept of candidate edge list used in (A-)NMCS is exactly the same as in local search, so as it consists in a fixed-size set of elite edges computed for each city. Let $c^{\lambda,\delta}(e)$ (to be minimized) be the aggregated and perturbed cost function of an edge $e \in E$, where $\lambda \in \Lambda_0$ is a weight and $\delta \geq 0$ is a data perturbation coefficient (see Section 4.1) and such that $\lambda f^{\delta}(\cdot)$ is the fitness minimized by (A-)NMCS. Let $\text{cel}(\chi)$ be the

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

candidate edge list associated with the city χ, such that $\text{cel}(\chi)$ contains the s best edges incident to χ given $c^{\lambda,\delta}(e)$. Let the *normalized cost* $\widehat{c}^{\lambda,\delta}(e,\chi) \in [0,1]$ of an edge $e \in E$ from χ be defined as:

$$\widehat{c}^{\lambda,\delta}(e,\chi) = \frac{c^{\lambda,\delta}(e) - c^{\lambda,\delta}_{min}(\chi)}{c^{\lambda,\delta}_{max}(\chi) - c^{\lambda,\delta}_{min}(\chi)}$$

where e is incident to a city χ, and $c^{\lambda,\delta}_{min}(\chi)$ (resp. $c^{\lambda,\delta}_{max}(\chi)$) is the minimum (resp. maximum) cost among all edges incident to χ, defined as:

$$\begin{cases} c^{\lambda,\delta}_{min}(\chi) = \min\{c^{\lambda,\delta}(e') \ : \ e' \in \text{cel}(\chi)\} \\ c^{\lambda,\delta}_{max}(\chi) = \max\{c^{\lambda,\delta}(e') \ : \ e' \in \text{cel}(\chi)\} \end{cases}$$

During a random simulation (level of recursion equal to 0), an edge $e \in \text{cel}(\chi)$ incident to the current city χ of a partial sequence *seq* is randomly selected with the following Boltzmann distribution formula:

$$\mathbb{P}(e \mid seq) = \frac{e^{-\beta \times \widehat{c}^{\lambda,\delta}(e,\chi)}}{\sum\limits_{e' \in \text{cel}(\chi) \setminus \overline{\chi}(seq)} e^{-\beta \times \widehat{c}^{\lambda,\delta}(e',\chi)}}$$

where $\overline{\chi}(seq)$ is the set of cities present in the partial sequence *seq* (i.e. already selected in previous actions and thus invalid for selection) and $\beta \geq 0$ is a parameter to be fixed. This formula encourages the selection of low-cost edges in a random way, such that the higher β is, the higher the probability of the smallest-cost edges to be selected. If $\beta = 0$, then the selection consists in a uniform random selection. The normalization of the cost of the edges allows for setting the same β value for all cities.

At a given level of recursion greater than or equal to 1, (A-)NMCS tries all available actions among these allowed by the candidate edge list of the current city.

At a given step of selection of an action (at any level of recursion), if no edge from the candidate edge list of the current city is valid, then the edge with the smallest cost is selected.

This implementation of (A-)NMCS has 3 parameters to be fixed: the usual level of recursion *level*, the size of the candidate edge list s and the Boltzmann parameter β. We have tested different configurations of values on the *learning benchmark* presented above, and we obtain best results with the following configuration: $level = 1$, $s = 10$, $\beta = 20$. Note that we have chosen a small level of recursion because higher levels of (A-)NMCS provided far too large execution times compared to C-LK runs. Indeed, recent versions of Lin Kernighan have a complexity of $O(n^{\kappa})$, where $\kappa \gtrsim 2$ [Helsgaun, 2000], while (A-)NMCS has a complexity of $O(n^{1+level})$.

5.2.2 Perturbation move

The perturbation move used at perturbation step (second phase of 2PIPLS/D) selected is the double bridge move (see Section 2.3.1) as it is simple, efficient and the most used in the literature. We use the implementation of Paquete[5].

5.2.3 Neighborhood structures of PLS-VND

The neighborhood structures used by PLS-VND are usual k-exchange neighborhoods. We consider as maximum neighborhood structure size $k \in \{2,3\}$ for PLS-VND. Obviously, PLS-VND with $k = 2$ corresponds to PLS.

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

5.2.4 Implementation of the neighborhood restriction structure of P-PLS

The neighborhood restriction structure employed in each part of P-PLS uses the combination of two preexisting techniques in a dynamic way: the *candidate edge list* (presented in Section 2.3.1) and the *locked edges* (Section 2.3.2). For a given part:

- the candidate edge lists contain only the edges present in *at least one* solution in the archives of the sources of this part (they can be considered as the *union* of edges of solutions of the sources).
- the *locked* edges is the set of edges present in *all* solutions in the archives of the sources of this part (they can be considered as the *intersection* of edges of solutions of the sources).

As indicated in Section 4.3, the neighborhood restriction function of an active part is built after the enlargement of the set of sources (Algorithm 34). The construction of the neighborhood restriction structure consists in building the candidate edge lists and identifying the locked edges. During PLS-VND, a move from a given solution will consist in exchanging a number of non-locked edges of the solution only for edges contained in a candidate edge list (with the exception of the close-up edge).

When P-PLS is running, the neighborhood restriction function of each active part is re-built at each iteration. As the size of the archives of the parts tends to increase when the number of iterations grows, the size of the candidate edge lists of an active part also tends to increase, while the set of locked edges of an active part tends to decrease. Note that both techniques have already been employed: the candidate edge list used in 2PPLS is used in a static way, as it consists in the edges present in at least one solution of the initial set found in the first phase, then used in PLS during the second phase. In MoMad, the candidate edge list is used in a more dynamic way as it consists in the edges present in at least one solution of the global archive, in such a way that the candidate edge list tends to increase at each new call of PLS. The locked edges technique has been introduced in MOCO in [Jaszkiewicz, 1999] and used by [Jaszkiewicz and Zielniewicz, 2006], such that locked edges are used in a static way in a path relinking procedure. The originality of our neighborhood restriction structure lies in the combination of these two techniques, the utilization of locked edges in a dynamic way, and obviously the embedding of such a structure in the partitioning system of P-PLS.

Technically, a matrix of all edges is memorized in each part, therefore the partitioning of P-PLS is volatile memory consuming, as the RAM needed is $O(\frac{|\overline{X}_{all}|}{\sigma} \times \frac{n(n-1)}{2})$, where $|\overline{X}_{all}|$ is the size of the global archive $\overline{X}_{all}$ just before the P-PLS partitioning, σ is the user-set parameter equal to the maximum authorized initial size of a part, and $\frac{n(n-1)}{2}$ is the number of edges of the addressed MOTSP instance. The smaller σ, the higher RAM needed by P-PLS.

5.3 Empirical evidence of global convexity on MOTSP

The aim of this section is to empirically verify the existence of global convexity on MOTSP. To do so, we first check if efficient solutions in the decision space are concentrated in a small fraction of the decision space. Secondly, we check if efficient solutions which are neighbors in the objective space tend to be also neighbors in the decision space.

We check these assumptions on the *learning benchmark* previously introduced. To find the efficient set of the instances of this benchmark, we run the exact method AUGMECON2 [Florios and Mavrotas, 2014] (introduced in Section 2.3.2) on the bi-objective instances of size 100. As suggested by the authors and in Section 2.3.2, AUGMECON2 is unable to find the efficient set of instances of larger size or larger number of objectives, thus we run 2PIPLS/D (using C-LK as SO optimizer) on each instance during several hours with different seeds in order to obtain an approximation of the efficient set.

		Proportion of efficient edges	Order of magnitude of non-dominated set size
$p = 2$	$n = 100$	10.5%	10^3
	$n = 500$	2.5%	10^4
$p = 3$	$n = 100$	22%	10^5
	$n = 500$	7.5%	10^6

Table 5.2 – Average proportion of edges used in (potentially) efficient solutions and order of magnitude of size of non-dominated sets in function of the number of objectives and the size of instances.

Let the notion of *(potentially) efficient solutions* encompassing either efficient solutions or approximate efficient solutions.

5.3.1 Concentration of (potentially) efficient solutions in the decision space

To show how much (potentially) efficient solutions are concentrated in the decision space, we are interested in the proportion of edges (which are the basic elements composing solutions of MOTSP) used in (potentially) efficient solutions. Let an *efficient edge* be an edge present in at least one (potentially) efficient solution of an instance.

The first column of Table 5.2 shows that the (potentially) efficient solutions contain a small fraction of the edges of an instance: from 2.5% (in average) for bi-objective instances of size 500 to 22% (in average) for tri-objective instances of size 100. When considering all instances of the test benchmark, only 11% (in average) of the edges are efficient.

Tri-objective instances have a proportion of efficient edges twice to triple larger than bi-objective instances, which is directly explained by the increase of the size of the non-dominated sets from the bi-objective to the tri-objective case, as indicated by the second column of Table 5.2.

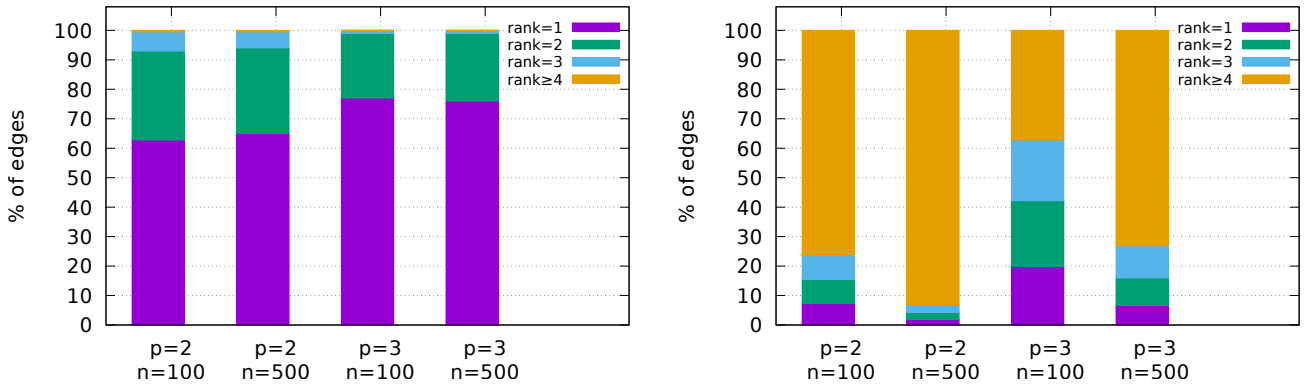
The efficient edges are of very good quality. Let $rank(e \mid \chi)$ be the *rank of an edge* $e \in E$ *given the city* χ, which is the Non-Dominated Sorting rank (cf. Section 2.4) of the edge e incident to χ, computed with respect to all edges incident to χ and such that the edges are compared through the MO cost function $c : E \longrightarrow \mathbb{R}^p$. Let

$$rank(e) = \min\{rank(e \mid \chi_1), rank(e \mid \chi_2)\}$$

be the rank of the edge $e \in E$ incident to both cities χ_1 and χ_2. Therefore, the rank of an edge $e \in E$ is its best rank given its two incident cities, and measures the quality of e. The lower the rank of an edge is, of better quality the edge.

From this formulation of the quality of an edge, we have made an interesting observation concerning the distribution of ranks of efficient edges, illustrated in Figure 5.1. Indeed, the rank of efficient edges is low, and in average on all instances of the benchmark, **96% of efficient edges are of rank lower than or equal to 2, while only 19% of all edges are of rank lower than or equal to 2**.

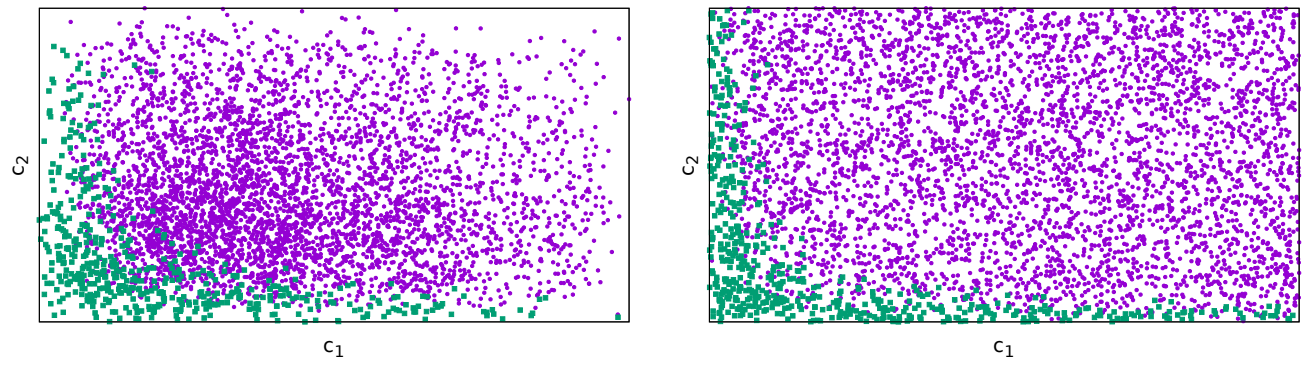
This distribution of ranks of edges leads to a rather neat distinction between efficient edges and the other edges visible in the MO cost space, as illustrated in Figure 5.2. Similar snapshots are obtained in the



(a) Distribution of ranks of *efficient edges*.

(b) Distribution of ranks of *all edges*.

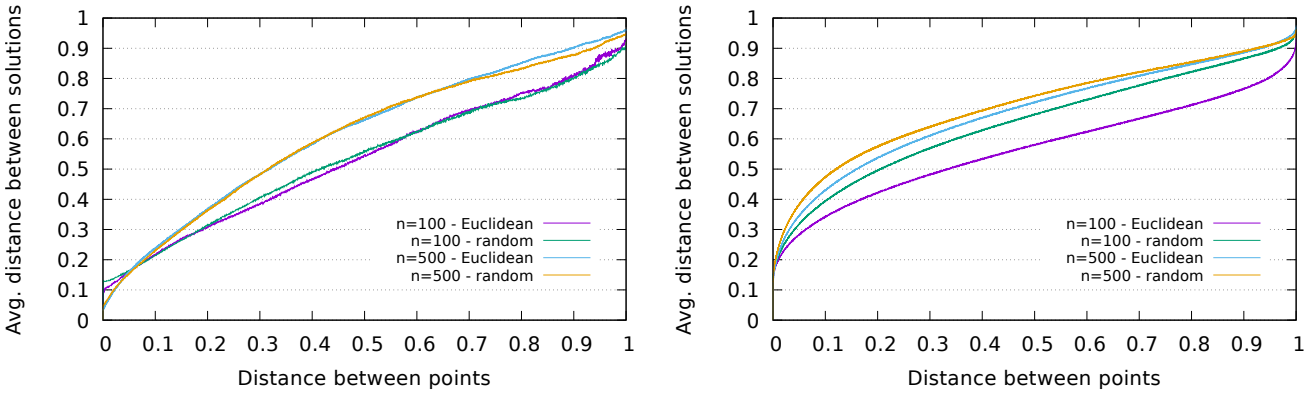
Figure 5.1 – Distribution of ranks of efficient edges vs. all edges, in function of the number of objectives and the size of instances.



(a) Euclidean instance.

(b) Random instance.

Figure 5.2 – Representation in the MO cost space of the edges composing (potentially) efficient solutions (represented by squares) and the other edges (represented by bullets) in bi-objective instances of size 100.



(a) $p = 2$ (b) $p = 3$

Figure 5.3 – Average distance between (potentially) efficient solutions in function of the distances between their images (points) in the objective space.

tri-objective case.

5.3.2 Are (potentially) efficient solutions neighbors in the objective space also neighbors in the decision space?

To check if (potentially) efficient solutions which are neighbors in the objective space tend also to be neighbors in the decision space, we computed the Spearman correlation between:

- the distances of (potentially) efficient solutions (which are tours of MOTSP) in the decision space
- and the distances of their images in the objective space.

The Spearman correlation assesses how well the relationship between two series of data can be described using a monotonic function. The Spearman correlation coefficient can take values between -1 and 1, such that a value of -1 indicates that data series are completely negatively correlated, while +1 indicates that data series are completely positively correlated.

Let $\widetilde{X}_e$ be the (potentially) efficient set of a given instance. For each (potentially) efficient solution $x \in \widetilde{X}_e$, let $f(x^k)$ s.t. $x^k \in \widetilde{X}_e$, is the k-th nearest neighbor of $f(x)$ given the Euclidean distance; for each $x^k \in \widetilde{X}_e$, we computed the Hamming distance between x and x^k. After averaging and normalizing these distances over all (potentially) efficient solutions and instances of each type, we obtain Figure 5.3. The Spearman correlation computed for each curve is strictly greater than 0.9999. Moreover, given a significance level of 1%, the p-value related to each Spearman correlation scores is lower than 10^{-5}, meaning that the Spearman correlation scores are statistically significant. This means that **the distances of solutions and the distances of their images in the objective space are almost perfectly positively correlated**.

From these empirical observations, we can conclude that the assumption of global convexity on MOTSP we have made is legitimate, which corroborate the observation made in [Borges and Hansen, 2002].

5.4 Parameter setting of 2PIPLS/D

2PIPLS/D has several parameters to set which are listed in Table 5.3. A preselection of alternative values and a selection of the default value for each parameter have been made on the basis of internal tests.

1. As SO optimizer (used by A-MDW and during each perturbation phase of second phase) we propose vanilla C-LK, improved C-LK, (vanilla) NMCS and A-NMCS. We select both the improved C-LK and A-NMCS as default SO optimizers.

2. A-MDW (cf. Section 4.2) is controlled by the minimum acceptance rate threshold α. A-MDW stops as soon as the efficiency of the SO optimizer (i.e. the sum of solutions accepted in the global archive divided by the sum of incomparable solutions generated by each run of the SO optimizer) goes below α. The lowest α is, more consuming A-MDW will be in terms of computational resources. The preselected range of α goes from 5% to 95% and the default value is 25%.

3. Data perturbation (cf. Section 4.1) is controlled by the coefficient δ. The highest δ is, a more important perturbation will be applied to TSP cost matrices. The preselected range of δ goes from 0% (no data perturbation) to 20% (in general, bad results are obtain with higher values). We select a pretty low default value of data perturbation: $\delta = 5\%$.

4. The maximum initial size σ of a part of P-PLS (cf. Section 4.3) corresponds to the maximum authorized initial size of a part at initialization of P-PLS. The preselected range of σ goes from 50 solutions to no partitioning ($\sigma = +\infty$). The lower bound of the range is limited by the RAM of our machine. The default selected value is $\sigma = 100$.

5. The pre-selected maximum neighborhood structure sizes for PLS-VND are $k = 2$ and $k = 3$, and both are selected by default.

6. Concerning PLS-VND (cf. Algorithm 32 of Section 4.3), we propose to activate by default all strategies: the *independent-pls* strategy, meaning that at each iteration of the second phase, PLS-VND is started with an empty global archive; the *first-dominating* strategy, forcing PLS-VND to quit the exploration of a solution as soon as a neighbor dominates it; and the *explore-dominated* strategy which enables PLS-VND to explore a solution even if it is dominated by a solution in the global archive.

Parameter	Module of 2PIPLS/D concerned	Selected range of values	Default values
SO optimizer	-	{vanilla C-LK, improved C-LK, NMCS, A-NMCS}	{improved C-LK, A-NMCS}
perturbation move	perturbation step (phase 2)	-	Double bridge move
acceptance rate threshold α	A-MDW	$\{5\%, 25\%, 50\%, 75\%, 95\%\}$	25%
data perturbation coefficient δ	SO optimizer (phase 2)	$\{0\%, 5\%, 10\%, 20\%\}$	5%
maximum initial size σ of part	P-PLS	$\{50, 100, 200, 400, +\infty\}$	100
maximum neighborhood structure size k	PLS-VND	$\{2, 3\}$	$\{2, 3\}$
independent-pls	PLS-VND	{no, yes}	yes
first-dominating	PLS-VND	{no, yes}	yes
explore-dominated	PLS-VND	{no, yes}	yes
tolerance for dominance relations ϵ	-	-	$\begin{cases} 0\% & \text{if } p = 2 \text{ or } n < 100 \\ 1\% & \text{otherwise} \end{cases}$
archive	-	-	$\begin{cases} \text{SAAVLA} & \text{if } p = 2 \\ \text{SANDRA} & \text{otherwise} \end{cases}$

Table 5.3 – Parameters of 2PIPLS/D, their respective selected range of values and default values.

We perform a sensitivity analysis of 2PIPLS/D on its parameters. In particular, we aim at comparing the relative importance of the different algorithmic components of 2PIPLS/D and finally propose a good parameter setting. Given the large number of parameters, we have tested each parameter separately from the other parameters, except for the *independent-pls* and *first-dominating* exploration strategies, which have been combined because they are strongly related.

We use the *learning benchmark* (Section 5.1.2) to perform this sensitivity analysis. On each instance of the benchmark, 2PIPLS/D is run given each alternative value with each possible default combination of values. For example, when the acceptance rate threshold α of A-MDW has been tested, we have run 2PIPLS/D 20 times on each instance: $\alpha \in \{5\%, 25\%, 50\%, 75\%, 95\%\} \times$ SO optimizer$\in$\{improved C-LK, A-NMCS\} $\times$ $k \in \{2, 3\}$ and the rest of parameters are set to their default values.

The running time of 2PIPLS/D is restricted to high limits: 30 seconds for $(p = 2, n = 100)$ instances, half an hour for $(p = 2, n = 500)$ and $(p = 3, n = 100)$ instances, and 2 hours for $(p = 3, n = 500)$ instances, then the I_H^-, I_ϵ and I_{R2} values of each generated set are computed. In the following, we generally display only the 2PIPLS/D results with C-LK as SO optimizer and $k = 3$ as PLS-VND maximum neighborhood structure size, because similar conclusions can be drawn when A-NMCS or $k = 2$ are used instead. For the sake of clarity, only I_H^- scores are displayed, knowing that same conclusions can be drawn with the other quality indicators. In order to improve the readability of graphics, we have normalized the I_H^- scores by simply dividing each I_H^- value by the maximum I_H^- value found on each type of instance.

5.4.1 Sensitivity analysis of 2PIPLS/D on SO optimizer versions

We compare separately the 2PIPLS/D results with the vanilla version of C-LK vs. the improved version of C-LK illustrated by Figure 5.4, and the vanilla version of NMCS vs. A-NMCS depicted in Figure 5.5. In both cases, we generally observe a pretty large improvement of the enhanced versions (Improved C-LK, A-NMCS) compared to vanilla version, which attests that memorizing the solutions found during the SO optimizer run is meaningful.

5.4.2 Sensitivity analysis of 2PIPLS/D on A-MDW setting

The influence of the minimum acceptance rate threshold α of A-MDW on the performance of 2PIPLS/D is exposed in Figure 5.6. The lower α is, the more weighted sum problems are solved by A-MDW, and as observed, the later the first phase (A-MDW + P-PLS) of 2PIPLS/D finishes but of better quality the set generated by first phase. Besides, when the number of iterations at second phase grows, this quality difference disappears on small-size instances ($n = 100$), while it is often maintained on large-size instances ($n = 500$). This observation reinforces the conclusions of the literature [Lust and Teghem, 2010, Paquete and Stützle, 2009b], showing the great importance of initializing PLS-VND with a large number of well-dispersed weighted-sum problems. Globally, good results are obtained when $\alpha \leq 50\%$, independently from the running time, the type, size or number of objectives of the addressed instance.

5.4.3 Sensitivity analysis of 2PIPLS/D on data perturbation setting

The influence of the data perturbation coefficient δ on the results of 2PIPLS/D is illustrated in Figure 5.7. Globally, the presence of data perturbation (i.e. when $\delta > 0$) seems to have a negligible effect (either positive or negative) on the performance of 2PIPLS/D compared to the absence of data perturbation ($\delta = 0$), except on small-size ($n = 100$) Euclidean bi-objective instances. On the other hand, a low data perturbation coefficient, i.e. $\delta \leq 5\%$ seems to work better on any type of instance compared to larger values.

5.4.4 Sensitivity analysis of 2PIPLS/D on P-PLS setting

The influence of the maximum initial size σ of parts of P-PLS on the results of 2PIPLS/D are shown in Figure 5.8. It is interesting to see that the activation of the partitioning ($\sigma < +\infty$) improves the anytime behavior of 2PIPLS/D. Firstly, as expected, the partitioning enables to reduce the time of PLS-VND, particularly on large-size instances ($n = 500$), to the extent that on tri-objective instances of size 500, P-PLS is not even

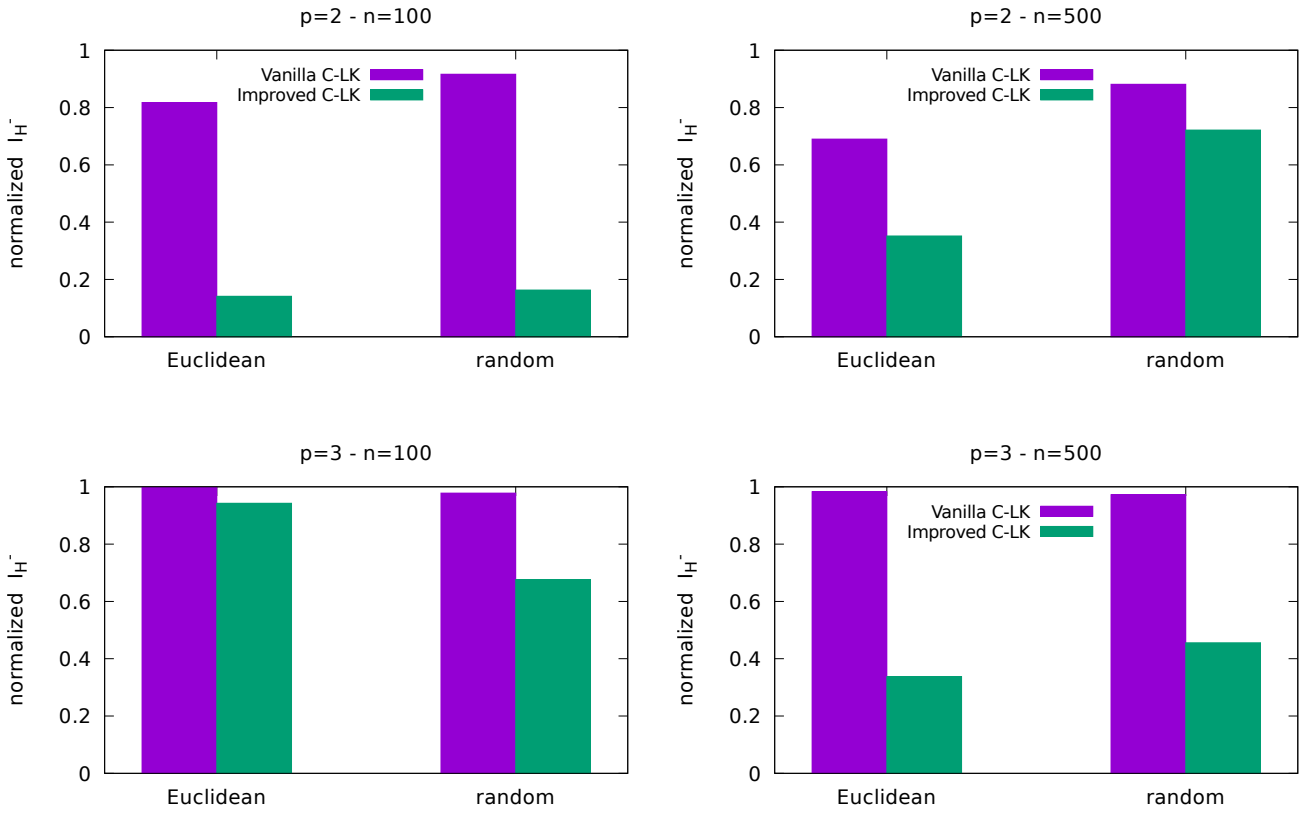


Figure 5.4 – Comparison in terms of normalized I_H^- (median values, to be minimized) between 2PIPLS/D with Vanilla C-LK and 2PIPLS/D with Improved C-LK.

finished when the allocated time is exceeded. Secondly, at a given arbitrary running time, the quality of the generated set is better when the partitioning is activated compared to when it is not, in a great majority of cases. On the other hand, it seems that $\sigma \leq 100$ works well on most instances, with an exception concerning bi-objective Euclidean instances of size 500, where a too small value of σ (i.e. $\sigma = 50$) seems to reduce the quality of the generated set. To summarize, $\sigma = 100$ obtains the best overall results.

Table 5.4 gathers some key additional information about the impact of the partitioning on the average size of the candidate edge lists managed by PLS-VND and the number of solutions examined by PLS-VND. We recall that each part built by P-PLS has its own candidate edge list associated to each city. First, we can see that the average size of candidate edge lists when the partitioning is activated ($\sigma < +\infty$) ranges from 1.3 up to 2.6 and is globally stable when the size or the number of objectives grows; while when the partitioning is deactivated ($\sigma = +\infty$), the average size of candidate edge lists ranges from 5.3 up to 18.9 and grows considerably when the size or the number of objectives grows. Indeed, the partitioning reduces from 78% (when $\sigma = 400$) up to 85% (when $\sigma = 50$) the size of candidate edge lists. As the size of the candidate edge lists is drastically reduced when partitioning is activated, then the number of solutions examined by PLS-VND is dramatically reduced: from 96.9% (when $\sigma = 400$) up to 99.4% (when $\sigma = 50$)! Despite this large reduction of exploration of the decision space, Figure 5.8 highlights that the partitioning does not lead to a reduction of the quality of the sets generated if σ is not too small (i.e. $\sigma \geq 100$) regardless of the type, size or number of objectives of the tested instances.

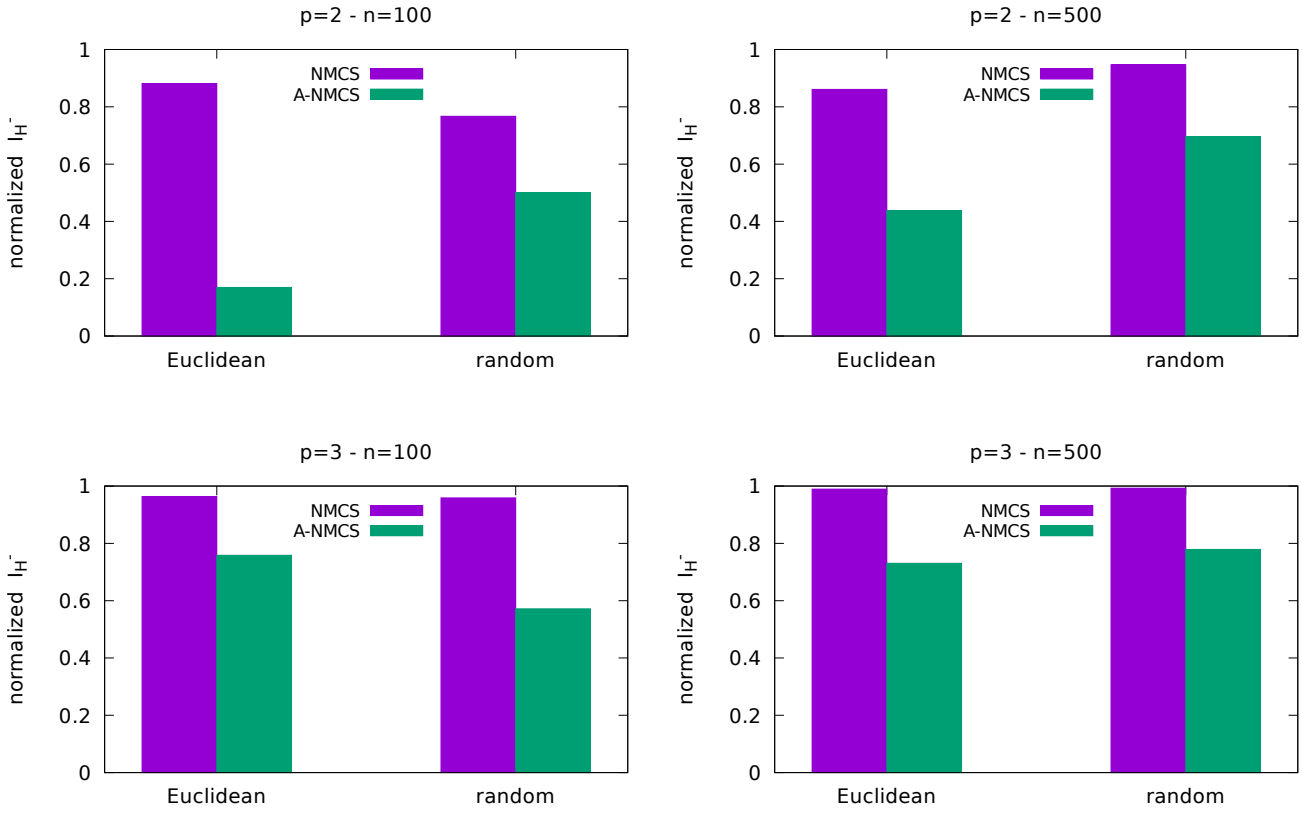


Figure 5.5 – Comparison in terms of normalized I_H^- (median values, to be minimized) between 2PIPLS/D with NMCS and 2PIPLS/D with A-NMCS.

			no partitioning ($\sigma = +\infty$)	$\sigma = 400$	$\sigma = 200$	$\sigma = 100$	$\sigma = 50$
Average size of PLS-VND's candidate edge lists	$p = 2$	$n = 100$	5.3	2.6	2.1	1.8	1.6
		$n = 500$	7.2	1.8	1.6	1.4	1.3
	$p = 3$	$n = 100$	11.0	2.6	2.2	2.0	1.8
		$n = 500$	18.9	2.4	2.1	1.9	1.6
Average decrease rate of PLS-VND's candidate edge lists size compared to no partitioning			-	-78%	-81%	-83%	-85%
Average decrease rate of nb. of solutions examined by PLS-VND compared to no partitioning			-	-96.9%	-98.1%	-98.9%	-99.4%

Table 5.4 – Impact of P-PLS partitioning on the average size of the candidate edge lists managed by PLS-VND and the number of solutions examined by PLS-VND.

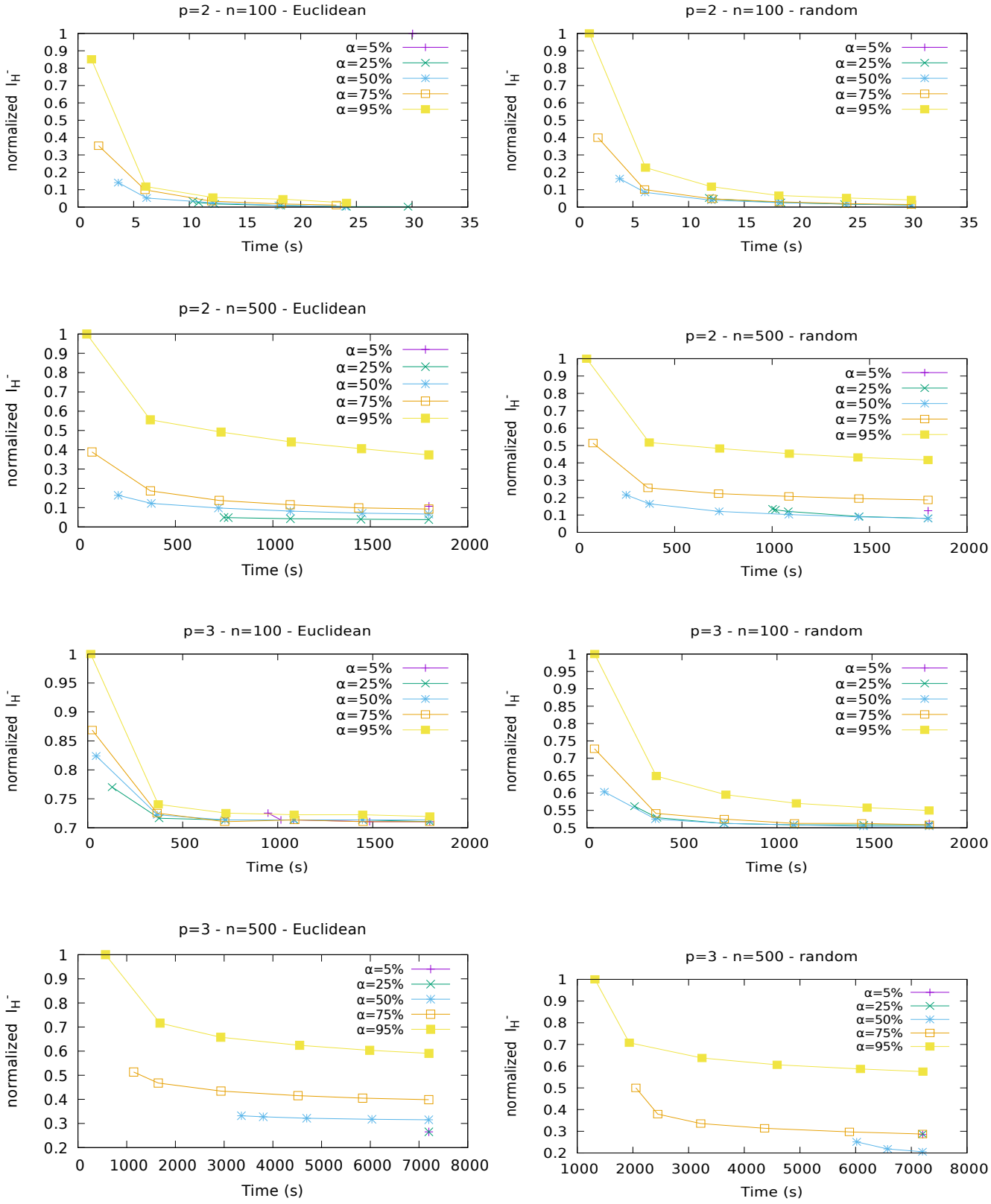


Figure 5.6 – Influence of the minimum acceptance rate threshold α of A-MDW on 2PIPLS/D performance in terms of normalized I_H^- (median values, to be minimized) in function of the running time. Curves start at the end of the first phase (A-MDW+P-PLS) of 2PIPLS/D.

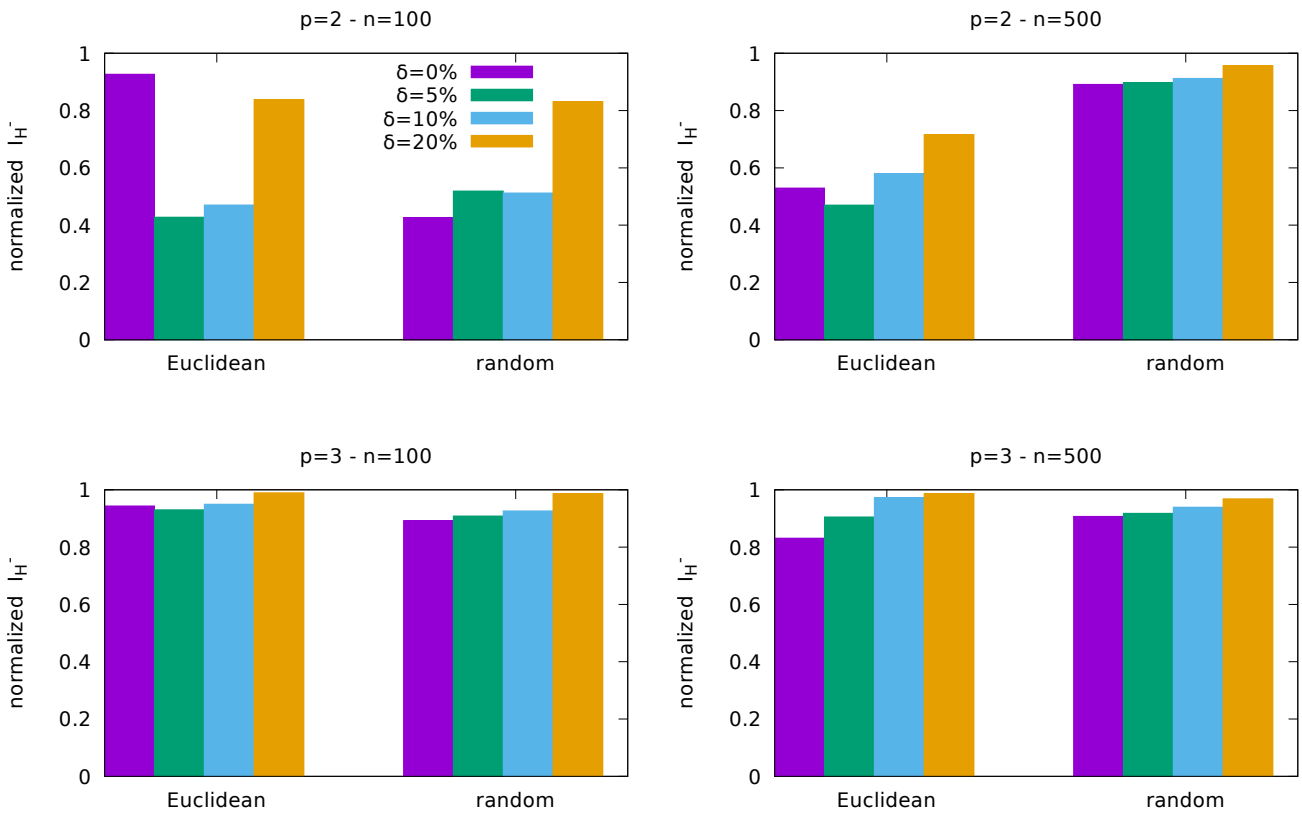


Figure 5.7 – Influence of the data perturbation coefficient δ on 2PIPLS/D performance in terms of normalized I_H^- (median values, to be minimized).

5.4.5 Sensitivity analysis of 2PIPLS/D on PLS-VND setting

We analyze the sensitivity of 2PIPLS/D performance first on the neighborhood structure size of PLS-VND, then on the choice of dependent vs independent PLS-VND, and finally on the different possible combinations of PLS-VND exploration strategies.

Maximum neighborhood structure size of PLS-VND

The comparison between the two alternatives of maximum neighborhood structure sizes ($k = 2$ or $k = 3$) is illustrated by Figure 5.9. The performance difference between the two neighborhood structure sizes is often significant for bi-objective instances, while it is generally negligible for tri-objective instances. In both cases, PLS-VND ($k = 2$) tends to catch up PLS-VND ($k = 3$) when the number of iterations grows, and this trend is much faster in the tri-objective case.

This reduction of efficiency of PLS-VND ($k = 3$) compared to PLS-VND ($k = 2$) when the number of objectives grows can be explained by the fact that PLS-VND is highly sensitive to the number of objectives (as often recalled in the literature - see [Jaszkiewicz, 2017] for example). By extension, a larger neighborhood structure is even more sensitive to an increase of the number of objectives. On the other hand, the optimization of weighted sum problems -what does the perturbation step of 2PIPLS/D- is much less sensitive from an increase of the number of objectives than PLS-VND. Consequently, the global idea is that when the number of objectives grows, the efficiency of PLS-VND decreases to the benefit of the perturbation phase. In addition, it turns out that PLS-VND ($k = 2$) consumes much less computational resources than PLS-VND

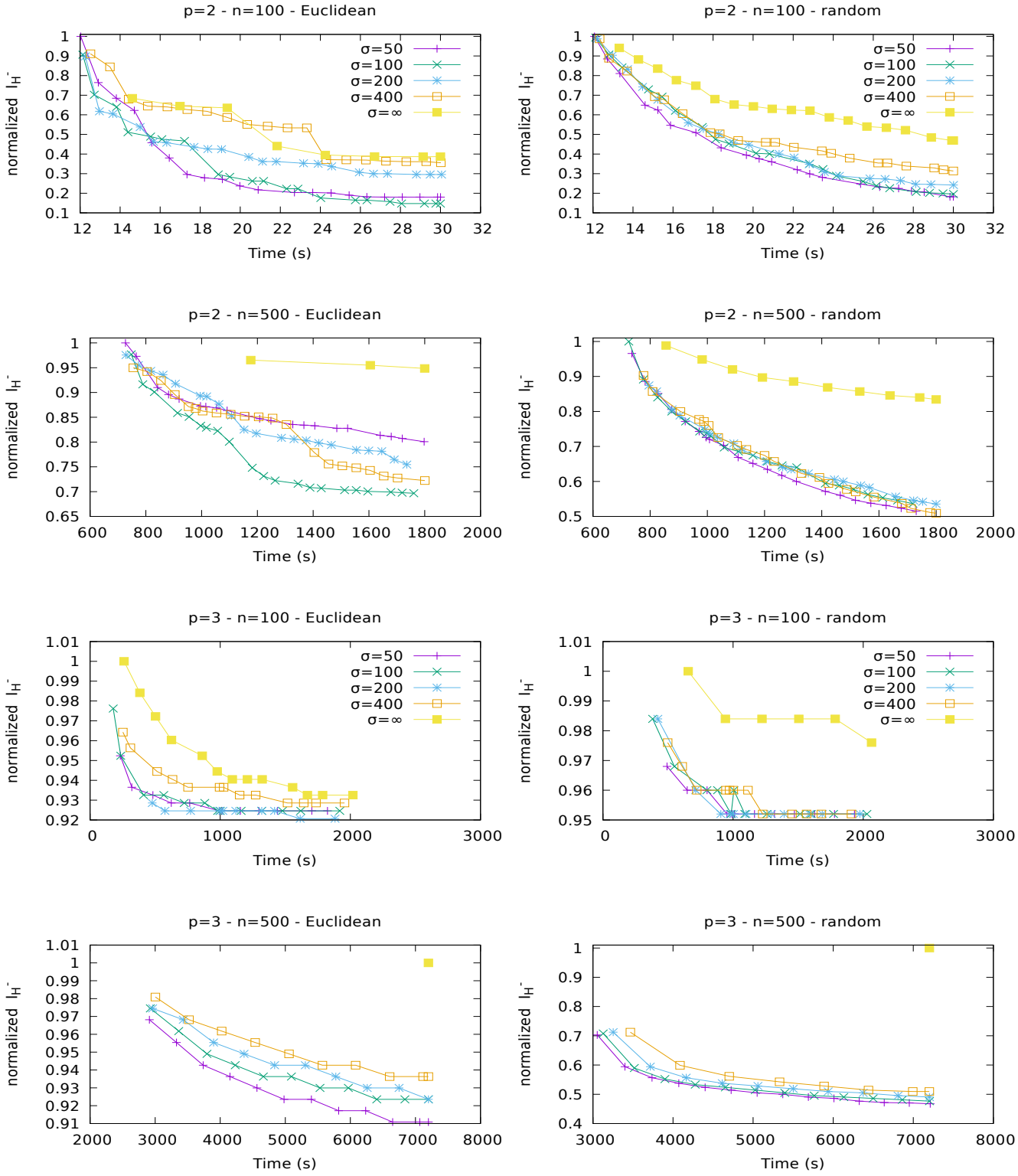


Figure 5.8 – Influence of the maximum initial size σ of part of P-PLS on 2PIPLS/D performance in terms of normalized I_H^- (median values, to be minimized) in function of the running time. Curves start at the end of the first phase (A-MDW+P-PLS) of 2PIPLS/D.

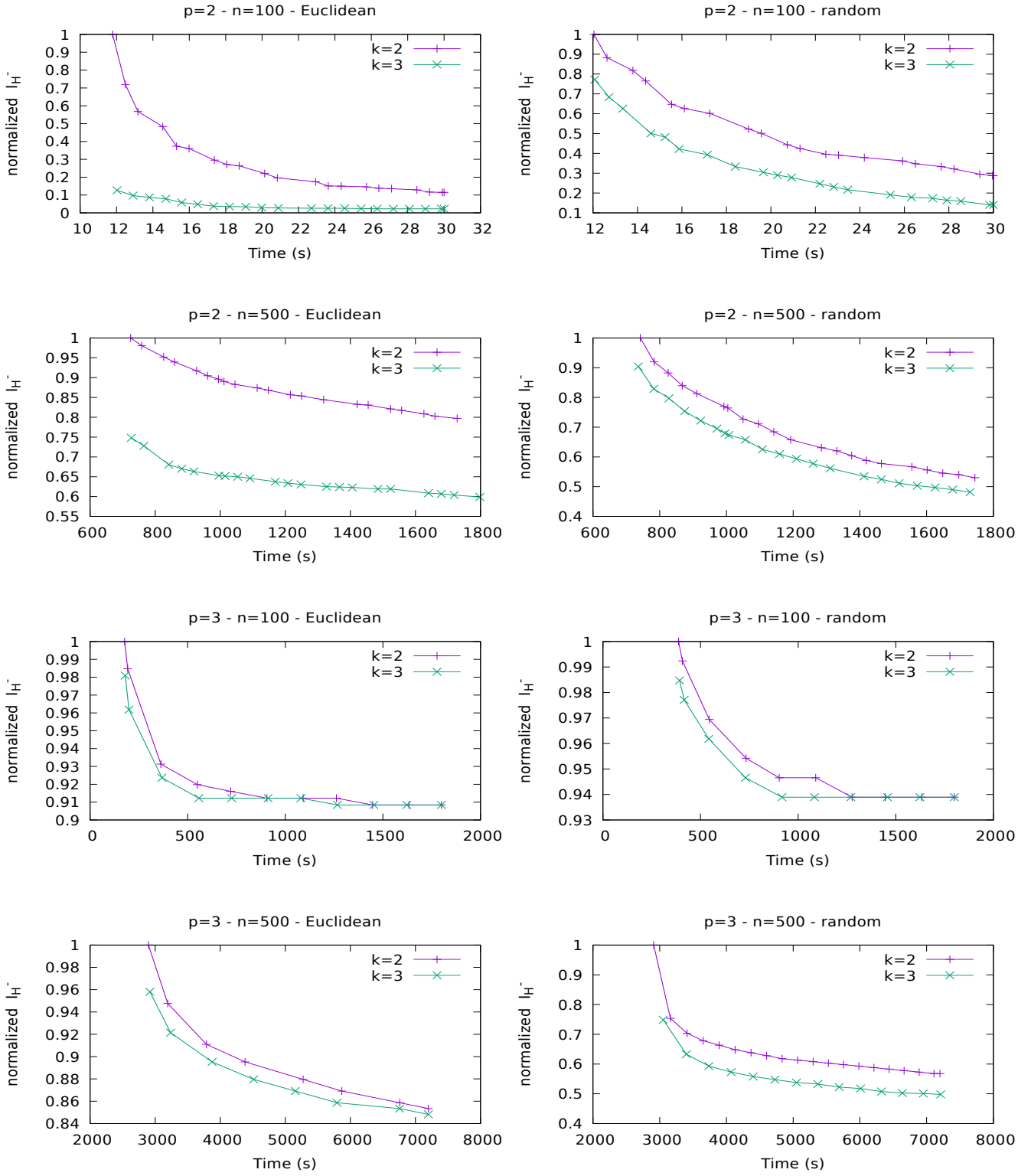


Figure 5.9 – Influence of the PLS-VND maximum neighborhood structure size k on 2PIPLS/D performance in terms of normalized I_H^- (median values, to be minimized) in function of the running time. Curves start at the end of the first phase (A-MDW+P-PLS) of 2PIPLS/D.

			PLS-VND $k = 3$	PLS-VND $k = 2$	Average reduction
Proportion of PLS-VND's execution time compared to 2PIPLS/D's total execution time	$p = 2$	$n = 100$	25.5%	8.5%	-70%
		$n = 500$	23.5%	7%	
	$p = 3$	$n = 100$	22.5%	6.5%	
		$n = 500$	22.5%	6%	

Table 5.5 – Comparison between PLS-VND ($k = 2$) and PLS-VND ($k = 3$) in terms of execution time (in proportion of 2PIPLS/D's total execution time) .

			dependent PLS-VND	independent PLS-VND	Average increase
PLS-VND's execution time in proportion of 2PIPLS/D's total execution time	$p = 2$	$n = 100$	2.5%	25.5%	+717%
		$n = 500$	2.5%	23.5%	
	$p = 3$	$n = 100$	1%	22.5%	
		$n = 500$	5.5%	22.5%	

Table 5.6 – Comparison between dependent PLS-VND and independent PLS-VND in terms of execution time.

($k = 3$), -70% in average over all types of instances, as indicated in Table 5.5. This saving allows 2PIPLS/D with PLS-VND ($k = 2$) to reallocate more computational resources on the perturbation step, and thus on weighted-sum optimizations.

Dependent vs. independent PLS-VND

The comparison between 2PIPLS/D with dependent PLS-VND and 2PIPLS/D with independent PLS-VND is depicted in Figure 5.10. There is no neat performance difference between the two alternatives, except on bi-objective Euclidean instances, where independent PLS-VND obtains better results.

It is interesting to note that these two alternatives induce drastically different allocations of computational resources during a run of 2PIPLS/D. Indeed, as independent PLS-VND takes as parameter an empty global archive, each independent PLS-VND consumes much more computational resources than a run of dependent PLS-VND, +717% in average over all types of instances, as suggested in Table 5.6.

Exploration strategies of PLS-VND

Figure 5.11 displays the performance of 2PIPLS/D with the four combinations of exploration strategies of PLS-VND. We can see that the selection of a given combination of exploration strategies does not influence the performance of 2PIPLS/D, with the exception of the (*First dominating+Explore dominated*) combination which outperforms the other combinations on small-size ($n = 100$) bi-objective Euclidean instances.

5.4.6 Final parameter setting of 2PIPLS/D

In consequence of the different analysis previously made in this section, we propose two final parameter settings of 2PIPLS/D (summarized in Table 5.7), diverging only on the SO optimizer used: the first version of 2PIPLS/D uses improved C-LK and the second uses A-NMCS. Table 5.7 also displays a ranking of the parameters in function of their influence on 2PIPLS/D performance.

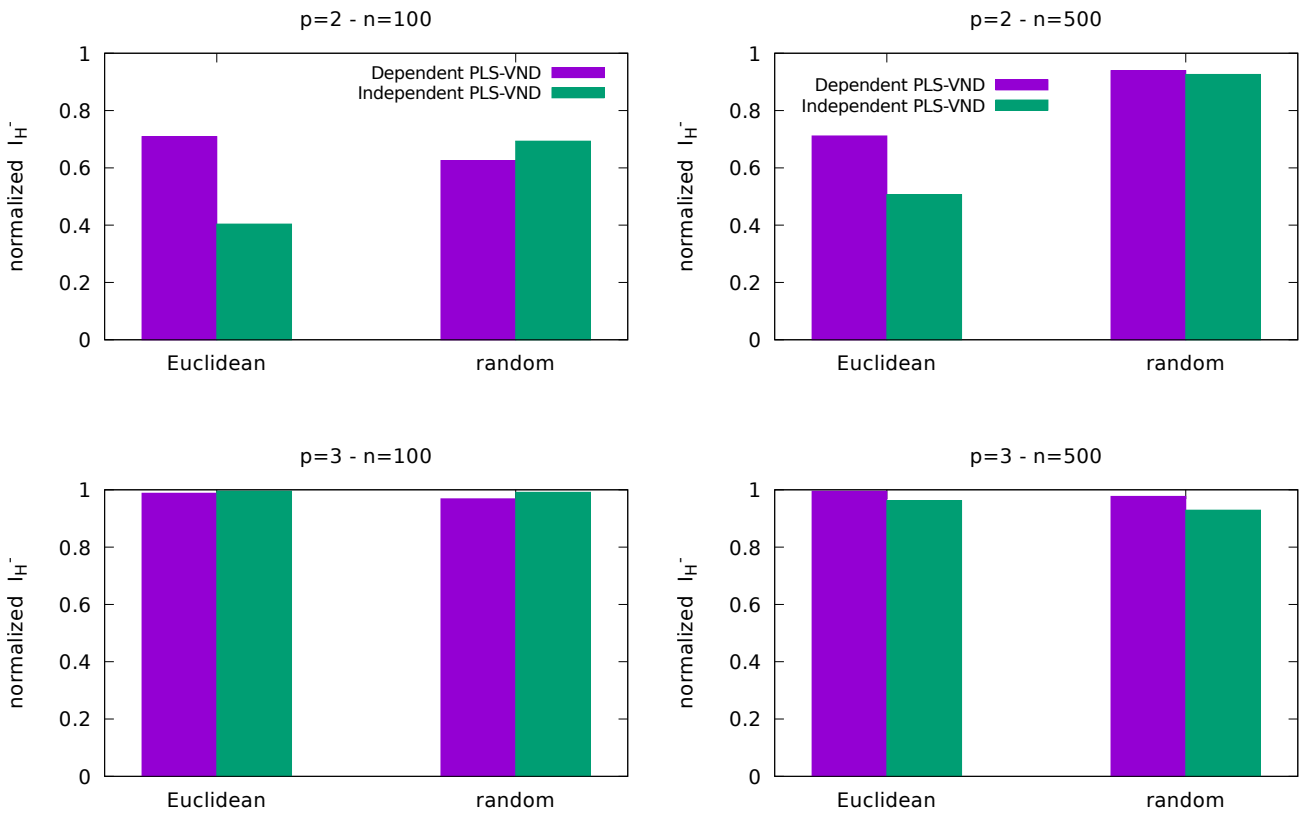


Figure 5.10 – Comparison in terms of normalized I_H^- (median values, to be minimized) between 2PIPLS/D with dependent PLS-VND and 2PIPLS/D with independent PLS-VND.

Parameter	Module concerned	Parameter rank in function of its influence on 2PIPLS/D performance	Final selected value(s)
SO optimizer	-	1	{improved C-LK, A-NMCS}
perturbation move	perturbation step (phase 2)	-	Double bridge move
acceptance rate threshold α	A-MDW	2	50%
data perturbation coefficient δ	SO optimizer (phase 2)	5	5%
maximum initial size σ of part	P-PLS	3	100
maximum neighborhood structure size k	PLS-VND	4	3
independent-pls	PLS-VND	6	yes
first-dominating	PLS-VND	6	yes
explore-dominated	PLS-VND	6	yes
tolerance for dominance relations ϵ	-	-	$\begin{cases} 0\% & \text{if } p = 2 \text{ or } n < 100 \\ 1\% & \text{otherwise} \end{cases}$
archive	-	-	$\begin{cases} \text{SAAVLA} & \text{if } p = 2 \\ \text{SANDRA} & \text{otherwise} \end{cases}$

Table 5.7 – The two final 2PIPLS/D parameter settings: 2PIPLS/D with (improved) C-LK and 2PIPLS/D with A-NMCS & a ranking of the parameters in function of their influence on 2PIPLS/D performance.

In the rest of this chapter, C-LK denotes the improved version of C-LK.

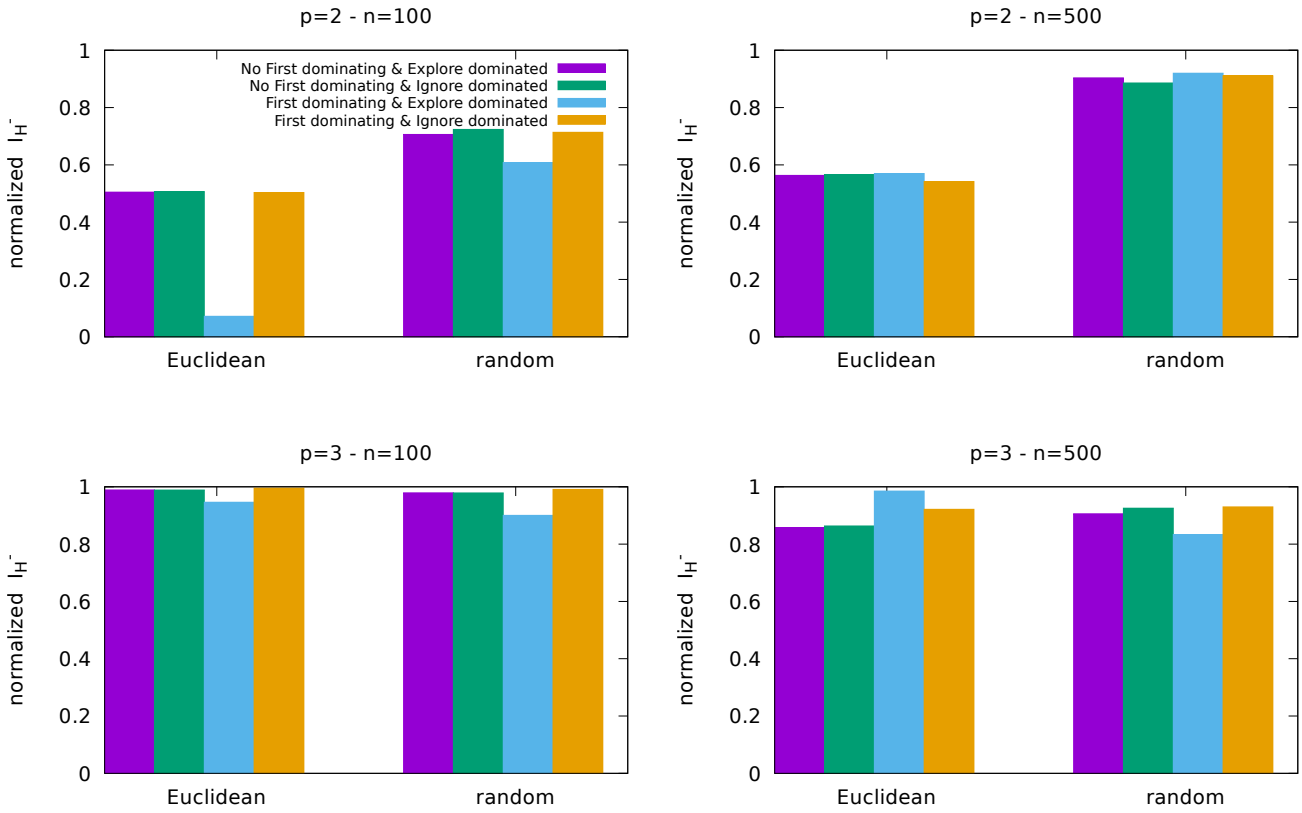


Figure 5.11 – Comparison in terms of normalized I_H^- (median values, to be minimized) between 2PIPLS/D with four different combinations of exploration strategies.

5.5 Comparison of 2PIPLS/D with state-of-the-art methods

2PIPLS/D will now be compared to the best known methods of the literature. To do so, 2PIPLS/D follows the parameter setting described in Table 5.7 such that two versions of 2PIPLS/D are considered:

- 2PIPLS/D with C-LK as SO optimizer;
- and 2PIPLS/D with A-NMCS as SO optimizer.

The running time of 2PIPLS/D will be limited to the minimum running time between all its competitors. Let us first present the selected competitors and a number of additional implementation information, then describe the results of our experiments.

5.5.1 Selection of competitors

The competitors we have chosen have already been described in Section 2.3.2 when we made a literature review for MOTSP. As the most advanced exact method of the literature, we selected AUGMECON2 [Florios and Mavrotas, 2014]. Given its limitations already mentioned, we have run **AUGMECON2** only **on bi-objective instances of size 100**. We use the executable implemented by the authors and available online[6].

[6] https://sites.google.com/site/kflorios/motsp

As the best known meta-heuristic for bi-objective TSP not proposed by us, **MoMad** [Ke et al., 2014] is selected to run **on bi-objective and tri-objective instances**. As proposed in [Ke et al., 2014], the maximum number of iterations for PLS is set to 10. During the decomposition phase, MoMad uses the MDW method to generate its set of weights. For bi-objective instances, the number of sub-problems is set to $\min(n, 600)$ as suggested in [Ke et al., 2014]. As done in [Cornu et al., 2017], we fixed the number of sub-problems to $\binom{50+3-1}{3-1} = 1326$ for (tri-objective) instances of size 50, and $\binom{60+3-1}{3-1} = 1891$ for (tri-objective) instances of size greater than or equal to 100. The number of iterations is fixed to 500 for all bi-objective instances and for tri-objective instances of size $n < 100$. For larger-sized tri-objective instances ($n \geq 100$), the number of iterations is fixed to 1000. MoMad is implemented in C/C++.

As the best known meta-heuristic for tri-objective TSP not proposed by us, **PD-TPLS** [Paquete and Stützle, 2009b] is selected to run **on tri-objective instances**. PD-TPLS also uses MDW to generate its set of weights, and we set the number of weights to $\binom{150+3-1}{3-1} = 11476$ for all instances, in order to avoid the clustering effect described in [Paquete and Stützle, 2009b]. For each generated weight, PD-TPLS optimizes the corresponding weighted sum problem by calling C-LK as suggested in [Paquete and Stützle, 2009b] as future works, instead of the 3-opt first improvement used in the original method. Internal tests have shown that PD-TPLS using C-LK gives better results. PD-TPLS is implemented in C/C++.

As the best known meta-heuristic for MOTSP, **PDA** [Cornu et al., 2017] is selected to run **on bi-objective and tri-objective instances**. PDA also uses MDW and uses the same number of sub-problems as MoMad. The running time of PDA is limited to the minimum running time between MoMad and PD-TPLS. PDA is implemented in C/C++.

5.5.2 Experimental results

2PIPLS/D, PDA, MoMad and PD-TPLS (resp. AUGMECON2) have been run 20 (resp. 1) times on the MOTSP *test benchmark* presented in Section 5.1. Tables 5.8 to 5.14 collect key information about the performance of the different methods. More precisely, the tables report for each method:

- the average value of I_ϵ, I_H^-, and I_{R2} over all runs;
- the average coverage -i.e. the proportion of (potentially) efficient solutions in a given approximation- over all runs, and the coverage of the union of the runs;
- the average number of solutions examined;
- the average running time;
- the average RAM utilized;
- the average size of the final approximation set $\overline{X}_{all}$ generated;
- the size of the (*exact* when $p = 2$ and $n = 100$, *approximated* otherwise) non-dominated set $\widetilde{Y}_{nd}$. When the (exact) non-dominated set is not known, we recall that its approximation is built by merging the final approximation sets provided by all the methods.

Figures 5.12 to 5.19 detail the I_ϵ, I_H^-, and I_{R2} performance of the approximation sets produced by 2PIPLS/D (with C-LK), PDA, MoMad and PD-TPLS.

Globally, 2PIPLS/D with C-LK obtains good results. Concerning bi-objective instances of size less than or equal to 100, when the 20 runs of the method are merged together, at least 99% of the (approximate) non-dominated points are found (i.e. coverage is greater than or equal to 99%). In particular, **2PIPLS/D is the only meta-heuristic which was able to find the (exact) non-dominated set of some instances** (7

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-5})	I_{R2} (10^{-1})	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\lvert\overline{X}_{all}\rvert$	$\lvert\widetilde{Y}_{nd}\rvert$
kroAB100	**2PIPLS/D (C-LK)**	**0.12**	**0.23**	**1.91**	**97 → 100**	7	11	**< 1**	3,288	3,332
	PDA	0.20	2.06	11.50	82 → 98	**2**	11	**< 1**	3,015	
	2PIPLS/D (A-NMCS)	0.34	9.55	58.30	77 → 98	16	11	**< 1**	3,146	
	MoMad	0.39	17.40	97.90	47 → 66	3	11	**< 1**	2,569	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*72h*	< 4	3,332	
kroAC100	**2PIPLS/D (C-LK)**	**0.14**	**0.17**	**0.77**	**98 → 100**	6	11	**< 1**	2,433	2,458
	PDA	0.18	1.60	11.40	84 → 98	**2**	11	**< 1**	2,263	
	2PIPLS/D (A-NMCS)	0.32	5.01	24.30	78 → 99.5	13	11	**< 1**	2,288	
	MoMad	0.40	14.30	68.50	52 → 71	**2**	11	**< 1**	2,062	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*40h*	< 4	2,458	
kroAD100	**2PIPLS/D (C-LK)**	**0.09**	**0.14**	**0.75**	**97 → 100**	5	10	**< 1**	2,314	2,351
	PDA	0.16	1.84	8.10	83 → 98	**2**	10	**< 1**	2,145	
	2PIPLS/D (A-NMCS)	0.35	8.71	37.30	74 → 99.1	11	10	**< 1**	2,197	
	MoMad	0.35	17.40	66.40	49 → 67	**2**	10	**< 1**	1,889	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*26h*	< 4	2,351	
kroBC100	**2PIPLS/D (C-LK)**	**0.08**	**0.09**	**0.21**	**98 → 100**	6	11	**< 1**	2,723	2,752
	PDA	0.14	1.55	5.58	83 → 98	**2**	11	**< 1**	2,538	
	2PIPLS/D (A-NMCS)	0.21	5.78	14.80	83 → 99.7	15	11	**< 1**	2,625	
	MoMad	0.28	12.20	46.50	47 → 68	**2**	11	**< 1**	2,205	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*41h*	< 4	2,752	
kroBD100	**2PIPLS/D (C-LK)**	**0.11**	**0.18**	**1.48**	**97 → 100**	6	11	**< 1**	2,632	2,657
	PDA	0.16	1.96	7.16	81 → 98	**2**	11	**< 1**	2,396	
	2PIPLS/D (A-NMCS)	0.28	10.30	33.10	72 → 98	12	11	**< 1**	2,454	
	MoMad	0.43	14.40	67.90	50 → 69	**2**	11	**< 1**	2,004	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*35h*	< 4	2,657	
kroCD100	**2PIPLS/D (C-LK)**	**0.10**	**0.13**	**0.77**	**98 → 100**	5	10	**< 1**	2,028	2,044
	PDA	0.19	1.69	6.91	86 → 99	**1**	10	**< 1**	1,907	
	2PIPLS/D (A-NMCS)	0.40	8.13	35.20	78 → 99	10	10	**< 1**	1,892	
	MoMad	0.41	15.10	65.70	56 → 74	2	10	**< 1**	1,742	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*21h*	< 4	2,044	
euclidAB100	**2PIPLS/D (C-LK)**	**0.11**	**0.21**	**1.64**	**97 → 99.9**	5	10	**< 1**	1,787	1,812
	PDA	0.21	3.03	12.30	79 → 97	**1**	10	**< 1**	1,612	
	2PIPLS/D (A-NMCS)	0.33	9.44	37.80	74 → 99	12	10	**< 1**	1,697	
	MoMad	0.36	19.60	85.60	48 → 67	2	10	**< 1**	1,400	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*22h*	< 4	1,812	
euclidCD100	**2PIPLS/D (C-LK)**	**0.12**	**0.37**	**1.24**	**96 → 100**	5	11	**< 1**	2,225	2,268
	PDA	0.20	2.97	12.00	79 → 98	**2**	11	**< 1**	2,008	
	2PIPLS/D (A-NMCS)	0.33	11.60	53.70	71 → 98	18	11	**< 1**	2,130	
	MoMad	0.42	23.30	96.70	45 → 63	**2**	11	**< 1**	1,683	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*36h*	< 4	2,268	
euclidEF100	**2PIPLS/D (C-LK)**	**0.14**	**0.50**	**2.67**	**97 → 99.9**	5	11	**< 1**	2,488	2,530
	PDA	0.22	2.71	9.96	84 → 99	**2**	11	**< 1**	2,319	
	2PIPLS/D (A-NMCS)	0.32	8.13	39.50	80 → 99	12	11	**< 1**	2,373	
	MoMad	0.48	24.00	102.00	49 → 70	**2**	11	**< 1**	1,967	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*28h*	< 4	2,530	
ClusterAB100	**2PIPLS/D (C-LK)**	**0.12**	**0.49**	**1.82**	**95 → 99.8**	12	14	**< 1**	2,981	3,036
	PDA	0.16	2.01	7.21	82 → 97	**2**	14	**< 1**	2,762	
	2PIPLS/D (A-NMCS)	0.29	5.78	16.20	77 → 98	3	14	**< 1**	2,850	
	MoMad	0.39	15.90	51.90	51 → 70	34	15	**< 1**	2,454	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*28h*	< 4	3,036	

Table 5.8 – Comparison between 2PIPLS/D, PDA, MoMad and AUGMECON2 performance on Euclidean and clustered bi-objective instances of size 100.

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-4})	I_{R2}	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\lvert\overline{X}_{all}\rvert$	$\lvert\widetilde{Y}_{nd}\rvert$
rdAB100	**2PIPLS/D (C-LK)**	**0.39**	**0.25**	**1.53**	**74 → 99.2**	**1**	12	**< 1**	1,469	1,707
	PDA	0.63	1.14	4.90	46 → 93	**1**	12	**< 1**	1,161	
	MoMad	2.20	8.53	44.30	10 → 18	**1**	12	**< 1**	641	
	2PIPLS/D (A-NMCS)	3.70	14.10	52.50	06 → 51	19	12	**< 1**	1,121	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*18h*	< 4	1,707	
rdCD100	**2PIPLS/D (C-LK)**	**0.37**	**0.22**	**1.28**	**78 → 99.5**	**1**	11	**< 1**	1,620	1,850
	PDA	0.69	1.18	5.24	48 → 93	**1**	11	**< 1**	1,258	
	MoMad	2.23	7.54	43.20	9 → 16	**1**	11	**< 1**	660	
	2PIPLS/D (A-NMCS)	4.05	12.90	42.90	10 → 56	18	11	**< 1**	1,182	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*21h*	< 4	1,850	
rdEF100	**2PIPLS/D (C-LK)**	**0.42**	**0.41**	**1.77**	**69 → 99**	2	12	**< 1**	1,596	1,882
	PDA	0.75	1.53	6.68	40 → 90	**1**	12	**< 1**	1,221	
	MoMad	2.48	8.74	46.20	9 → 15	**1**	12	**< 1**	664	
	2PIPLS/D (A-NMCS)	4.70	17.20	66.10	4 → 32	21	12	**< 1**	1,152	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*24h*	< 4	1,882	
mixedGG100	**2PIPLS/D (C-LK)**	**0.28**	**0.16**	**0.40**	**87 → 99.8**	3	11	**< 1**	1,714	1,848
	PDA	0.46	0.64	1.38	66 → 96	**1**	11	**< 1**	1,461	
	MoMad	1.52	5.14	21.80	25 → 41	**1**	11	**< 1**	983	
	2PIPLS/D (A-NMCS)	2.49	5.72	11.00	48 → 79	18	11	**< 1**	1,441	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*20h*	< 4	1,848	
mixedHH100	**2PIPLS/D (C-LK)**	**0.35**	**0.24**	**0.65**	**83 → 99.2**	4	12	**< 1**	1,920	2,108
	PDA	0.55	0.81	2.42	65 → 96	**1**	12	**< 1**	1,698	
	MoMad	1.19	4.40	21.30	28 → 42	**1**	12	**< 1**	1,200	
	2PIPLS/D (A-NMCS)	2.74	5.42	14.50	49 → 83	16	12	**< 1**	1,692	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*19h*	< 4	2,108	
mixedII100	**2PIPLS/D (C-LK)**	**0.33**	**0.11**	**0.71**	**89 → 99.8**	4	12	**< 1**	1,768	1,883
	PDA	0.45	0.62	3.28	64 → 97	**1**	12	**< 1**	1,499	
	MoMad	1.33	4.86	23.80	25 → 38	**1**	12	**< 1**	1,019	
	2PIPLS/D (A-NMCS)	3.46	6.25	15.80	43 → 85	20	12	**< 1**	1,518	
	AUGMECON2	*0*	*0*	*0*	*100*	-	*22h*	< 4	1,883	

Table 5.9 – Comparison between 2PIPLS/D, PDA, MoMad and AUGMECON2 performance on random and mixed bi-objective instances of size 100.

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-4})	I_{R2}	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\lvert\overline{X}_{all}\rvert$	$\lvert\widetilde{Y}_{nd}\rvert$
kroAB200	**2PIPLS/D (C-LK)**	**0.06**	**0.02**	**0.37**	**93 → 99.9**	17	59	**< 1**	8,703	8,915
	PDA	0.12	0.15	1.51	71 → 95	**16**	59	**< 1**	7,880	
	MoMad	0.35	0.97	9.70	33 → 47	17	59	**< 1**	6,649	
	2PIPLS/D (A-NMCS)	0.45	3.62	39.90	15 → 65	30	59	**< 1**	7,856	
kroAB300	**2PIPLS/D (C-LK)**	**0.09**	**0.09**	**0.57**	**86 → 99**	**38**	169	**< 1**	18,569	19,032
	PDA	**0.09**	0.15	2.06	60 → 94	70	169	**< 1**	16,963	
	MoMad	0.18	0.63	7.53	23 → 34	69	169	**< 1**	14,880	
	2PIPLS/D (A-NMCS)	0.76	9.74	134	1 → 10	87	169	**< 1**	17,171	
euclidAB300	**2PIPLS/D (C-LK)**	**0.07**	**0.08**	**1.33**	**79 → 99**	**46**	164	**< 1**	17,779	18,523
	PDA	0.08	0.20	3.32	57 → 94	67	164	**< 1**	16,334	
	MoMad	0.13	0.72	10.40	25 → 34	63	164	**< 1**	14,436	
	2PIPLS/D (A-NMCS)	0.70	9.98	147	1 → 14	95	164	**< 1**	16,516	
ClusterAB300	**2PIPLS/D (C-LK)**	**0.09**	**0.05**	**1.00**	**84 → 99**	**58**	209	**< 1**	21,124	21,617
	PDA	0.20	0.39	3.62	42 → 85	203	209	**< 1**	18,488	
	MoMad	0.62	3.42	21	11 → 19	113	209	**< 1**	15,931	
	2PIPLS/D (A-NMCS)	1.18	9.11	115	1 → 11	121	209	**< 1**	18,661	
kroAB400	**2PIPLS/D (C-LK)**	**0.08**	**0.10**	**2.05**	**68 → 96**	**75**	346	**< 1**	28,677	30,505
	PDA	0.11	0.28	5.11	37 → 80	178	346	1.6	25,027	
	MoMad	0.21	0.82	12.10	11 → 17	152	346	1.5	21,793	
	2PIPLS/D (A-NMCS)	1.04	12.30	231	0 → 0	107	346	**< 1**	26,539	
kroAB500	**2PIPLS/D (C-LK)**	**0.06**	**0.15**	**4.63**	**45 → 93**	**157**	632	**< 1**	43,372	46,144
	PDA	0.11	0.39	11.00	17 → 65	430	632	2.6	37,863	
	MoMad	0.22	0.83	18.50	6 → 9	339	632	2.5	33,436	
	2PIPLS/D (A-NMCS)	1.05	13.60	323	0 → 0	231	633	**< 1**	37,707	
euclidAB500	**2PIPLS/D (C-LK)**	**0.08**	**0.16**	**3.59**	**58 → 96**	**161**	605	**< 1**	42,528	44,989
	PDA	0.09	0.38	9.82	24 → 77	392	606	2.6	37,984	
	MoMad	0.12	0.78	19.30	08 → 12	338	605	2.5	34,148	
	2PIPLS/D (A-NMCS)	0.97	13.80	348	0 → 0	257	605	**< 1**	38,378	
ClusterAB500	**2PIPLS/D (C-LK)**	0.16	**0.11**	**4.62**	**68 → 97**	**124**	677	1.3	52,148	54,415
	PDA	**0.13**	0.22	4.93	32 → 79	577	678	2.6	45,706	
	MoMad	0.71	0.86	15.60	8 → 12	440	677	2.5	40,936	
	2PIPLS/D (A-NMCS)	1.46	12.90	417	0 → 0	247	678	**< 1**	45,295	
kroAB750	**2PIPLS/D (C-LK)**	**0.10**	**0.21**	**8.02**	**30 → 87**	**260**	1,417	2.3	78,341	84,807
	PDA	0.12	0.55	21.2	4 → 33	1,462	1,417	6.1	66,365	
	MoMad	0.16	0.86	30.40	1 → 02	1,151	1,417	6.0	60,415	
	2PIPLS/D (A-NMCS)	1.52	16.90	629	0 → 0	831	1,418	**1.3**	77,438	
kroAB1000	**2PIPLS/D (C-LK)**	**0.09**	**0.32**	**10.50**	**14 → 67**	**365**	2,465	5.8	120,607	128,282
	PDA	0.16	0.79	35.10	1 → 10	3,264	2,466	10.3	105,174	
	MoMad	0.26	1.06	44.90	0 → 0	2,663	2,465	10.2	98,476	
	2PIPLS/D (A-NMCS)	2.03	19.00	941	0 → 0	1,218	2,468	**1.9**	120,008	

Table 5.10 – Comparison between 2PIPLS/D, PDA and MoMad performance on large-size Euclidean and clustered bi-objective instances.

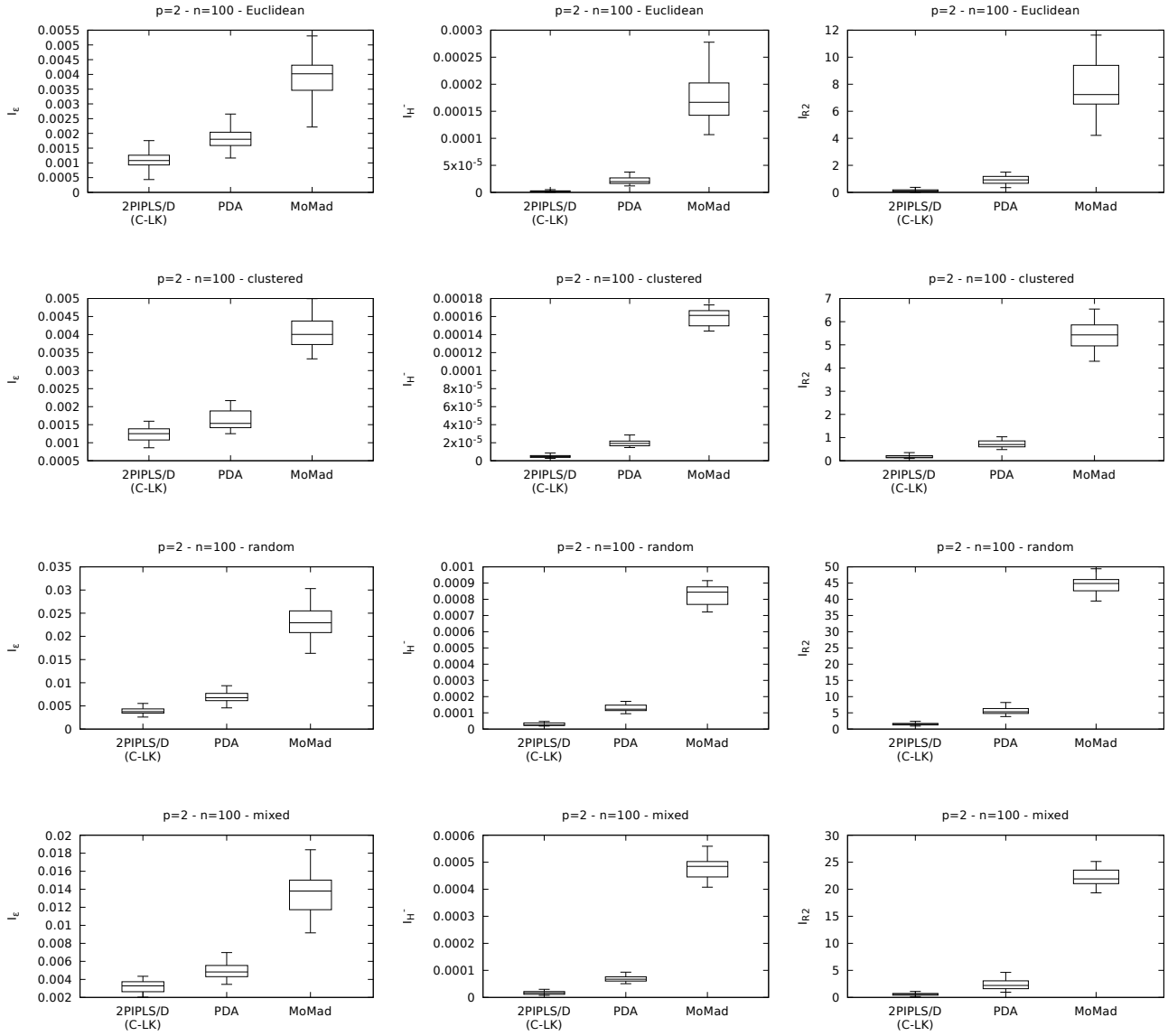


Figure 5.12 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, and MoMad on bi-objective instances of size 100.

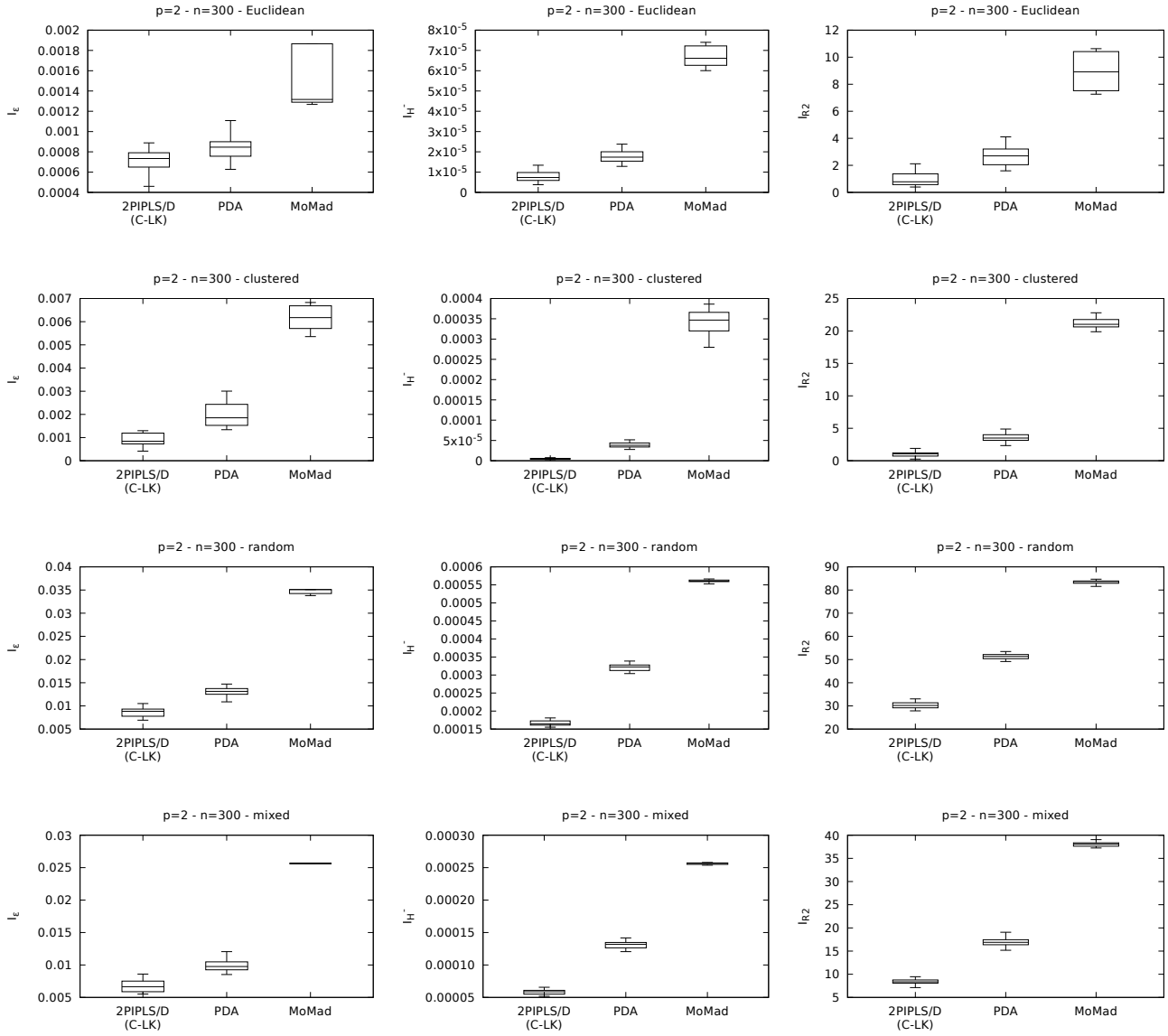


Figure 5.13 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, and MoMad on bi-objective instances of size 300.

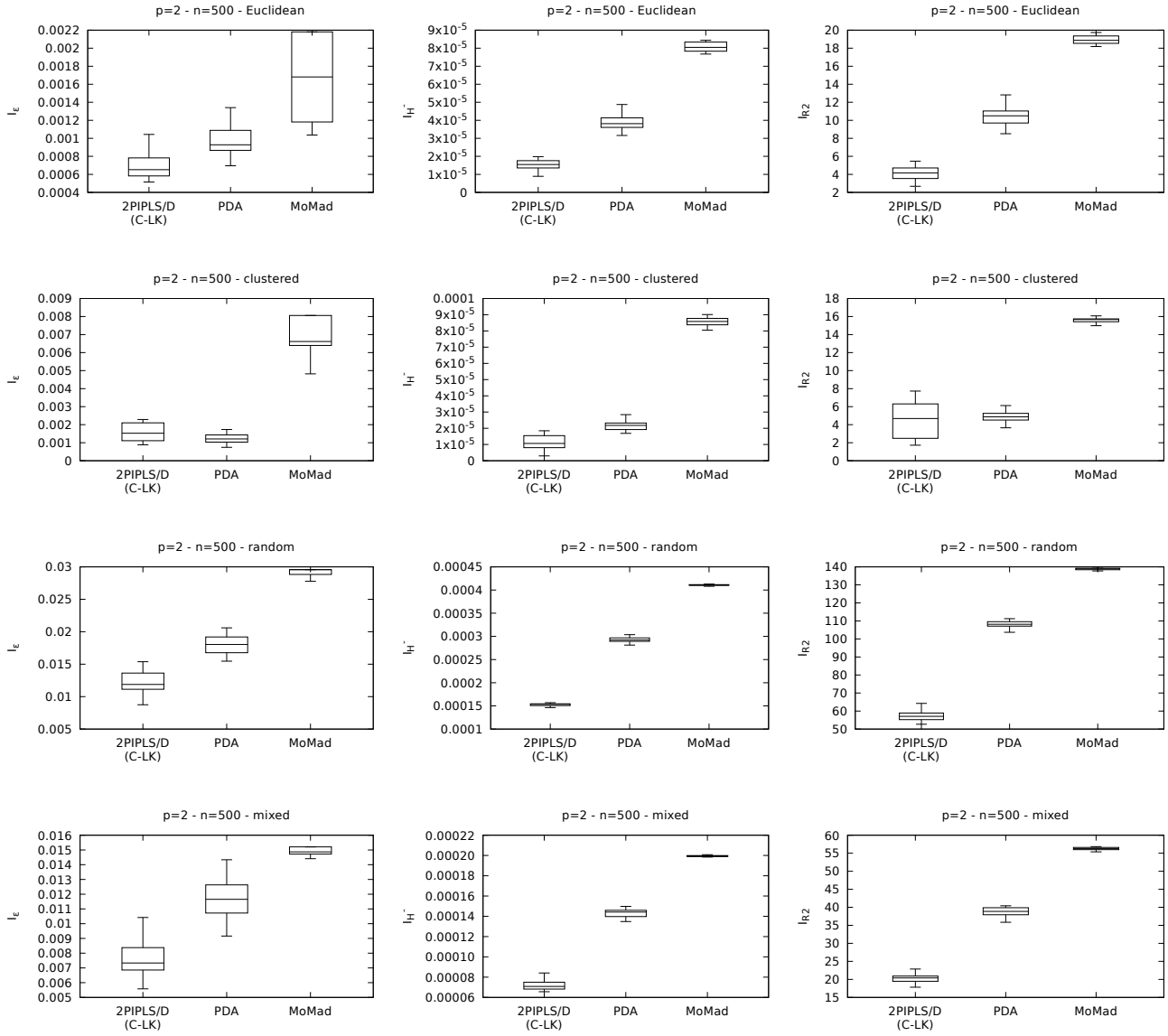


Figure 5.14 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, and MoMad on bi-objective instances of size 500.

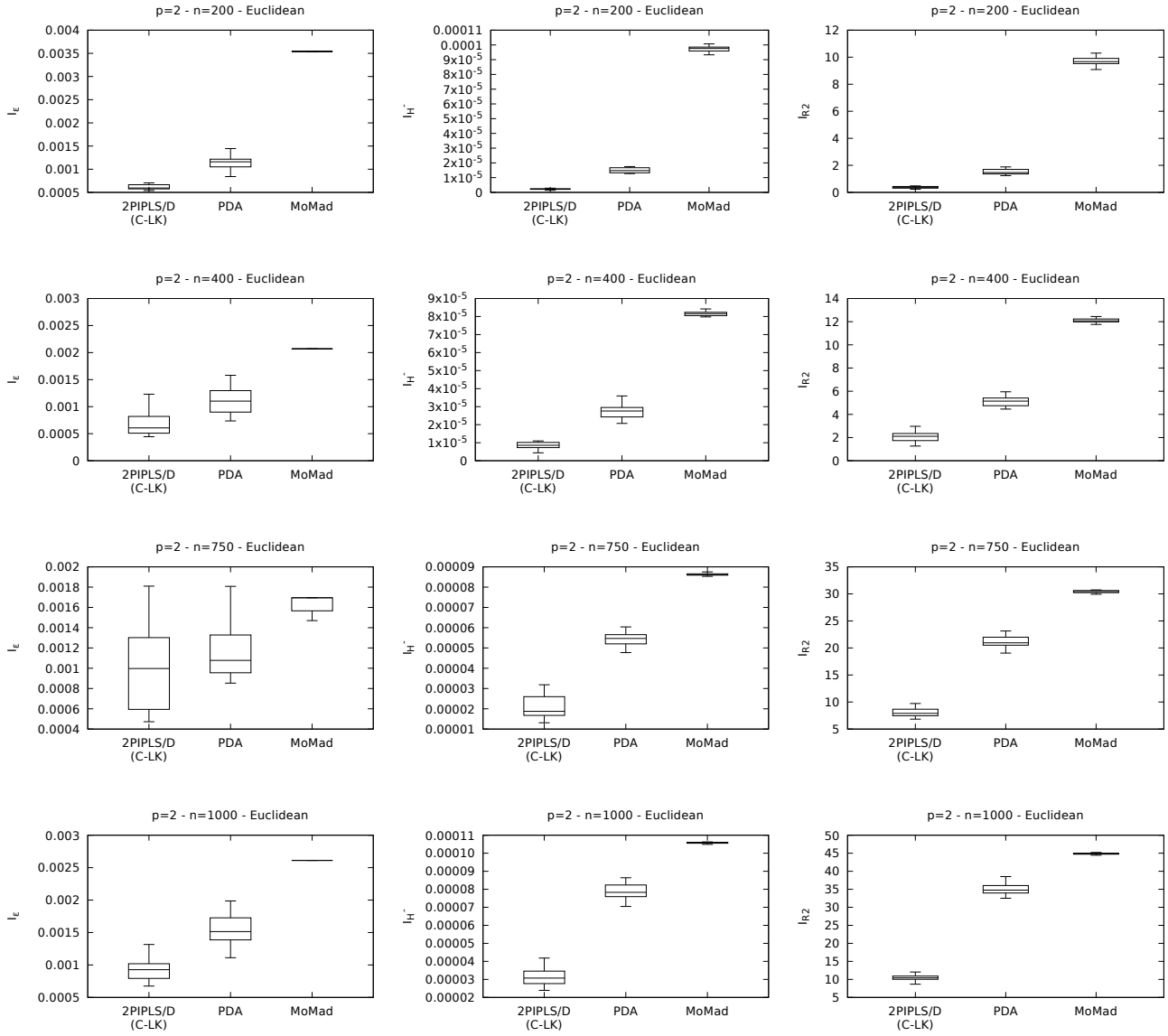


Figure 5.15 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, and MoMad on bi-objective Euclidean instances of size 200, 400, 750 and 1000.

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-4})	I_{R2}	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\lvert\overline{X}_{all}\rvert$	$\lvert\widetilde{Y}_{nd}\rvert$
mixedAB300	**2PIPLS/D (C-LK)**	**0.68**	**0.58**	**8.34**	**46 → 81**	**28**	185	**< 1**	9,636	12,490
	PDA	1.00	1.31	16.90	24 → 58	33	185	**< 1**	7,477	
	MoMad	2.56	2.56	38.10	7 → 12	**28**	185	**< 1**	5,786	
	2PIPLS/D (A-NMCS)	13	17.60	232	1 → 7	134	185	**< 1**	8,865	
rdAB300	**2PIPLS/D (C-LK)**	**0.87**	**1.67**	**30.30**	**3 → 35**	**9**	205	**< 1**	4,769	9,168
	PDA	1.30	3.21	51.10	0 → 7	13	205	**< 1**	3,231	
	MoMad	3.45	5.60	83.40	0 → 0	10	205	**< 1**	2,079	
	2PIPLS/D (A-NMCS)	14.10	32.60	546	0 → 0	126	205	**< 1**	4,481	
mixedAB500	**2PIPLS/D (C-LK)**	**0.76**	**0.72**	**20.10**	**29 → 70**	**67**	721	**< 1**	21,976	26,526
	PDA	1.17	1.44	38.80	10 → 40	182	721	2.6	16,692	
	MoMad	1.49	2.00	56.20	3 → 4	145	721	2.5	14,008	
	2PIPLS/D (A-NMCS)	16.90	20.90	544	0 → 0	323	722	**< 1**	21,115	
rdAB500	**2PIPLS/D (C-LK)**	**1.22**	**1.54**	**57.30**	**1 → 13**	**27**	750	**< 1**	7,660	12,155
	PDA	1.81	2.93	108	0 → 0	38	750	2.6	4,822	
	MoMad	2.90	4.11	139	0 → 0	30	750	2.5	3,518	
	2PIPLS/D (A-NMCS)	22.30	38.80	1400	0 → 0	269	751	**< 1**	9,316	

Table 5.11 – Comparison between 2PIPLS/D, PDA and MoMad performance on mixed and random bi-objective instances of size 300 and 500.

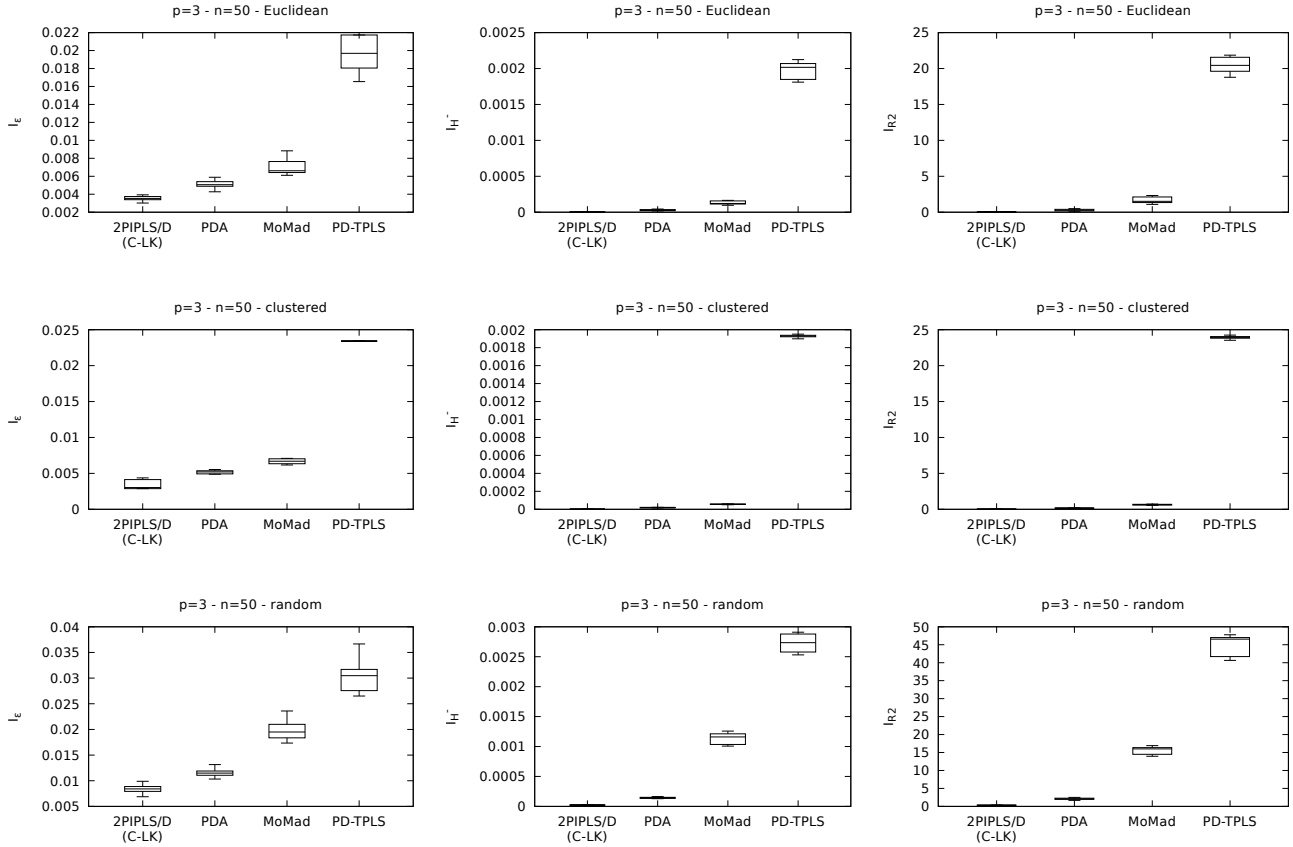


Figure 5.16 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, MoMad and PD-TPLS on tri-objective instances of size 50.

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-3})	I_{R2}	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\lvert\overline{X}_{all}\rvert$	$\lvert\widetilde{Y}_{nd}\rvert$
	2PIPLS/D (C-LK)	**0.37**	**< 0.01**	**0.03**	**99 → 99.9**	392	228	**< 1**	102,165	
	2PIPLS/D (A-NMCS)	**0.37**	**< 0.01**	0.05	**99 → 99.9**	647	231	**< 1**	102,595	
euclidA-3-50	PDA	0.55	0.02	0.17	91 → 98	**73**	224	**< 1**	96,292	103,439
	MoMad	0.79	0.11	1.37	81 → 90	83	224	**< 1**	91,192	
	PD-TPLS	1.80	2.01	21.70	15 → 16	897	738	**< 1**	25,290	
	2PIPLS/D (C-LK)	**0.34**	**< 0.01**	**0.04**	98 → 99.8	325	236	**< 1**	126,201	
	2PIPLS/D (A-NMCS)	0.37	**< 0.01**	0.06	**99 → 99.9**	549	232	**< 1**	126,724	
euclidB-3-50	PDA	0.50	0.04	0.43	85 → 97	**87**	228	**< 1**	113,181	128,273
	MoMad	0.69	0.16	2.20	73 → 83	99	228	**< 1**	106,154	
	PD-TPLS	1.97	2.08	20.30	12 → 13	920	751	**< 1**	27,493	
	2PIPLS/D (C-LK)	**0.39**	**< 0.01**	**0.02**	98 → 99.8	436	262	**< 1**	126,996	
	2PIPLS/D (A-NMCS)	0.43	**< 0.01**	0.03	**99 → 100**	668	262	**< 1**	127,628	
euclidC-3-50	PDA	0.50	0.03	0.28	88 → 98	**89**	258	**< 1**	116,378	128,616
	MoMad	0.64	0.12	1.45	77 → 87	107	258	**< 1**	109,375	
	PD-TPLS	2.13	1.84	19.50	15 → 16	916	721	**< 1**	31,783	
	2PIPLS/D (A-NMCS)	**0.30**	**< 0.01**	**0.02**	**99 → 100**	829	298	**< 1**	109,861	
	2PIPLS/D (C-LK)	0.34	**< 0.01**	0.03	**99** → 99.9	543	301	**< 1**	109,622	
clusterA-3-50	PDA	0.52	0.02	0.16	92 → 98	**78**	295	**< 1**	103,497	110,447
	MoMad	0.67	0.06	0.65	87 → 92	100	327	**< 1**	100,917	
	PD-TPLS	2.33	1.93	23.90	14 → 15	1,076	418	**< 1**	26,867	
	2PIPLS/D (C-LK)	**1.02**	**1.68**	**37.30**	**16** → 27	295	801	**< 1**	16,884	
	2PIPLS/D (A-NMCS)	1.13	1.77	40.50	15 → 32	672	801	**< 1**	16,797	
kroABC100	PDA	1.18	1.85	39.30	12 → **46**	**43**	801	**< 1**	16,127	103,540
	MoMad	1.34	2.33	50.40	11 → 21	69	810	**< 1**	15,608	
	PD-TPLS	2.14	2.30	50.00	11 → 17	2,219	933	**< 1**	13,982	
	2PIPLS/D (C-LK)	**1.03**	**2.35**	**46.10**	**8** → 14	310	776	**< 1**	13,246	
	2PIPLS/D (A-NMCS)	1.14	2.48	49.80	**8** → 16	501	776	**< 1**	13,179	
euclidABC100	PDA	1.15	2.49	48.20	7 → **22**	**30**	784	**< 1**	12,687	164,003
	MoMad	1.35	3.10	58.70	5 → 10	53	784	**< 1**	12,196	
	PD-TPLS	2.65	3.09	59.50	5 → 8	2,455	982	**< 1**	11,124	
	2PIPLS/D (C-LK)	**1.03**	**1.79**	**35.30**	**15** → 28	327	774	**< 1**	14,243	
	PDA	1.16	1.96	37.00	12 → **46**	**34**	774	**< 1**	13,580	
euclidDEF100	2PIPLS/D (A-NMCS)	1.16	1.95	40.10	14 → 34	537	774	**< 1**	14,170	93,030
	MoMad	1.34	2.61	48.10	10 → 22	58	782	**< 1**	13,016	
	PD-TPLS	1.95	2.59	49.20	10 → 18	2,472	1,007	**< 1**	11,906	
	2PIPLS/D (C-LK)	**1.09**	**1.40**	**29.90**	**15** → 34	157	895	**< 1**	17,466	
	2PIPLS/D (A-NMCS)	1.17	1.49	31.00	**15** → 32	621	895	**< 1**	17,487	
ClusterABC100	PDA	1.23	1.63	33.20	11 → **44**	**43**	905	**< 1**	16,343	109,313
	MoMad	1.47	2.17	42.40	10 → 19	78	905	**< 1**	15,933	
	PD-TPLS	2.41	2.02	44.20	10 → 15	2,366	1,439	**< 1**	14,088	

Table 5.12 – Comparison between 2PIPLS/D, PDA, MoMad and PD-TPLS performance on Euclidean and clustered tri-objective instances of size 50 and 100.

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-3})	I_{R2}	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\lvert\overline{X}_{all}\rvert$	$\lvert\widetilde{Y}_{nd}\rvert$
	2PIPLS/D (C-LK)	**1.12**	**1.75**	**106**	**2 → 25**	293	2,640	**< 1**	20,929	
	PDA	1.22	1.91	114	1 → 18	**197**	2,668	3.0	20,347	
euclidABC300	PD-TPLS	2.35	2.23	126	2 → 22	7,052	2,667	**< 1**	18,146	456,433
	MoMad	2.15	3.56	188	0 → 1	399	3,092	2.9	18,451	
	2PIPLS/D (A-NMCS)	2.05	5.22	350	0 → 1	573	2,640	**< 1**	20,204	
	2PIPLS/D (C-LK)	**1.15**	**1.66**	**93.60**	**2 → 29**	318	2,715	**< 1**	21,670	
	PDA	1.29	1.96	109	1 → 19	**212**	2,716	3.0	20,938	
euclidDEF300	PD-TPLS	2.37	2.17	119	2 → 27	7,467	2,743	**< 1**	18,711	415,231
	MoMad	2.18	3.53	187	1 → 1	404	2,971	2.9	19,224	
	2PIPLS/D (A-NMCS)	2.06	5.12	340	0 → 2	514	2,715	**< 1**	20,990	
	2PIPLS/D (C-LK)	**1.15**	**1.75**	**102**	**2 → 27**	559	2,861	**< 1**	27,163	
	PDA	1.31	2.07	116	1 → 19	**261**	2,862	3.0	25,933	
euclidG-3-300	PD-TPLS	2.44	2.44	133	2 → 25	7,338	2,890	**< 1**	22,753	552,917
	MoMad	2.14	3.80	199	1 → 1	492	3,191	2.9	23,557	
	2PIPLS/D (A-NMCS)	2.06	5.26	354	0 → 2	745	2,862	1.0	26,423	
	2PIPLS/D (C-LK)	**1.26**	**1.50**	**89.90**	**2 → 23**	579	3,577	1.0	32,460	
	PDA	2.13	1.97	104	1 → 14	**367**	3,614	3.0	30,791	
ClusterABC300	PD-TPLS	3.18	2.28	120	2 → 21	7,788	3,633	**< 1**	26,297	657,700
	MoMad	3.96	3.94	185	0 → 1	730	3,614	2.9	28,064	
	2PIPLS/D (A-NMCS)	2.41	3.97	268	0 → 2	899	3,577	1.0	31,480	
	2PIPLS/D (C-LK)	**1.30**	**1.57**	**150**	1 → 27	668	5,805	2.0	32,974	
	PDA	2.39	2.11	182	1 → 13	**604**	5,807	7.8	30,330	
euclidABC500	PD-TPLS	2.96	2.65	226	**2 → 36**	12,352	5,805	**< 1**	23,381	523,721
	MoMad	4.10	5.48	444	0 → 1	865	5,980	7.7	19,872	
	2PIPLS/D (A-NMCS)	2.65	7.36	838	0 → 0	1,380	5,805	2.0	30,491	
	2PIPLS/D (C-LK)	**1.24**	**1.48**	**143**	1 → 26	641	5,894	1.4	23,791	
	PDA	2.10	1.90	174	1 → 12	**455**	5,896	7.8	22,226	
euclidDEF500	PD-TPLS	2.84	2.22	201	**2 → 40**	12,149	6,295	**< 1**	18,418	390,954
	MoMad	4.12	4.58	384	0 → 1	763	5,613	7.7	16,727	
	2PIPLS/D (A-NMCS)	2.41	7.17	823	0 → 0	1,028	5,894	2.0	22,324	
	2PIPLS/D (C-LK)	**1.26**	**1.68**	**170**	1 → 20	692	5,360	2.0	32,802	
	PDA	2.14	2.17	194	0 → 9	**598**	5,362	7.8	30,545	
euclidG-3-500	PD-TPLS	2.91	2.64	236	**2 → 30**	12,180	5,360	**< 1**	23,749	606,916
	MoMad	4.07	5.45	448	0 → 0	894	6,454	7.7	20,400	
	2PIPLS/D (A-NMCS)	2.58	7.26	835	0 → 0	1,387	5,360	2.0	30,739	
	2PIPLS/D (C-LK)	**1.22**	**1.02**	**114**	2 → 29	**826**	6,849	2.0	43,685	
	PDA	4.71	2.22	158	1 → 11	857	6,524	7.8	37,840	
ClusterABC500	PD-TPLS	4.28	2.48	198	**2 → 37**	12,004	6,522	**< 1**	28,053	657,038
	2PIPLS/D (A-NMCS)	3.11	5.19	635	0 → 0	1,648	6,849	2.0	41,059	
	MoMad	7.43	5.26	348	0 → 0	1,451	9,068	7.7	28,890	

Table 5.13 – Comparison between 2PIPLS/D, PDA, MoMad and PD-TPLS performance on Euclidean and clustered tri-objective instances of size 300 and 500.

Instance	Algorithm	I_ϵ (%)	I_H^- (10^{-3})	I_{R2}	Coverage (%)	Nb. of exam. sol. (10^6)	Time (s)	RAM (GB)	$\left\|\overline{X}_{all}\right\|$	$\left\|\widetilde{Y}_{nd}\right\|$
rdA-3-50	**2PIPLS/D (C-LK)**	**0.89**	**0.02**	**0.30**	88 → 97	303	212	**< 1**	103,095	114,490
	2PIPLS/D (A-NMCS)	1.08	0.05	0.40	**91 → 99.5**	384	212	**< 1**	106,163	
	PDA	1.08	0.16	2.31	56 → 92	**67**	210	**< 1**	74,105	
	MoMad	1.85	1.16	14.40	27 → 46	75	210	**< 1**	53,634	
	PD-TPLS	2.83	2.73	41.40	8 → 9	1,058	784	**< 1**	22,293	
rdB-3-50	**2PIPLS/D (C-LK)**	**0.88**	**0.02**	**0.35**	88 → 97	375	198	**< 1**	76,640	84,992
	2PIPLS/D (A-NMCS)	1.38	0.08	0.57	**92 → 99.8**	431	198	**< 1**	79,554	
	PDA	1.20	0.14	1.91	63 → 94	**52**	197	**< 1**	58,846	
	MoMad	2.14	1.22	16.30	27 → 47	54	197	**< 1**	39,589	
	PD-TPLS	3.01	2.89	46.70	8 → 9	1,047	943	**< 1**	17,478	
rdC-3-50	**2PIPLS/D (C-LK)**	**0.80**	**0.02**	**0.34**	88 → 97	320	197	**< 1**	93,044	102,866
	2PIPLS/D (A-NMCS)	1.00	0.05	0.67	**92 → 99.7**	428	197	**< 1**	96,437	
	PDA	1.19	0.14	2.06	60 → 93	**61**	194	**< 1**	69,584	
	MoMad	1.96	1.03	16.20	29 → 50	67	194	**< 1**	49,902	
	PD-TPLS	3.32	2.57	47.00	8 → 9	1,023	889	**< 1**	20,651	
rdABC100	**2PIPLS/D (C-LK)**	**1.20**	**0.97**	**32.90**	**13** → 29	223	883	**< 1**	38,274	287,363
	PDA	1.52	1.13	35.20	10 → 42	65	891	**< 1**	34,334	
	2PIPLS/D (A-NMCS)	3.09	2.03	58.80	9 → **50**	1,420	884	**< 1**	35,409	
	PD-TPLS	3.54	2.31	73.80	4 → 10	1,951	892	**< 1**	22,667	
	MoMad	3.90	2.81	73.60	4 → 11	**119**	979	**< 1**	26,218	
rdD-3-100	**2PIPLS/D (C-LK)**	**1.14**	**1.20**	**41.80**	**7** → 16	167	899	**< 1**	39,702	562,801
	PDA	1.67	1.37	45.2	5 → 24	**74**	899	**< 1**	35,255	
	2PIPLS/D (A-NMCS)	3.10	2.21	67.60	5 → **25**	1,213	900	**< 1**	37,246	
	PD-TPLS	3.53	2.51	80.10	2 → 5	1,926	909	**< 1**	24,382	
	MoMad	3.73	2.93	81.90	2 → 6	131	986	**< 1**	27,867	
rdABC300	**2PIPLS/D (C-LK)**	**2.42**	**1.15**	**128**	**1 → 27**	**523**	3,193	1.6	76,193	1,588,953
	PDA	4.34	1.46	139	1 → 20	578	3,225	3	68,526	
	PD-TPLS	7.48	2.37	190	1 → 14	5,948	3,225	**< 1**	49,981	
	MoMad	14.00	4.52	294	0 → 1	958	4,083	2.9	50,365	
	2PIPLS/D (A-NMCS)	17.80	7.19	671	0 → 0	1,262	3,206	1.1	67,884	
rdD-3-300	**2PIPLS/D (C-LK)**	**2.60**	**1.18**	**129**	**1 → 27**	**532**	3,047	1.6	72,931	1,518,306
	PDA	7.09	1.69	145	1 → 10	554	3,046	3	63,916	
	PD-TPLS	7.34	2.47	190	1 → 15	6,041	3,076	**< 1**	47,112	
	MoMad	14.00	4.78	294	0 → 1	892	4,021	2.9	47,777	
	2PIPLS/D (A-NMCS)	18.80	7.45	720	0 → 0	1,356	3,059	1.1	64,628	
rdABC500	**2PIPLS/D (C-LK)**	**4.70**	**1.03**	**179**	**2 → 36**	1,249	5,956	3.4	87,682	1,248,086
	PDA	14.90	1.96	197	1 → 25	1,205	5,957	7.8	71,506	
	PD-TPLS	10.40	2.63	288	1 → 20	10,229	5,955	**< 1**	43,666	
	MoMad	26.50	6.31	519	0 → 1	1,507	8,039	7.7	39,534	
	2PIPLS/D (A-NMCS)	27.40	10.10	1700	0 → 0	**989**	6,008	2.1	76,483	
rdD-3-500	**2PIPLS/D (C-LK)**	**4.98**	**1.05**	**184**	**1 → 26**	1,316	6,322	3.4	87,142	1,507,910
	PDA	14.80	2.12	206	1 → 17	**1,163**	6,322	7.8	70,890	
	PD-TPLS	11.00	2.64	297	1 → 14	10,087	6,320	**< 1**	43,813	
	MoMad	28.20	6.22	525	0 → 0	1,461	8,076	7.7	40,200	
	2PIPLS/D (A-NMCS)	26.40	10.30	1760	0 → 0	1,662	6,376	2.1	75,691	

Table 5.14 – Comparison between 2PIPLS/D, PDA, MoMad and PD-TPLS performance on random tri-objective instances.

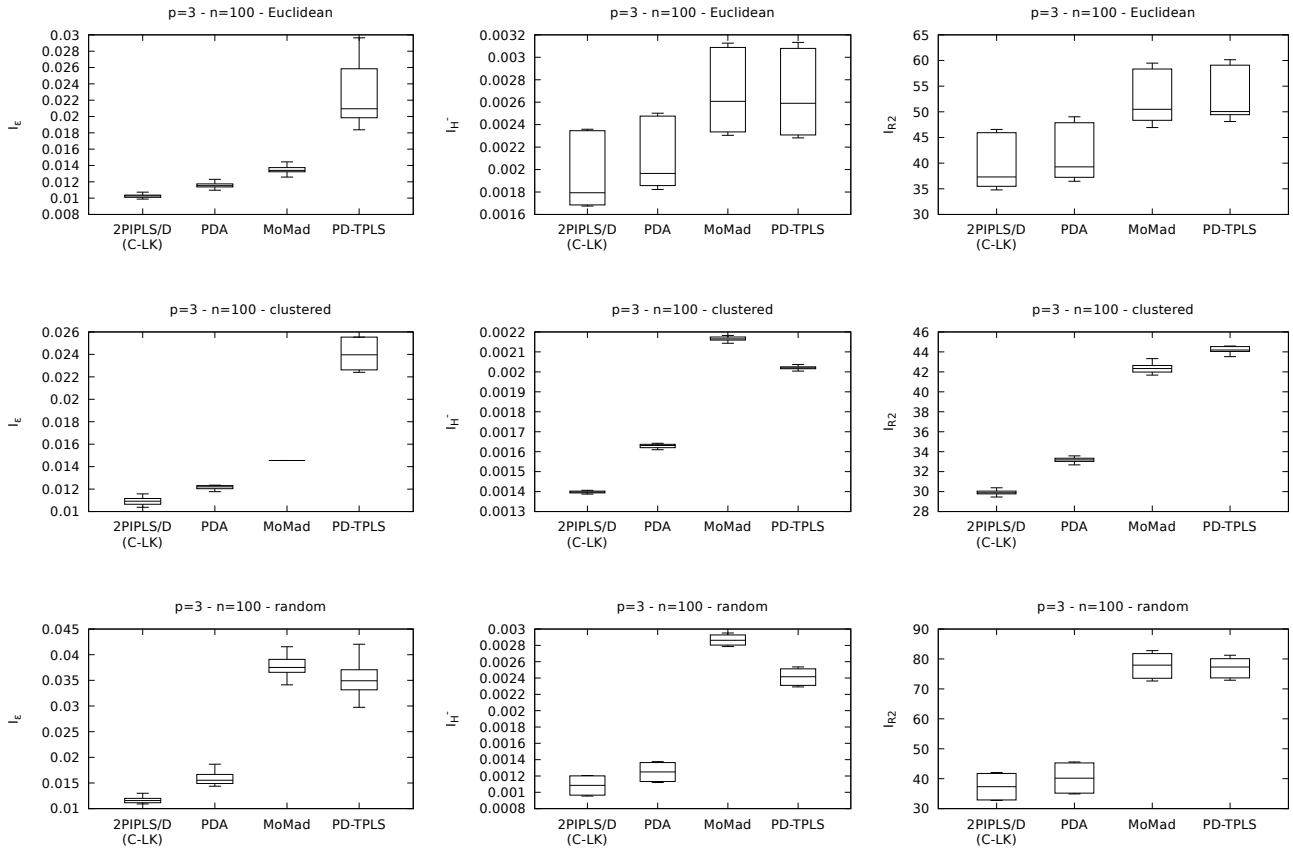


Figure 5.17 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, MoMad and PD-TPLS on tri-objective instances of size 100.

bi-objective instances of size 100 over 16). For comparison, AUGMECON2 is an exact method and therefore finds the (exact) non-dominated set of all bi-objectives instances of size 100 in slightly less than 30 hours in average, while 2PIPLS/D finds at least 99% of the non-dominated points in less than 4 minutes in average.

Over all quality indicators and all tested instances, 2PIPLS/D (with C-LK) outperforms both MoMad and PD-TPLS. Indeed, 2PIPLS/D (with C-LK) has a strictly better average performance than both MoMad and PD-TPLS and the Mann-Whitney test (cf. Section 2.1.4) confirms that 2PIPLS/D is strictly better than both MoMad and PD-TPLS. In a great majority of cases, for a given quality indicator on a given instance, the worst run of 2PIPLS/D (with C-LK) is better than the best run of MoMad or PD-TPLS. Moreover, the worst I_ϵ average value of 2PIPLS/D over all tested instances is (slightly less than) 5%, meaning that 2PIPLS/D is able to provide for all tested instances an ϵ-approximation of very good quality. In comparison, the best competitor (PD-TPLS) proposes a worst I_ϵ average value of 11%.

Over all quality indicators and tested instances, 2PIPLS/D (with C-LK) outperforms PDA, except for 3 instances (over 55). Indeed, the Mann-Whitney test indicates that 2PIPLS/D is strictly better than PDA over all instances except on: kroAB300 over I_ϵ, ClusterAB500 over I_ϵ and I_{R2} and kroAB750 over I_ϵ, for which 2PIPLS/D and PDA are considered as equivalent.

Besides, 2PIPLS/D with A-NMCS provides very poor performance, except on tri-objective instances of size lower than or equal to 100 (Tables 5.12 and 5.14). In particular, its performance worsen when the size of the instance grows. This confirms the fact that construction-based heuristics are generally not efficient on large-size MOTSP instances. Indeed, the same observation can be made for Ant Colony Optimization solver [López-Ibáñez and Stützle, 2012]. This also shows the **major influence of the SO optimizer on**

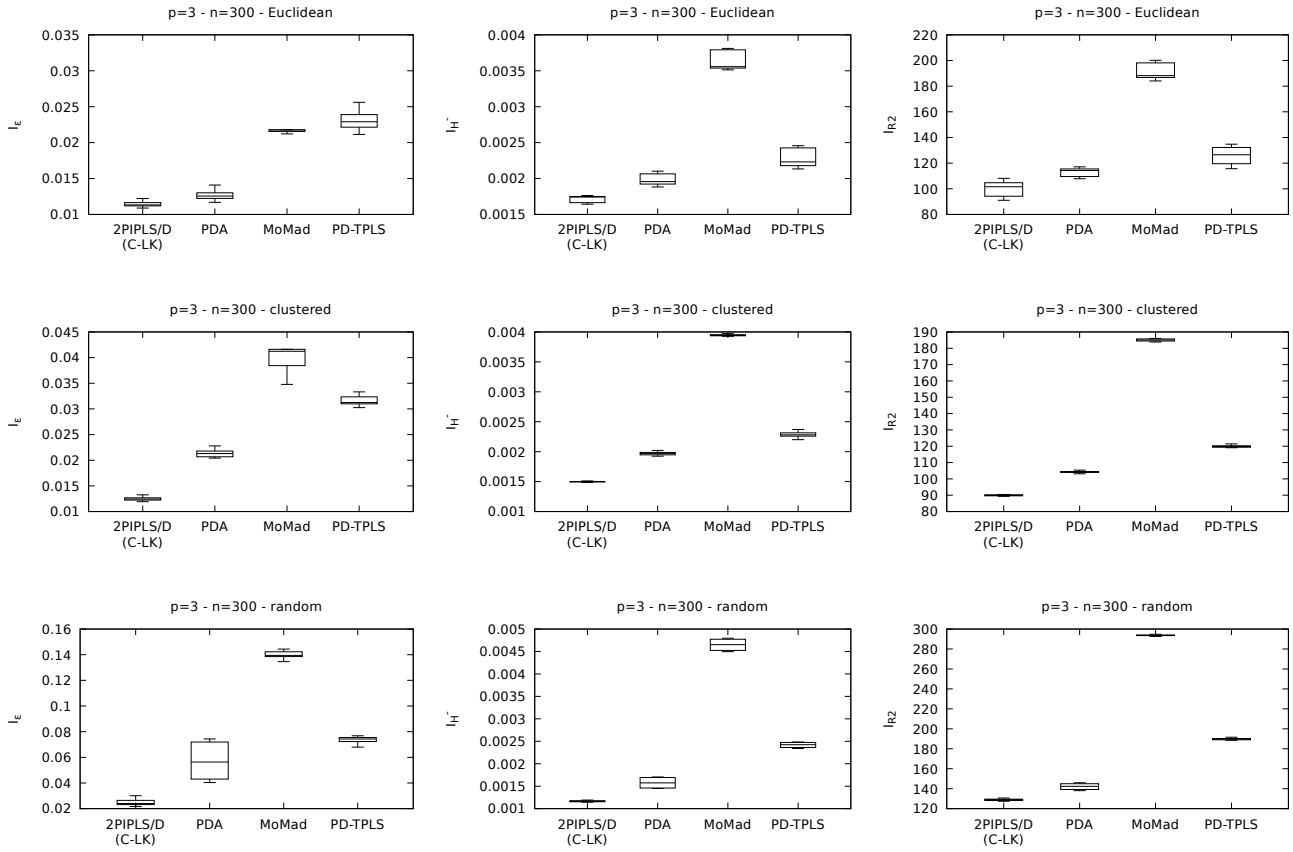


Figure 5.18 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, MoMad and PD-TPLS on tri-objective instances of size 300.

the performance of 2PIPLS/D.

It is interesting to see that all meta-heuristics (2PIPLS/D, PDA, MoMad, PD-TPLS) obtains much better results on non-random instances compared to random ones. As examples:

- On the bi-objective Euclidean instance kroAB500 (Table 5.10), 2PIPLS/D (with C-LK) obtains an average I_ϵ value of 0.06%, and MoMad 0.22%; on the other hand, on the bi-objective random instance rdAB500 (Table 5.11), 2PIPLS/D (with C-LK) obtains an average I_ϵ value of 1.22%, and MoMad 2.90%.
- On the tri-objective Euclidean instance euclidABC500 (Table 5.13), 2PIPLS/D (with C-LK) obtains an average I_ϵ value of 1.3%, and MoMad 4.1%; on the other hand, on the tri-objective random instance rdABC500 (Table 5.14), 2PIPLS/D (with C-LK) obtains an average I_ϵ value of 4.7%, and MoMad 26.5%.

This effect can be explained by the fact that C-LK (which is the solver used by all the meta-heuristics except 2PIPLS/D with A-NMCS) is particularly efficient on instances where the triangular inequality holds (Euclidean and clustered instances), but has difficulties on instances where the inequality does not hold (which is the case on random instances).

Note that the size of the approximate non-dominated set $\widetilde{Y}_{nd}$ strongly grows with the instance size n and the number of objectives p, reaching at least 1.1 million points for tri-objective random instances of size at least 300 (Table 5.14). For tri-objective instances of size $n \geq 100$, the approximation sets found by the methods

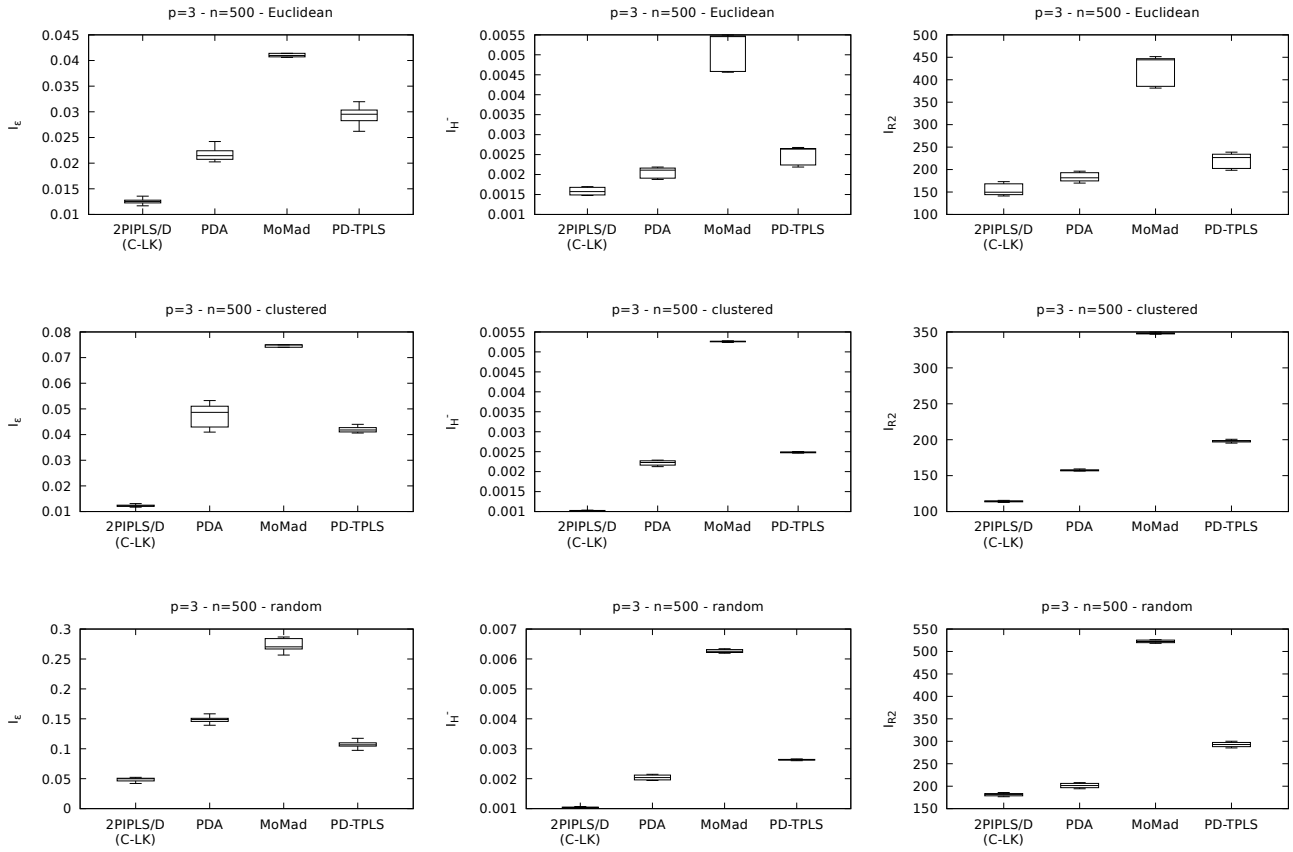
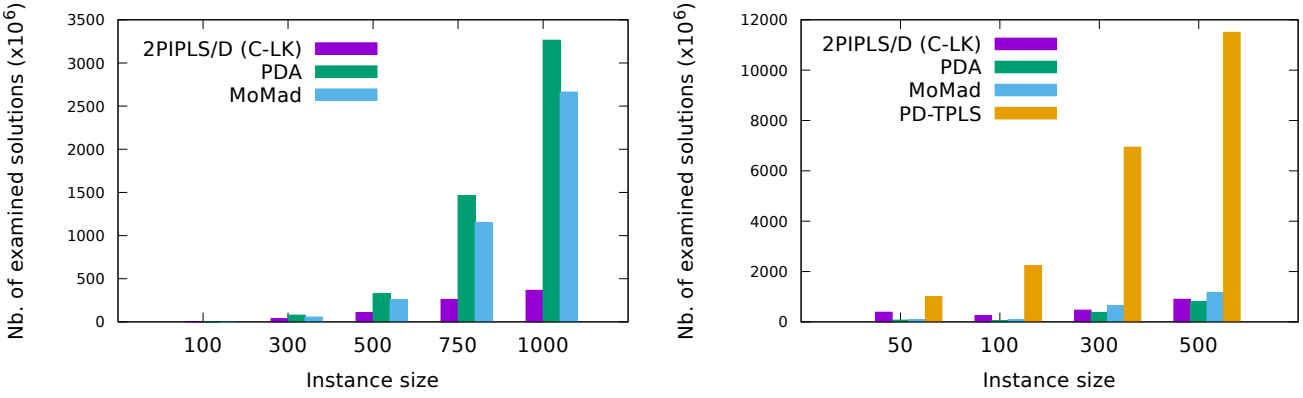


Figure 5.19 – I_ϵ (left), I_H^- (middle) and I_{R2} (right) performance comparison between 2PIPLS/D (with C-LK optimizer), PDA, MoMad and PD-TPLS on tri-objective instances of size 500.

are much smaller than $\widetilde{Y}_{nd}$. This is due to the use of ϵ-archive with $\epsilon = 1\%$, which bounds the size of the approximation while preserving a good distribution of the points in the objective space. Indeed, results of 2PIPLS/D (with C-LK) are still of good quality. One can consider the instance rdD-3-500 (Table 5.14), for which 2PIPLS/D (with C-LK) has its worst I_ϵ results. On this instance, the approximations generated by 2PIPLS/D (with C-LK) is worse than $\widetilde{Y}_{nd}$ by only a factor of 4.98% (in average), indicating that 2PIPLS/D (with C-LK) generates well dispersed approximation sets over $\widetilde{Y}_{nd}$.

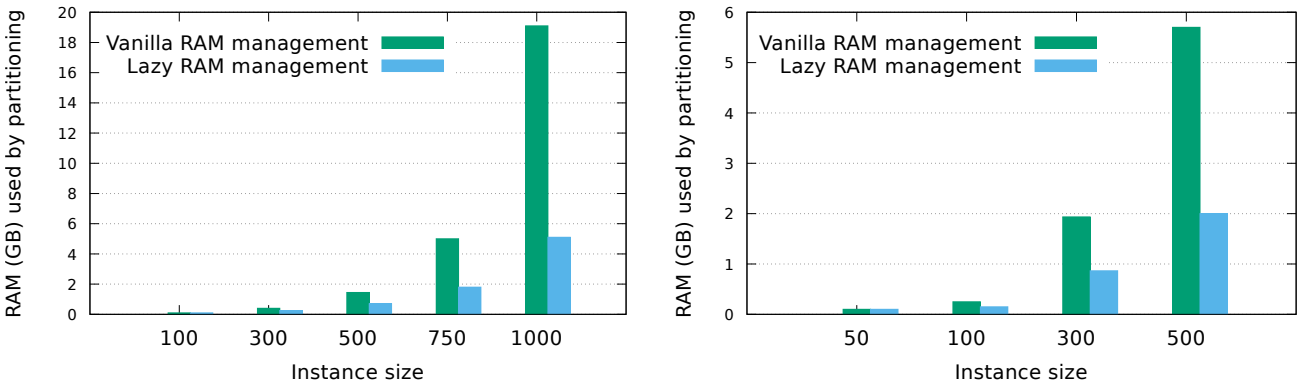
Figure 5.20 highlights the number of solutions examined by 2PIPLS/D (with C-LK), PDA, MoMad and PD-TPLS. First, it is important to indicate that PLS-VND provides more than 99% of the solutions examined by 2PIPLS/D, the rest being provided by the SO optimizer. On bi-objective instances, 2PIPLS/D (with C-LK) examines much less solutions than its competitors, and the gap grows significantly as the instance size grows. This observation is particularly interesting as 2PIPLS/D uses a 3-exchange as PLS-VND neighborhood, while its competitors only use a 2-exchange neighborhood in their PLS and obtain worse results, which shows the efficiency of the partitioning system of P-PLS.

On tri-objective instances, 2PIPLS/D (with C-LK) does not examine less solutions than PDA and MoMad, however the number of solutions examined by 2PIPLS/D increases slower than for its competitors when the instance size grows. On the other hand, it is interesting to note that while PD-TPLS runs a single iteration of PLS with a 3-opt, it already examines much more solutions than the other methods. This is due to the fact that the candidate edge list used by PD-TPLS is not adapted, contrary to the one used by 2PIPLS/D. This remark points out the difficulty of designing an efficient PLS using a neighborhood larger than 2-exchange neighborhood for MOTSP.



(a) $p = 2$ (b) $p = 3$

Figure 5.20 – Comparison of number of examined solutions between 2PIPLS/D (with C-LK) and its competitors.



(a) $p = 2$ (b) $p = 3$

Figure 5.21 – Comparison of RAM utilization of partitioning between *vanilla RAM management* and *lazy RAM management*.

A final comment can be made on RAM usage of 2PIPLS/D. Despite the fact that the partitioning brought by P-PLS is intended to increase drastically the RAM usage compared to no partitioning (as mentioned in Section 5.2.4), 2PIPLS/D generally consumes twice less RAM than its competitors. This is due to a trick we propose, the so-called *lazy RAM management*, which consists in loading on memory only the edges effectively used during PLS-VND. Figure 5.21 shows the differences in terms of RAM consumption of partitioning, between vanilla RAM management and lazy RAM management.

5.6 Conclusion

In this chapter, we first proposed an implementation of the different components of 2PIPLS/D, in particular we proposed to use C-LK and A-NMCS as SO optimizers, and suggested an implementation of the neighborhood restriction structure for P-PLS. Then, we presented an empirical evidence of global convexity on MOTSP in order to legitimate the use of the P-PLS partitioning system. We found that (potentially) efficient solutions are effectively concentrated in decision space and that (potentially) efficient solutions neighbors in the objective space tend to be also neighbors in the decision space. After a sensitivity analysis of 2PIPLS/D on its parameters, we ranked the different parameters in function of their influence on the performance of the method. We found that 2PIPLS/D is sensitive to the number of weighted-sum problems solved at initialization during A-MDW, and the maximum authorized size of a part at P-PLS initialization. In particular, we have noted that P-PLS reduces drastically the number of solutions examined by PLS-VND (-98% in average over the instances tested) thanks to its partitioning system. In addition, memorizing the solutions during the run of a SO solver appears to be highly effective, for both C-LK and A-NMCS. In addition, 2PIPLS/D seems moderately sensitive to the maximum neighborhood structure size.

On the other hand, 2PIPLS/D seems not to be particularly sensitive to data perturbation and to the different exploration strategies and mods used by PLS-VND. Finally, we compared 2 versions of 2PIPLS/D (either with improved C-LK or A-NMCS) to the best current methods on a large benchmark of bi-objective and tri-objective instances.

2PIPLS/D (with C-LK) obtains good results and found the exact non-dominated set of 7 bi-objective instances of size 100 over 20 runs, so that 2PIPLS/D is, to our knowledge, the first meta-heuristic able to find non-dominated set of MOTSP instances of such size. 2PIPLS/D (with C-LK) outperforms MoMad and PD-TPLS on all tested instances; 2PIPLS/D also outperforms PDA on 95% of the tested instances and has equivalent performance on the 5% remaining instances.

Finally, 2PIPLS/D with C-LK outperforms in a great majority of cases 2PIPLS/D with A-NMCS, which strengthens the fact that 2PIPLS/D performance highly depends on the efficiency of the SO solver.

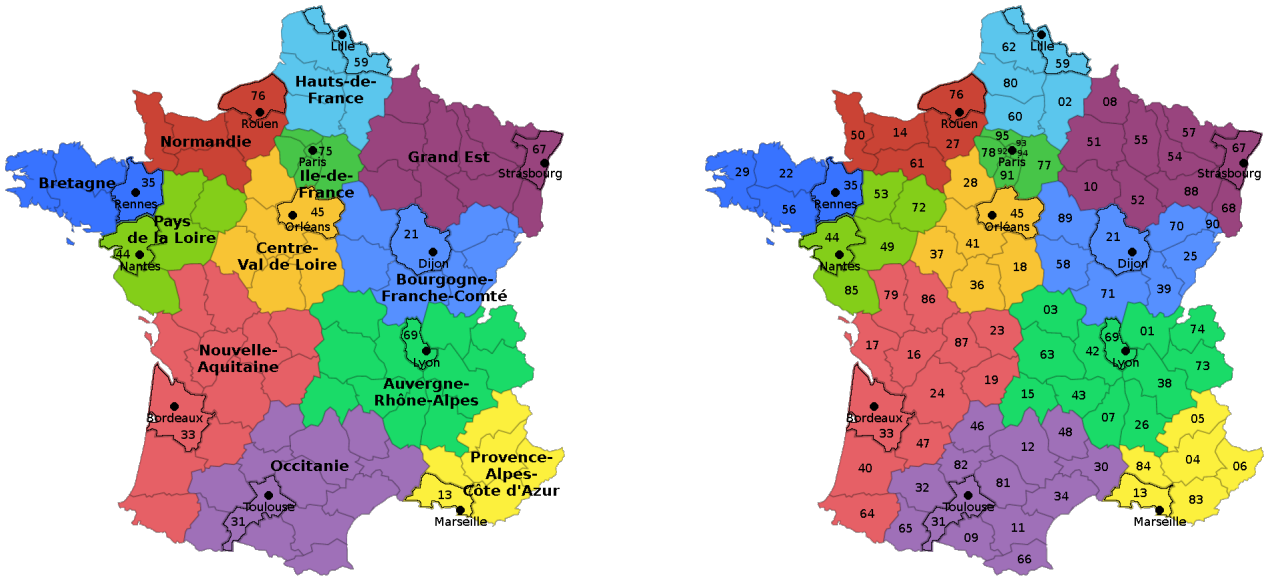
Concerning perspectives, it could be interesting to implement neighborhood structures for PLS-VND of larger size than 3-exchange, or even better, to propose variable k-exchange moves like in Lin-Kernighan. However, we can enumerate two main limitations about this perspective. Firstly, as already mentioned in Section 2.3, the techniques which made LS-based methods so powerful on TSP, like don't-look-bits or candidate edge lists, are not as far as efficient in MOTSP. Secondly, it seems that the global perturbation step of 2PIPLS/D is sufficiently efficient to compensate a small PLS-VND neighborhood structure size.

A second perspective is the application of 2PIPLS/D on larger instances, both in terms of sizes and number of objectives. According to us, the main limitation concerning this proposal is that 2PIPLS/D, like any method based on the 2-phase PLS framework, are highly sensitive to the SO optimizer employed to (re-)start PLS. However, Lin-Kernighan-based SO optimizers seems to be the best option we currently have, but are not so efficient on instances in which the triangular inequality does not hold. Therefore, we expect difficulties for this type of methods on random instances, like bi-objective or tri-objective of size 1000, or on large-scale four-objective instances.

Chapter 6

Application of 2PIPLS/D to MOFRMP

This chapter introduces a new five-objective real-world problem called MO French Regions Mapping Problem (MOFRMP). It is related to the recent territorial reform of French regions which resulted in the reassignment of departments to new larger regions. The aim of this problem is to find a map (i.e. an assignment of departments to regions) optimizing five objectives, which evaluate the strength of interactions between the departments assigned to the same region, as well as the economical and demographic weights of each region. We propose to apply 2PIPLS/D to this problem. We first present an implementation of 2PIPLS/D, then we study the existence of global convexity on MOFRMP. After a sensitivity analysis on its parameters, we compare several versions of 2PIPLS/D with different parameter settings and discuss about the results.



(a) The regions of Metropolitan France and their respective capital.

(b) The departments of Metropolitan France.

Figure 6.1 – Two complementary maps of Metropolitan France (excluding Corsica island) illustrating the 12 regions, their respective capital and departments.

Introduction

This chapter introduces the MO French Regions Mapping Problem (MOFRMP), a new five-objective real-world problem falling within the scope of the recent territorial reform of French regions which resulted in the reassignment of departments to new larger regions. Firstly, Section 6.1 presents MOFRMP, then Section 6.2 proposes an implementation of 2PIPLS/D and Section 6.3 studies the global convexity on the problem. Computational experiments are performed on Section 6.4, and the results are discussed in Section 6.4.2. Finally, we summarize the chapter and propose some perspectives concerning MOFRMP.

6.1 Presentation of MOFRMP

Metropolitan France is the part of France located in Europe. It is composed of 96 administrative divisions, called *departments*. Each department belongs to a higher-level administrative division called *region*.

In January 1st, 2016, a new administrative map of Metropolitan France has taken effect (the so-called *current map*), grouping departments into larger regions and reducing the number of regions from 22 down to 13. Figure 6.1 illustrates the 12 regions (excluding Corsica island), their respective capital and departments, and Table 6.1 lists them.

This historical territorial reform takes place within a larger process called *decentralization*, consisting in transferring administrative powers from the French State to local entities distinct from it, making the regions more autonomous from Paris[1] and registered in the French Constitution since 2003[2].

[1] http://www.vie-publique.fr/decouverte-institutions/institutions/collectivites-territoriales/principes-collectivites-territoriales/qu-est-ce-que-decentralisation.html

[2] http://www.assemblee-nationale.fr/connaissance/constitution.asp

Region	Capital	Departments
Provence-Alpes-Côte d'Azur	Marseille (13)	Alpes (04), Hautes-Alpes (05), Alpes-Maritimes (06), Bouches-du-Rhône (13), Var (83), Vaucluse (84)
Bourgogne-Franche-Comté	Dijon (21)	Côte-d'Or (21), Doubs (25), Jura (39), Nièvre (58), Haute-Saône (70), Saône-et-Loire (71), Yonne (89), Territoire-de-Belfort (90)
Occitanie	Toulouse (31)	Ariège (9), Aude (11), Aveyron (12), Gard (30), Haute-Garonne (31), Gers (32), Hérault (34), Lot (46), Lozère (48), Hautes-Pyrénées (65), Pyrénées-Orientales (66), Tarn (81), Tarn-et-Garonne (82)
Nouvelle-Aquitaine	Bordeaux (33)	Charente (16), Charente-Maritime (17), Corrèze (19), Creuse (23), Dordogne (24), Gironde (33), Landes (40), Lot-et-Garonne (47), Pyrénées-Atlantiques (64), Deux-Sèvres (79), Vienne (86), Haute-Vienne (87)
Bretagne	Rennes (35)	Côtes (22), Finistère (29), Ille-et-Vilaine (35), Morbihan (56)
Pays de la Loire	Nantes (44)	Loire-Atlantique (44), Maine-et-Loire (49), Mayenne (53), Sarthe (72), Vendée (85)
Centre-Val de Loire	Orléans (45)	Cher (18), Eure-et-Loir (28), Indre (36), Indre-et-Loire (37), Loir-et-Cher (41), Loiret (45)
Hauts-de-France	Lille (59)	Aisne (02), Nord (59), Oise (60), Pas-de-Calais (62), Somme (80)
Grand Est	Starsbourg (67)	Ardennes (08), Aube (10), Marne (51), Haute-Marne (52), Meurthe-et-Moselle (54), Meuse (55), Moselle (57), Bas-Rhin (67), Haut-Rhin (68), Vosges (88)
Auvergne-Rhône-Alpes	Lyon (69)	Ain (01), Allier (03), Ardèche (07), Cantal (15), Drôme (26), Isère (38), Loire (42), Haute-Loire (43), Puy-de-Dôme (63), Rhône (69), Savoie (73), Haute-Savoie (74)
Île-de-France	Paris (75)	Paris (75), Seine-et-Marne (77), Yvelines (78), Essonne (91), Hauts-de-Seine (92), Seine-Saint-Denis (93), Val-de-Marne (94), Val-d'Oise (95)
Normandie	Rouen (76)	Calvados (14), Eure (27), Manche (50), Orne (61), Seine-Maritime (76)

Table 6.1 – The 12 regions of Metropolitan France (excluding Corsica island) with their respective capital and departments.

Officially, two main reasons drive this regrouping of the departments into larger regions[3][4]:

1. Reducing the administrative complexity and thus the surcharge in terms of operating cost, the so-called *"millefeuille administratif"* in French, for both, French and foreign, individuals and companies, by making economies of scale and simplify the administrative procedures.
2. Increasing the exposure, the attractiveness and competitiveness of the French regions within both European and international spheres by increasing their economical power.

However, it has been recognized[5] that some departments are attracted to regions other than their current region of assignment. The current map of regions can therefore be potentially improved. But further research is needed to address this problem.

This is why from 2016 to 2019, departments attracted by another region than their current region of assignment have the possibility to launch a procedure to change to another region if both regions give their support[6].

The different criteria that assess the attraction of a department to a given region are complex and of multiple natures: economic, financial, social, cultural, environmental, but we will only consider here the criteria proposed by a recent economic analysis of this problem [Amabile et al., 2015][7] published in French by France Stratégie and initiated by A. Amabile (France Stratégie), C. Bernard (France Stratégie) and A. Épaulard (France Stratégie and Laboratoire d'Économie de Dauphine) and implemented by T. Cazenave. France Stratégie[8] is an institution attached to the government, whose objective is to contribute to determine the main orientations for the future of the nation and the medium-term and long-term objectives of its economic, social, cultural and environmental development, as well as the preparation of reforms.

This analysis identified 5 objectives (or criteria) divided in two groups, to evaluate the assignment of departments to a given region. The first group of objectives (the so-called *travel times to regional capitals*, *inter-departmental commuting intensities*, *inter-departmental shareholding intensities* objectives, described below) proposes to assign to the same region, departments that have strong economic links in order to ensure a greater regional economic coherence. The second group of objectives (the so-called *regional GDP weights*, *regional population weights* objectives, described below) aims at building regions whose economic and demographic roles are not too heterogeneous, to ensure that each region has sufficient resources to carry out economic development policies and to cope with the fixed costs of regional administration.

From this analysis, we introduce the subsequent new MOCO problem, called MO French Regional Mapping Problem (MOFRMP). It consists in building a so-called *map (of regions)*, i.e. an assignment of each department of Metropolitan France (except Corsica island -departments 2A and 2B- forming the *Corsican Territorial Community*) to a region, such that two constraints are satisfied:

- the *constraint of fixed capitals*: the departments containing a regional capital are fixed (departments 13, 21, 31, 33, 35, 44, 45, 59, 67, 69, 75, 76);
- and the *constraint of regional continuity*: a region forms a block of departments such that there is no discontinuity of the regional territory (in other words: there is no departments isolated from their regional capital).

[3] https://www.interieur.gouv.fr/Archives/Archives-sous-sites/Reforme-des-collectivites-territoriales/La-reforme/Questions-les-plus-frequentes
[4] http://www.gouvernement.fr/action/la-reforme-territoriale
[5] http://www.strategie.gouv.fr/publications/reforme-territoriale-coherence-economique-regionale
[6] http://www.strategie.gouv.fr/point-de-vue/repenser-decoupage-regional
[7] http://www.strategie.gouv.fr/point-de-vue/repenser-decoupage-regional,
http://www.strategie.gouv.fr/publications/une-evaluation-de-coherence-economique-interne-regions,
http://www.strategie.gouv.fr/sites/strategie.gouv.fr/files/atoms/files/dt_2015-01_rrgions_cbae_rev_selda_final.pdf
[8] http://www.strategie.gouv.fr/

while optimizing the 5 objectives previously mentioned: (1) *travel times to regional capitals*, (2) *inter-departmental commuting intensities*, (3) *inter-departmental shareholding intensities,* (4) *regional GDP weights*, (5) *regional population weights*.

More formally, a feasible solution (a *map*) is represented by a vector $x = (r(1), ..., r(19), r(21), ..., r(95)) \in X$, where $r(i) \in R$ represents the assignment of department i to the region $r(i)$ (identified by the code of the department containing its capital), where $R = \{13, 21, 31, 33, 35, 44, 45, 59, 67, 69, 75, 76\}$ is the set of departments containing a regional capital ($|R| = 12$), for each $i \in D$ where $D = \{1, ..., 19, 21, ..., 95\}$ is the set of departments ($|D| = 94$), while satisfying:

- the constraint of fixed capitals: $r(j) = j$ for each department $j \in R$;
- and the constraint of regional continuity: for any department $d_1 \in D$ such that $r(d_1) = j$, there is a path of departments $(d_1, ..., d_K, j)$ such that for each $k = 1, ..., K$, $r(d_k) = j$ and departments d_k and d_{k+1} are adjacent (assuming $d_{K+1} = j$).

MOFRMP is defined by (1.1) with $p = 5$ and such that X is the set of maps:

$$\begin{cases} \min f(x) = (f_1(x), ..., f_5(x)) \\ \text{subject to } x \in X \end{cases}$$

where the objective function f is composed of the five following single-objective functions:

1) The sum of **travel times to regional capitals** (to be minimized):

For each region, the time to travel by car from a department to its regional capital is considered as an indication of the ability of local government to get information about demand, preferences and production costs for goods and services that are provided by the local government. A higher travel time between a department and the regional capital reduces the informational advantage of the local government. This loss of efficiency is supposed to be quadratic with travel time. For each region is computed the population[9]-weighted average of the squared travel time from the capital of the department to the regional capital. After summing over all regions, we obtain the following formula:

$$f_1(x) = \sum_{r \in R} \sum_{d \in D_r(x)} pop_d \times (time_{d,r})^2$$

where $D_r(x)$ is the set of departments assigned to the region $r \in R$ in the map x, pop_d is the population of department $d \in D$, $time_{d,r}$ is the average time to travel from the department d to the capital of the region r.

2) The sum of **inter-departmental commuting intensities** (to be minimized):

This objective uses an indicator[10] proposed in [Amabile et al., 2015] which measures the intensity of commuting between two departments of the same region. The indicator value ranges from 0.0 to 1.0 (lower is better), such that a value of 1.0 means no people travel between the two departments. For each region, the second objective is computed as the average indicator value over all ordered pairs of departments of the region. The aim is to favor the grouping in the same region of departments between which these commuting links are strong. After summing over all regions, we obtain the following formula:

$$f_2(x) = \sum_{r \in R} \frac{\sum_{d \in D_r(x)} \sum_{d' \in D_r(x) : d' > d} com_{d,d'}}{|D_r(x)| \left(|D_r(x)| - 1\right)/2}$$

[9] based on the Insee 2011 database on demography

[10] based on the Insee 2010 database on commuting

where $com_{d,d'}$ is the intensity indicator value of commuting between departments d and d'.

3) The sum of **inter-departmental shareholding intensities** (to be minimized):

This objective uses an indicator[11] proposed in [Amabile et al., 2015] which measures the intensity of financial exchanges between two departments of the same region, measured by the shareholding of a department in the companies of another department. The indicator value ranges from 0.0 to 1.0 (lower is better), such that a value of 1.0 means no shareholder link between two departments. More the shareholders of a department control a high percentage of jobs in the other department, the lower the indicator will be. For each region, the third objective is computed as the average indicator value over all ordered pairs of departments of the region. The aim is to favor the grouping in the same region of departments between which these financial links are strong. After summing over all regions, we obtain the following formula:

$$f_3(x) = \sum_{r \in R} \frac{\sum\limits_{d \in D_r(x)} \sum\limits_{d' \in D_r(x) \,:\, d' > d} fin_{d,d'}}{|D_r(x)| \left(|D_r(x)| - 1\right)/2}$$

where $fin_{d,d'}$ is the intensity indicator value of shareholding between department d and department d'.

4) The sum of **regional GDP weights** (to be minimized):

This objective uses an economic indicator[12] proposed in [Amabile et al., 2015] which measures the GDP weight of a department. The indicator value of a department consists in the standardized GDP score of the department. For each region, the fourth objective considers the inverse squared sum of indicator value over all departments. The aim of this objective is to build regions whose economic roles are not too heterogeneous. After summing over all regions, we obtain the following formula:

$$f_4(x) = \sum_{r \in R} \left(\frac{1}{\sum\limits_{d \in D_r(x)} gdp_d} \right)^2$$

where gdp_d is the GDP score of the department d.

5) The sum of **regional population weights** (to be minimized):

This objective uses a demographic indicator[13] proposed in [Amabile et al., 2015] which measures the population weight of a department. The indicator value of a department consists in the standardized population score of the department. For each region, the fifth objective considers the inverse squared sum of indicator value over all departments. The aim of this objective is to build regions whose demographic roles are not too heterogeneous. After summing over all regions, we obtain the following formula:

$$f_5(x) = \sum_{r \in R} \left(\frac{1}{\sum\limits_{d \in D_r(x)} pop_d} \right)^2$$

where pop_d is the population score of the department d.

An important remark can be made on [Cazenave et al., 2016]. In this paper, Cazenave, Bernard and Épaulard propose to optimize the so-called *original FRMP*, which follows another formulation of the FRMP than the one we present. While the constraints of the problem are exactly the same, the problem is considered as a *single objective problem* such that the function to optimize is a weighted sum of the objectives, and the authors solve the problem with different weights. Furthermore, the objectives of the problem are different as the authors consider only the four following objectives:

[11] based on the Diane database: www.bvdinfo.com/fr-fr/our-products/company-information/national/diane
[12] based on the Insee 2005 database on GDP
[13] based on the Insee 2011 database on demography

- the three first objectives we consider: *(1) travel times to regional capitals*, *(2) inter-departmental commuting intensities* and *(3) inter-departmental shareholding intensities*;
- the fourth objective, called *political preferences heterogeneity*, is different from the ones we consider and consists in minimizing the heterogeneity of political preferences (computed with the results of presidential elections) between the departments assigned to the same region.

In our MO formulation, we do not consider this additional objective (the political preferences) because after some discussions with A. Épaulard, it appears to us that this objective was probably not consistent with the problem.

On the contrary, according to us, the two last objectives we consider in MOFRMP (and not considered in the original FRMP), the *(4) regional GDP weights* and *(5) regional population weights*, are essential. In fact, a region is responsible for economic development policies and essential public services -mainly transports, education, employment- and has to cope with the fixed costs of administration. So a region has important financial needs. Taxes levied to the population (local taxes and property tax among others) and companies (property tax) allow to finance a part of the needs[14]. Therefore it appears important to consider the *(5) population weights* as an objective, in order to build regions with a demographic disparity as low as possible. Furthermore, in the draft finance bill for 2017[15], the State-financed Regional Operating Fund (Dotation Globale de Fonctionnement, in French) must disappear and be replaced by the transfer of a fraction of the Value-added tax (VAT - TVA in French). Given that VAT is globally proportional to the GDP, the *(4) GDP weights* objective seems also important to be optimized, in order to obtain regions with economic roles not too heterogeneous. Finally, it is important to know that France suffers from massive unemployment for many years now, its reduction becoming a National priority for the successive governments. The regions are therefore in competition to attract companies, and in order to encourage them to establish at home, the regions are investing considerable sums of money through regional investment funds (e.g. Normandie Participations[16] for the Normandie region) to provide companies with an access to substantial financial or material support for business development.

Consequently, it appear to us that regions with GDP and population disparities as low as possible, and thus a low disparity in terms of income between the regions, could make the competition as fair as possible to reduce unemployment and keep good public services in all regions in an homogeneous way; making essential to consider the *(4) regional GDP weights* and the *(5) regional population weights* objectives in MOFRMP.

6.2 Implementation of 2PIPLS/D to address MOFRMP

As a component-wise method, 2PIPLS/D contains several algorithmic components to implement for MOFRMP: the elementary LS move, the neighborhood structure utilized by PLS-VND, the neighborhood restriction structure used in P-PLS, the perturbation move and the SO optimizer. Given the high number of objectives of the problem, the non-dominated set is certainly extremely large, consequently the archives of 2PIPLS/D are managed as ϵ-archives with SANDRA and a tolerance for dominance relations $\epsilon = 1\%$ is used. Let us present more precisely the implementation of 2PIPLS/D.

6.2.1 Local Search-oriented components

The most essential algorithmic component is the elementary LS move, the so-called ***k-department move***. It consists, from a given feasible map, to select k departments and for each department, choosing a *target*

[14] https://www.collectivites-locales.gouv.fr/files/files/statistiques/brochures/bpr_2016.pdf
[15] http://www.caissedesdepotsdesterritoires.fr/cs/ContentServer?pagename=Localtis/LOCActu/ArticleActualite&jid=1250271788545&cid=1250271787054&nl=1
[16] https://adnormandie.fr/normandie-participations/

region different from the region it belongs to, such that the modified map is still feasible. The parameter k is called the *size of the move*.

The **neighborhood structure for PLS-VND** is a k-department neighborhood and we consider different maximum neighborhood structure sizes $k = 1, 2, 3$. Obviously, PLS-VND with $k = 1$ corresponds to PLS.

The **neighborhood restriction structure**, employed in the parts of P-PLS, consists in computing for each part the set of *locked departments*, i.e. the set of departments assigned to the same region in *all* solutions in the archives of the sources of this part, and forbidding during PLS-VND the reassignment of any locked department.

Concerning the **perturbation move** used at perturbation step (second phase), internal tests have shown that using a 10-department move obtains stable and good results.

6.2.2 SO optimizer

The SO optimizer is used in A-MDW and during the perturbation step (second phase). A selection of 3 SO optimizers has been made.

Iterated Local Search (ILS)

We have implemented an ILS with Variable Neighborhood Descent (VND) and able to memorize all incomparable locally minimum solutions generated (as suggested in Section 4.1). Except the starting solution, ILS has three input parameters whose values have been set through internal tests: a number of iterations set to 100, a VND using k-department neighborhood with a maximum neighborhood structure size of 2, and a 10-department move as perturbation move.

A-NMCS

Cazenave has implemented an NMCS optimizing a weighted sum of MOFRMP objectives. We have converted it into A-NMCS by:

- memorizing all incomparable generated maps;
- managing the optimization of a data perturbed weighted sum problem;
- taking an initial map and its associated sequence as input parameters at the highest nested level.

In A-NMCS, the construction of a sequence consists in building a map by iteratively assigning a department to a region. More precisely, let a *feasible partial map* be a map such that a number of departments are not yet assigned to a region, but such that the constraint of fixed capitals is satisfied and the constraint of regional continuity can still be satisfied if the partial map is completed. An action consists in assigning to a region the department with the lowest code among all unassigned departments, while the obtained map is still feasible. Therefore, a (partial) sequence consists in an ordered list of assignments.

At a level of recursion greater than or equal to 1, A-NMCS tries all available actions. In random simulations, actions are selected uniformly at random. We found that A-NMCS with a level of recursion equal to 1 provides poor results, while A-NMCS of level 3 conducts very long runs, thus we choose level 2.

Iterative Deepening A* (IDA*)

IDA* [Korf, 1985] is an exact solver based on the Branch-and-Bound methodology using a problem-specific admissible heuristic to bound search sub-spaces. Cazenave has implemented an IDA* for the studies [Amabile

et al., 2015, Cazenave et al., 2016][17] (in French). This method is able to optimize a weighted sum of all the six objectives previously described: the five objectives of MOFRMP, and the *political preferences heterogeneity* objective.

It has been employed to solve the original FRMP (previously introduced). Efficient admissible heuristics have been found and enable IDA* to solve any tested weighted sum aggregation of the four objectives of the original FRMP in few seconds. On the other hand, the admissible heuristics for the two remaining objectives (both considered in MOFRMP): the *(4) regional GDP weights* and the *(5) regional population weights* objectives, have not already been tested.

6.3 Study of global convexity on MOFRMP

As no exact method has been implemented for MOFRMP, we do not know its efficient set. We have merged all approximation sets generated during experiments and internal tests to produce an approximation of the efficient set. The study presented in this section is based on this approximation set. Let the *(potentially) efficient solutions* be the solutions of this approximation set. Note that this work might be biased by the efficiency of 2PIPLS/D on this problem, despite the large number of 2PIPLS/D runs and settings tested.

The aim of this section is to empirically study global convexity on MOFRMP. Firstly, we check if (potentially) efficient solutions which are neighbors in the objective space tend to be also neighbors in the decision space. Secondly, we analyze the concentration of (potentially) efficient solutions in the decision space.

Are (potentially) efficient solutions neighbors in the objective space also neighbors in the decision space?

To check if (potentially) efficient solutions which are neighbors in the objective space tend also to be neighbors in the decision space, we follow the same process as in Section 5.3.2 for MOTSP and compute the Spearman correlation between the distances of (potentially) efficient solutions in the decision space, and the distances of their images in the objective space. We obtain a Spearman correlation value equal to 0.989202. Given a significance level of 1%, the p-value related to the Spearman correlation score is lower than 10^{-5}, meaning that the Spearman correlation score is statistically significant. This means that **the distances of solutions and the distances of their images in the objective space are almost perfectly positively correlated**.

Concentration of (potentially) efficient solutions in the decision space

The assignment of a department $d \in D$ to a region $r \in R$ represents the basic element composing the solutions of MOFRMP. We have computed the proportion of all possible assignments (department, region) of MOFRMP used in (potentially) efficient solutions, in order to see how much they are concentrated in the decision space. In fact, the (potentially) efficient solutions contain 52% of all possible assignments (in average each department is assigned to 6.2 regions over 12). This corresponds to a large fraction of all possible assignments, in comparison of the 11% of edges used in average on MOTSP instances tested in Section 5.3.1. Thus we can say that **the (potentially) efficient solutions are not particularly concentrated in the decision space**. As a consequence, the intersection between solutions (i.e. the number of departments in $D \setminus R$ assigned to the same region in all these solutions) neighbors in the objective space decreases exponentially fast as the distance (in the objective space) grows, as illustrated in Figure 6.2. In particular,

[17] http://www.strategie.gouv.fr/point-de-vue/repenser-decoupage-regional,
http://www.strategie.gouv.fr/publications/une-evaluation-de-coherence-economique-interne-regions,
http://www.strategie.gouv.fr/sites/strategie.gouv.fr/files/atoms/files/dt_2015-01_rrgions_cbae_rev_selda_final.pdf

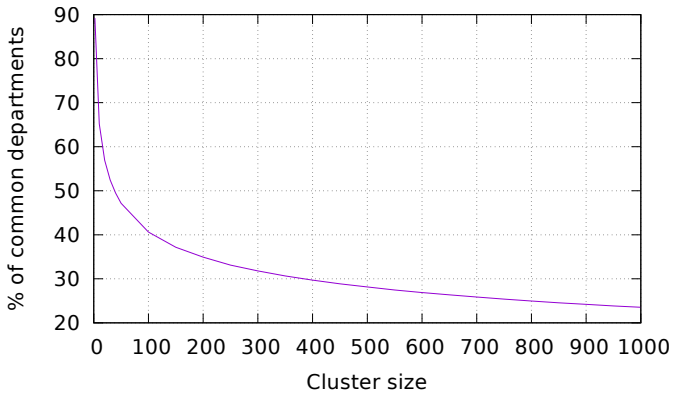


Figure 6.2 – Average size of intersection (in terms of % of departments in $D \setminus R$) between solutions in the same cluster in function of the cluster size, where a cluster is a set of solutions neighbors in the objective space.

the intersection goes below the 40% between 100 solutions and below 24% between 1000 solutions. Therefore, it is expected that the partitioning system of P-PLS may not be so useful and not as efficient as in MOTSP.

6.4 Experiments

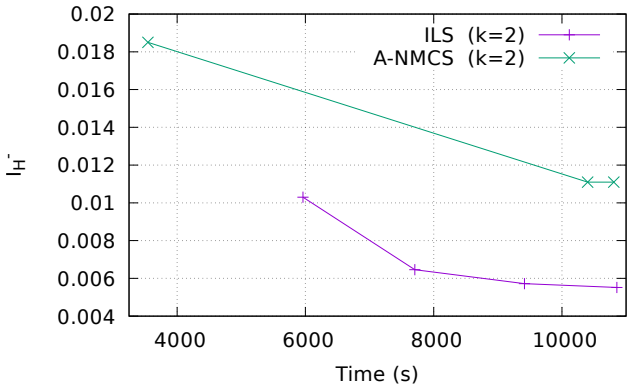
In this section, we first analyze the influence of the parameters on the performance of 2PIPLS/D, then we discuss about the results.

6.4.1 Sensitivity analysis of 2PIPLS/D on its parameters

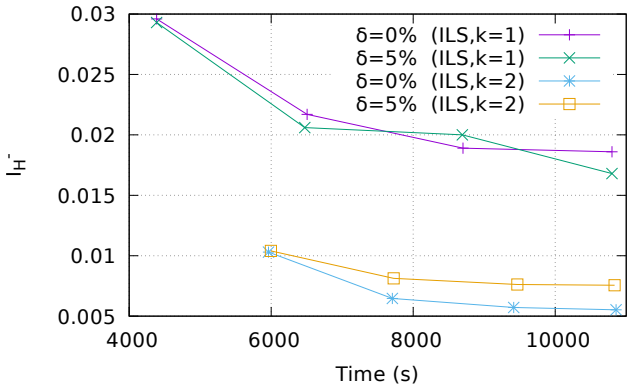
We carried out a sensitivity analysis of 2PIPLS/D on its parameters. Table 6.2 lists the parameters and their respective range of values, preselected after a number of internal tests. We will focus here only on the most important results. Each version of 2PIPLS/D with a given combination of parameter values has been run 10 times and the I_H^-, I_ϵ and I_{R2} values of each approximation set has been computed. For the sake of clarity, only I_H^- scores are displayed, knowing that same conclusions can be drawn with the other quality indicators. Except when IDA* is employed as SO solver, the running time of 2PIPLS/D is limited to 3 hours. Note that we have tested internally 2PIPLS/D with a running time of 6 hours and results do not change significantly.

Parameter	Module of 2PIPLS/D concerned	Selected range of values	Final value
SO optimizer	-	{IDA*,ILS,A-NMCS}	ILS
perturbation move	perturbation step (phase 2)	-	10-department move
min. acceptance rate threshold α	A-MDW	$\{5\%, 25\%, 50\%\}$	25%
data perturbation coefficient δ	SO optimizer (phase 2)	$\{0\%, 5\%\}$	0%
maximum initial size of a part σ	P-PLS	$\{5, 50, +\infty\}$	5
maximum neighborhood structure size k	PLS-VND	$\{1, 2, 3\}$	2
independent-pls	PLS-VND	{no, yes}	yes
first-dominating	PLS-VND	{no, yes}	yes
explore-dominated	PLS-VND	{no, yes}	yes
tolerance for dominance relations ϵ	-	-	1%
archive	-	-	SANDRA

Table 6.2 – Parameters of 2PIPLS/D, their respective selected range of values and final values.



(a) Comparison between 2PIPLS/D with ILS (and $k = 2$) vs. 2PIPLS/D with A-NMCS (and $k = 2$).



(b) Comparison between 2PIPLS/D with data perturbation ($\delta = 5\%$ and $k = 1, 2$) vs. 2PIPLS/D without data perturbation ($\delta = 0\%$ and $k = 1, 2$).

Figure 6.3 – Influence of the SO solver used and the presence of data perturbation on 2PIPLS/D performance in terms of I_H^- (median values, to be minimized) in function of the running time. Curves start at the end of the first phase (A-MDW+P-PLS) of 2PIPLS/D.

First of all, running **2PIPLS/D with IDA* as SO optimizer achieved poor performance**. While each IDA* run was limited to 1 hour, not a single run found any efficient solution. 2PIPLS/D remained stuck at A-MDW and we stopped 2PIPLS/D after a whole day of execution. Therefore, we will not consider anymore IDA* in the rest of this chapter.

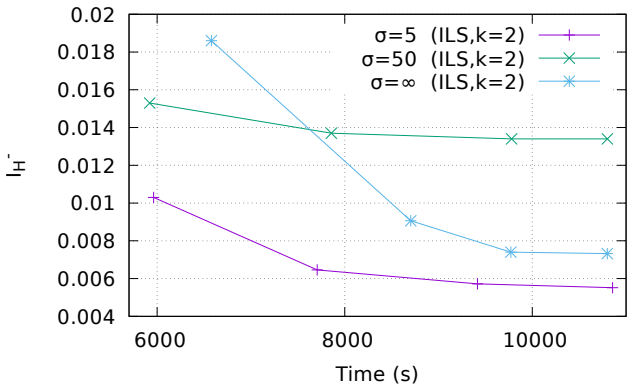
Secondly, like in MOTSP (see Section 5.4.2), an acceptance rate threshold α of A-MDW set to 25% provides good results, whatever the SO optimizer used (either ILS or A-NMCS) in 2PIPLS/D.

Thirdly, like in MOTSP (see Section 5.4.5), we found that any combination of exploration strategies of PLS-VND (without/with *first-dominating* strategy and *ignore/explore-dominated* strategy) provide very similar results. Their influence on 2PIPLS/D is therefore negligible and we activate both strategies.

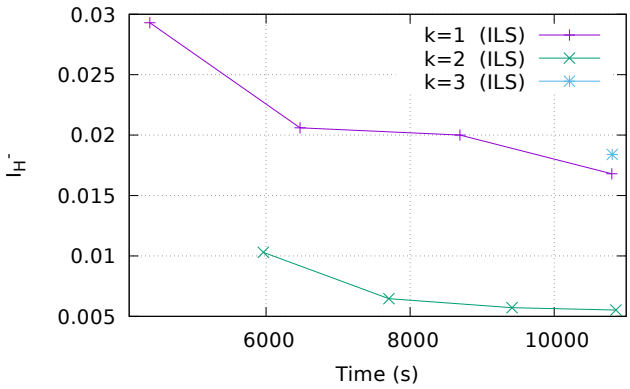
The comparison between 2PIPLS/D with ILS (and a maximum neighborhood structure size $k = 2$) and 2PIPLS/D with A-NMCS (and $k = 2$) is illustrated in Figure 6.3(a). While a single run of ILS is 15 times faster (in average) than an A-NMCS run, **2PIPLS/D with ILS outperforms 2PIPLS/D with A-NMCS**. Similar results are obtained with $k = 1$. As a consequence, the remaining results will only consider 2PIPLS/D with ILS as SO solver.

Figure 6.3(b) shows the performance of 2PIPLS/D with data perturbation ($\delta = 5\%$, $k = 1, 2$) and without ($\delta = 0\%$, $k = 1, 2$). It seems that the presence of data perturbation has a negative impact on 2PIPLS/D performance, either for $k = 1$ or $k = 2$. **This highlights the fact that optimizing a perturbed weighted-sum problem is quite hazardous.** Indeed, as mentioned in [Cornu et al., 2017], there is no guarantee that the optimal solution of a data perturbed weighted sum problem is efficient for the addressed MO problem. Therefore, solving a perturbed weighted sum problem may not produce any efficient solution, contrary to a non-perturbed weighted-sum problem which is therefore safer to optimize. Typically, on MOTSP, C-LK is so efficient that it generally finds in a single run the optimal solution of a non-perturbed weighted sum problem on small-size Euclidean instances. In consequence, C-LK quickly finds a large portion of the supported solutions of such instances. Because supported solutions are generally far fewer than non-supported solutions, data perturbation is welcome in such instances (as seen in Section 5.4.3) because it brings the ability to C-LK (or any other SO solver) to find non-supported solutions, and thus restores the utility of C-LK. On the other hand, our ILS is not as efficient on MOFRMP as C-LK on small-size Euclidean instances. Consequently, it seems safer to optimize non-perturbed weighted-sum problems.

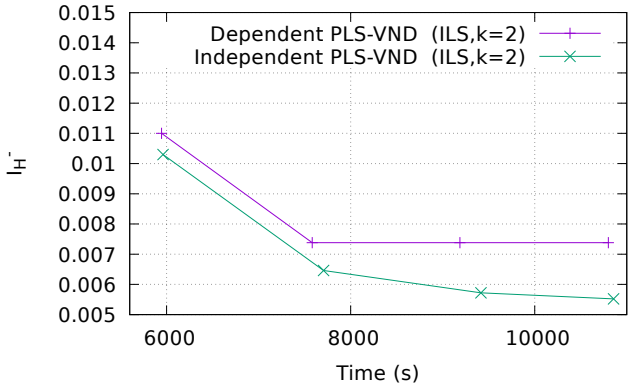
As expected, the partitioning system of P-PLS is not as efficient as for MOTSP, as shown in Figure 6.4(a),



(a) Comparison between different versions of 2PIPLS/D (with ILS and $k = 2$) with alternative values of maximum initial size of a part σ of P-PLS.



(b) Comparison between different versions of 2PIPLS/D (with ILS) with alternative values of PLS-VND maximum neighborhood structure size k.



(c) Comparison between 2PIPLS/D with dependent PLS-VND vs. 2PIPLS/D with independent PLS-VND.

Figure 6.4 – Influence of the maximum initial size of a part σ of P-PLS, the PLS-VND maximum neighborhood structure size k, and the (In)dependence of PLS-VND on 2PIPLS/D performance in terms of I_H^- (median values, to be minimized) in function of the running time. Curves start at the end of the first phase (A-MDW+P-PLS) of 2PIPLS/D.

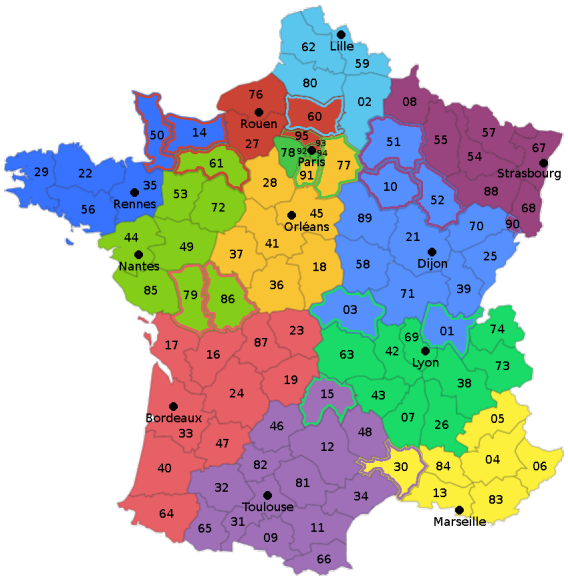


Figure 6.5 – One of the (potentially) efficient maps the least distant from the current map.

even though 2PIPLS/D with $\sigma = 5$ finds better results than no partitioning ($\sigma = +\infty$).

Concerning the maximum neighborhood structure size k used in PLS-VND, Figure 6.4(b) shows that $k = 2$ is the best alternative given the total execution time imposed (3 hours for each run), as $k = 1$ finds poorer results while $k = 3$ is too time consuming.

Finally, in Figure 6.4(c), we can see that using PLS-VND in Dependent mod stops improving very early (from 2 hours of running time). This shows the utility to rediscover the search space with PLS-VND with a different starting set of solutions each time, like the PLS-VND in Independent mod does.

6.4.2 Discussion about the results

After merging and filtering all solutions generated by the different 2PIPLS/D runs, we found 248,794 (potentially) efficient maps.

Before attempting to propose a map as an alternative to the current map (Figure 6.1) among the (potentially) efficient maps, it seems important to indicate that all the (potentially) efficient maps have at least 18 departments assigned to a region different from their current region of assignment. Figure 6.5 illustrates a map with 18 assignments of departments different from the current map and dominating it.

Knowing this, it seems difficult to propose any of these (potentially) efficient maps as an alternative. Indeed, it is very likely that any decision-maker will reject any of these alternative maps since the political and structural changes induced by a reassignment of at least 18 departments (over 82) are probably unacceptable in real life. Therefore, instead of suggesting an alternative map, we propose to identify which departments are more attracted to other regions than the one they are assigned to in the current map. To do that, we will use the (potentially) efficient maps found.

Since the decision-makers are members of the government, or at least their closest advisers in the state apparatus, we do not have access to their preferences concerning the relative importance between the objectives. Besides, in order to avoid making any compromise between the objectives, we decided to consider only the (potentially) efficient maps which dominate the current map, which are 4913 -representing 2% of all (potentially) efficient maps-.

From this reduced set of maps, we compute the so-called *membership score* of each department, where the membership score of a department $d \in D$ is the number of solutions of the reduced set in which d is assigned

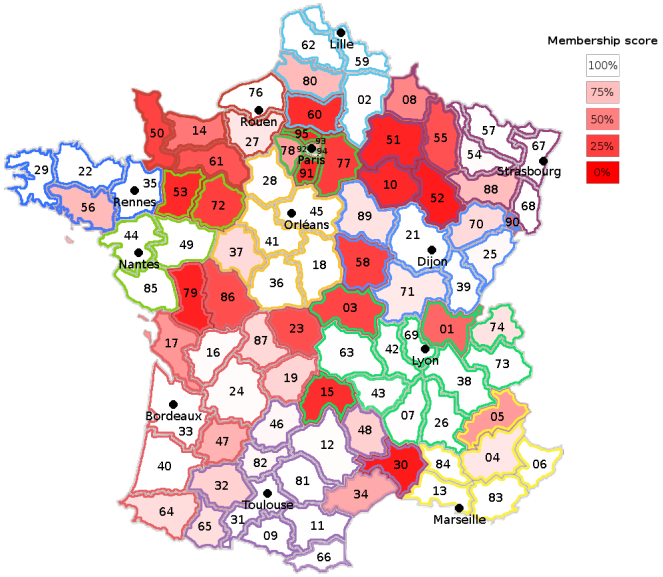


Figure 6.6 – Map of membership scores of departments.

Department	Membership score	Current region of assignment	Most attractive region
Gard (30)	10%	Occitanie (31)	Provence-Alpes-Côte d'Azur (13)
Haute-Marne (52)	11%	Grand Est (67)	Bourgogne-Franche-Comté (21)
Deux-Sèvres (79)	13%	Nouvelle-Aquitaine (33)	Pays de la Loire (44)
Marne (51)	14%	Grand Est (67)	Bourgogne-Franche-Comté (21)
Essonne (91)	15%	Île-de-France (75)	Centre-Val de Loire (45)
Aube (10)	16%	Grand Est (67)	Bourgogne-Franche-Comté (21)
Val-d'Oise (95)	16%	Île-de-France (75)	Normandie (76)
Mayenne (53)	16%	Pays de la Loire (44)	Bretagne (35)
Oise (60)	17%	Hauts-de-France (59)	Normandie (76)
Cantal (15)	18%	Auvergne-Rhône-Alpes (69)	Occitanie (31)

Table 6.3 – The top 10 departments with the lowest membership score (i.e. the departments the most attracted to other regions than their current region of assignment).

to its current region of assignment .The membership score of a department $d \in D$ ranges from 0% to 100%, where 0% indicates that d has no membership towards its current region of assignment and thus is completely attracted to other regions, while 100% indicates that d is always assigned to its current region and thus is absolutely not attracted by another region. We obtain the map in Figure 6.6 and Table 6.3 lists the top 10 departments with the lowest membership score and the region to which they are the most attracted.

Given the multiple knowledge (economic, financial, political, administrative and many others) necessary to master the question of the need (or not) of the re-assignment of a department to another region than its current region of assignment, it is difficult for us to know if the membership score is a consistent evaluation for such a question. Anyway, it is interesting to note that a large number of departments currently assigned to the region *Grand Est (67)* have a very low membership score. Besides, politicians regularly suggest[18][19] to reduce the size of this region.

[18] http://france3-regions.francetvinfo.fr/grand-est/francois-fillon-nouvelle-reforme-territoriale-1143001.html

[19] [http://www.dna.fr/social/2017/04/29/medef-alsace-la-nouvelle-region-est-trop-grande||http://www.dna.fr/social/2017/04/29/medef-alsace-la-nouvelle-region-est-trop-grande]

6.5 Conclusion

In this chapter, we are interested in the recent territorial reform of French regions which resulted in the reassignment of departments to new larger regions. Starting by the observation that some departments are attracted to other regions than their current region of assignment, the aim of this chapter was to attempt to identify such departments through the optimization of a new many-objective real-world problem called MOFRMP. The problem consists in finding alternative maps optimizing five objectives based on economical and financial indicators measuring the strength of interactions between the departments assigned to the same region, as well as the economical and demographic weights of each region. We applied 2PIPLS/D to this problem and tested ILS and A-NMCS as SO solvers. We found that 2PIPLS/D with ILS outperforms 2PIPLS/D with A-NMCS. An approximation of the efficient set has been generated with the different runs of 2PIPLS/D. However, we found that the (potentially) efficient maps found have far more departments of difference with the current map, making unrealistic to propose them as alternative of the current map. Instead, we built an indicator based on the found approximation set which aims at identifying the departments the most attracted by other regions than the current they are assigned to. In particular, we found that Gard (30), Haute-Marne (52) and Deux-Sèvres (79) are particularly attracted to other regions.

MOFRMP raises an important question: could the current map of regions be improved? The work proposed in this chapter corresponds only to a preliminary answer to this question. We believe that a new formulation of MOFRMP is necessary for obtaining realistic alternative maps. Firstly it seems to us important not to restrict this problem to economic and financial criteria only. As examples, the cultural exchanges between departments could be considered, or the global intensity of movements of people between the departments, not only commuting. Secondly, it is imperative to obtain the preferences of the decision-makers about the relative importance of the different criteria considered. Thirdly, we think that any alternative map has to propose a very limited number of reassignments of departments compared to the current map. If the maximum number of reassignments is limited to 3 or 4 for example, the problem becomes quite easy and it seems that no optimization method is needed. Indeed, in this case, a simple enumeration of the solutions is necessary, finally a filtering to keep only efficient solutions, then applying a Multiple Criteria Decision Aid method with the preferences of the decision-makers previously retrieved. On the other hand, if a higher number of reassignments is accepted (i.e. 5 or higher), the number of feasible solutions seems too large for enumeration. In this case, a MO optimization method is necessary and 2PIPLS/D could be adapted to this new formulation of MOFRMP.

Conclusion

Falling in the scope of Multi-Objective Combinatorial Optimization (MOCO), this thesis proposed a number of new approximation methods and data structures for large-scale MOCO problems. Firstly, the contributions of the thesis are summarized, then a number of perspectives are suggested.

Contributions

1. We proposed two new archives: AVL-Archive for the bi-objective case and NDR*-Archive for any number of objectives; as well as their self-adjusting versions using the concepts of temporal and spatial locality, and thus especially designed for Pareto Local Search (PLS): Self-Adjusting AVL-Archive (SAAVLA) and Self-Adjusting NDR*-Archive (SANDRA). We compared these archives to state-of-the-art archives and we obtain better results than competitors on all tested instances, making AVL-Archive, NDR*-Archive and their self-adjusting versions the most efficient archives up to 5 objectives.

2. We presented a new method generalizing the concept of Maximally Dispersed Weights called Adaptive MDW (A-MDW) and making it more adaptive such that the method continues generating weights until the SO solver optimizing the related weighted sum problems is no longer efficient.

3. We introduced Partitioned Pareto Local Search (P-PLS), which aims at speeding-up PLS-VND through the partitioning of the set of solutions to explore, and a smart restriction of the PLS-VND neighborhood structure based on the presumption of global convexity on the addressed problem.

4. We suggested some modifications for any SO optimizer to improve its ability to find efficient solutions in MOCO problems, in particular memorizing the solutions generated during the optimization process. Based on these specifications, we proposed a new version of Nested Monte Carlo Search (NMCS), called Aggregation-based NMCS (A-NMCS) better suited for MOCO than vanilla NMCS.

5. We presented a new component-wise meta-heuristic called 2-Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D). During the first phase, A-MDW is used as initialization method and P-PLS builds a partition for speeding-up PLS-VND. At each iteration of the second phase, a set of solutions is first generated through the optimization of several well-dispersed data perturbed weighted-sum problems, then PLS-VND is conducted from this set. 2PIPLS/D handles different mods and exploration strategies for PLS-VND, and embeds an ϵ-archive system which enable a good distribution in the objective space of the generated points and bounds the size of the approximation.

6. We implemented 2PIPLS/D for the MO symmetric Traveling Salesman Problem (MOTSP). 2PIPLS/D has been compared with state-of-the-art methods on large scale bi-objective and tri-objective TSP instances. Computational experiments show that 2PIPLS/D outperforms competitors on all tested instances, except for 5% of the instances where our own previous method PDA and 2PIPLS/D perform similarly. P-PLS provides surprisingly good performance as it reduces the number of explored solutions by 98% in average, compared to no partitioning. In the continuation of the work of Borges and Hansen [Borges and Hansen, 2002], we have experimentally confirmed the assumption of global convexity on

MOTSP on a number of bi-objective and tri-objective instances. In the continuation of the work of [Lust and Jaszkiewicz, 2010], we found that edges present in efficient solutions are of good quality compared to the other edges.

7. We implemented 2PIPLS/D for a new real-world problem related to the recent territorial reform of French regions, and called the French Regions Mapping Problem (MOFRMP), for which we proposed a MO formulation with 5 objectives. We have detected a number of departments of Metropolitan France more attracted by other regions than the current they belong to.

8. For both MOTSP and MOFRMP, 2PIPLS/D with the Monte Carlo Search-based method A-NMCS as SO solver is outperformed in most cases by 2PIPLS/D with a Local Search-based method.

Short-term and Middle-term perspectives

2PIPLS/D could be tested on other MOCO problems like MO asymmetric TSP, MO Minimum Spanning Tree, MO (Multidimensional) 0-1 Knapsack or MO (Quadratic) Assignment Problems. We are particularly interested in analyzing the behavior of the partitioning system of P-PLS on instances with negatively correlated objectives, making much more difficult the optimization of Local Search techniques, because it has an impact on the level of clustering of the solutions on these instances ([Paquete and Stützle, 2009a, Verel et al., 2011]).

Concerning PLS-VND in independent mod, it could be interesting to implement a MO tabu mechanism [Hansen, 1997] in order not to re-explore solutions already explored in previous runs.

Long-term perspectives: beyond PLS ?

Within the 2PIPLS/D framework, instead of employing PLS(-VND), it could be interesting to use an exact combinatorial optimization method. In fact, the partitioning method introduced in P-PLS is independent from the concept of PLS (it has indeed be designed as completely distinct from PLS). Therefore, the idea is to run a MO Branch and Bound or a MO Dynamic Programming algorithm (for example) from the reduced search space provided by the partitioning method. Note that the new archives we have proposed would be of high interest if a MO Dynamic Programming is used. For example, on MOTSP, one could use a MO version of the Branch Decomposition-based Dynamic Programming method of Cook and Seymour [Cook and Seymour, 2003]. For MOKP, the method employed could be the MO Dynamic Program of Bazgan et al. [Bazgan et al., 2009b]. We believe that this search direction is promising given the large reduction of the search space provided by the partitioning system experienced on MOTSP.

References

[Abramson, 1990] Abramson, B. (1990). Expected-outcome: A general model of static evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(2):182–193.

[Adelson-Velskii, 1962] Adelson-Velskii, G. (1962). Landis.". *An algorithm for the organization of information, Proceedings of the USSR Academy of Sciences*, 146:263–266.

[Alaya et al., 2007] Alaya, I., Solnon, C., and Ghedira, K. (2007). Ant colony optimization for multi-objective optimization problems. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, volume 1, pages 450–457. IEEE.

[Altwaijry and Menai, 2012] Altwaijry, N. and Menai, M. E. B. (2012). Data structures in multi-objective evolutionary algorithms. *Journal of Computer Science and Technology*, 27(6):1197.

[Amabile et al., 2015] Amabile, A., Bernard, C., and Epaulard, A. (2015). Une évaluation de la cohérence économique interne des régions. Technical report, France Stratégie, Document de travail.

[Andersen et al., 1996] Andersen, K. A., Jörnsten, K., and Lind, M. (1996). On bicriterion minimal spanning trees: An approximation. *Computers & Operations Research*, 23(12):1171–1182.

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

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

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

[Arneson et al., 2010] Arneson, B., Hayward, R. B., and Henderson, P. (2010). Monte carlo tree search in hex. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):251–258.

[Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

[Bäck et al., 1997] Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997). Handbook of evolutionary computation. *New York: Oxford*.

[Barán and Schaerer, 2003] Barán, B. and Schaerer, M. (2003). A multiobjective ant colony system for vehicle routing problem with time windows. In *Applied informatics*, pages 97–102.

[Bazgan et al., 2009a] Bazgan, C., Hugot, H., and Vanderpooten, D. (2009a). Implementing an efficient fptas for the 0–1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1):47–56.

[Bazgan et al., 2009b] Bazgan, C., Hugot, H., and Vanderpooten, D. (2009b). Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279.

[Beckmann et al., 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The r*-tree: an efficient and robust access method for points and rectangles. In *ACM SIGMOD Record*, volume 19, pages 322–331. Acm.

[Belhoul et al., 2014] Belhoul, L., Galand, L., and Vanderpooten, D. (2014). An efficient procedure for finding best compromise solutions to the multi-objective assignment problem. *Computers & Operations Research*, 49:97–106.

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

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

[Bentley et al., 1993] Bentley, J. L., Clarkson, K. L., and Levine, D. B. (1993). Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, 9(2):168–183.

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

[Blazinskas and Misevicius, 2011] Blazinskas, A. and Misevicius, A. (2011). Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. In *17th International Conference on Information and Software Technologies*, pages 27–29.

[Bökler and Mutzel, 2015] Bökler, F. and Mutzel, P. (2015). Output-sensitive algorithms for enumerating the extreme nondominated points of multiobjective combinatorial optimization problems. In *Algorithms-ESA 2015*, pages 288–299. Springer.

[Borges and Hansen, 2002] Borges, P. C. and Hansen, M. P. (2002). A study of global convexity for a multiple objective travelling salesman problem. In *Essays and surveys in metaheuristics*, pages 129–150. Springer.

[Brimberg et al., 2000] Brimberg, J., Hansen, P., Mladenović, N., and Taillard, E. D. (2000). Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem. *Operations Research*, 48(3):444–460.

[Browne et al., 2012] Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43.

[Cazenave, 2009] Cazenave, T. (2009). Nested monte-carlo search. In *IJCAI*, pages 456–461.

[Cazenave, 2012] Cazenave, T. (2012). Monte carlo beam search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):68–72.

[Cazenave, 2015] Cazenave, T. (2015). Generalized rapid action value estimation. In *IJCAI*, pages 754–760.

[Cazenave, 2016] Cazenave, T. (2016). Nested rollout policy adaptation with selective policies. In *Computer Games*, pages 44–56. Springer.

[Cazenave et al., 2009] Cazenave, T., Balbo, F., and Pinson, S. (2009). Monte-Carlo bus regulation. In *ITSC*, pages 340–345, St. Louis.

[Cazenave et al., 2016] Cazenave, T., Bernard, C., and Epaulard, A. (2016). Optimizing french regions. In *JFPDA, Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes*.

[Cazenave and Teytaud, 2012] Cazenave, T. and Teytaud, F. (2012). Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In *LION*, volume 7219 of *Lecture Notes in Computer Science*, pages 42–54. Springer.

[Cerqueus et al., 2017] Cerqueus, A., Gandibleux, X., Przybylski, A., and Saubion, F. (2017). On branching heuristics for the bi-objective 0/1 unidimensional knapsack problem. *Journal of Heuristics*, 23(5):285–319.

[Chang et al., 2008] Chang, P. C., Chen, S. H., Zhang, Q., and Lin, J. L. (2008). Moea/d for flowshop scheduling problems. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1433–1438. IEEE.

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

[Charon and Hudry, 2002] Charon, I. and Hudry, O. (2002). The noising methods: a survey. In *Essays and surveys in metaheuristics*, pages 245–261. Springer.

[Chaslot et al., 2008] Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. (2008). Monte-carlo tree search: A new framework for game ai. In *AIIDE*.

[Chen et al., 2009] Chen, C.-M., Chen, Y.-p., and Zhang, Q. (2009). Enhancing moea/d with guided mutation and priority update for multi-objective optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 209–216. IEEE.

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

[Clímaco and Pascoal, 2012] Clímaco, J. C. and Pascoal, M. (2012). Multicriteria path and tree problems: discussion on exact algorithms and applications. *International Transactions in Operational Research*, 19(1-2):63–98.

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

[Coello et al., 2007] Coello, C. A. C., Lamont, G. B., and Van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer.

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

[Cook and Seymour, 2003] Cook, W. and Seymour, P. (2003). Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248.

[Cornu et al., 2017] Cornu, M., Cazenave, T., and Vanderpooten, D. (2017). Perturbed decomposition algorithm applied to the multi-objective traveling salesman problem. *Computers & Operations Research*, 79:314–330.

[Coulom, 2006] Coulom, R. (2006). Efficient selectivity and back-up operators in monte-carlo tree search. In *Computers and Games 2006*, Volume 4630 of LNCS, pages 72–83, Torino, Italy. Springer.

[Coulom, 2007] Coulom, R. (2007). Computing elo ratings of move patterns in the game of go. In *Computer games workshop*.

[Czyzżak and Jaszkiewicz, 1998] Czyzżak, P. and Jaszkiewicz, A. (1998). Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47.

[Dächert et al., 2017] Dächert, K., Klamroth, K., Lacour, R., and Vanderpooten, D. (2017). Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research*, 260(3):841–855.

[Deb et al., 2000] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: Nsga-ii. In *PPSN*, pages 849–858.

[Deb et al., 2016] Deb, K., Sindhya, K., and Hakanen, J. (2016). Multi-objective optimization. In *Decision Sciences: Theory and Practice*, pages 145–184. CRC Press.

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

[Dréo et al., 2003] Dréo, J., Pétrowski, A., Siarry, P., and Taillard, E. (2003). *Métaheuristiques pour l'optimisation difficile*. Eyrolles.

[Drozdik et al., 2015] Drozdik, M., Akimoto, Y., Aguirre, H., and Tanaka, K. (2015). Computational cost reduction of nondominated sorting using the m-front. *IEEE Transactions on Evolutionary Computation*, 19(5):659–678.

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

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

[Dubois-Lacoste et al., 2011a] Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011a). A hybrid tp+ pls algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236.

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

[Dubois-Lacoste et al., 2012] Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2012). Pareto local search algorithms for anytime bi-objective optimization. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 206–217. Springer.

[Dubois-Lacoste et al., 2013] Dubois-Lacoste, J., López-Ibánez, M., and Stützle, T. (2013). Combining two search paradigms for multi-objective optimization: Two-phase and pareto local search. In *Hybrid Metaheuristics*, pages 97–117. Springer.

[Dubois-Lacoste et al., 2015] Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2015). Anytime pareto local search. *European journal of operational research*, 243(2):369–385.

[Dubois-Lacoste et al., 2014] Dubois-Lacoste, J., Stützle, T., Birattari, M., and López-Ibáñez, M. (2014). *Anytime Local Search for Multi-Objective Combinatorial Optimization: Design, Analysis and Automatic Configuration*. PhD thesis, Citeseer.

[Edelkamp et al., 2013] Edelkamp, S., Gath, M., Cazenave, T., and Teytaud, F. (2013). Algorithm and knowledge engineering for the tsptw problem. In *Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on*, pages 44–51. IEEE.

[Edelkamp et al., 2016] Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., and Lawo, M. (2016). Monte-carlo tree search for logistics. In *Commercial Transport*, pages 427–440. Springer.

[Edelkamp and Greulich, 2014] Edelkamp, S. and Greulich, C. (2014). Solving physical traveling salesman problems with policy adaptation. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–8. IEEE.

[Edelkamp and Tang, 2015] Edelkamp, S. and Tang, Z. (2015). Monte-carlo tree search for the multiple sequence alignment problem. In *Eighth Annual Symposium on Combinatorial Search*.

[Edelsbrunner and Grayson, 2000] Edelsbrunner, H. and Grayson, D. R. (2000). Edgewise subdivision of a simplex. *Discrete & Computational Geometry*, 24(4):707–719.

[Ehrgott, 2006] Ehrgott, M. (2006). *Multicriteria optimization*. Springer Science & Business Media.

[Ehrgott, 2009] Ehrgott, M. (2009). Multiobjective (combinatorial) optimisation-some thoughts on applications. In *Multiobjective Programming and Goal Programming*, pages 267–282. Springer.

[Ehrgott and Gandibleux, 2004] Ehrgott, M. and Gandibleux, X. (2004). Approximative solution methods for multiobjective combinatorial optimization. *Top*, 12(1):1–63.

[Ehrgott and Gandibleux, 2008] Ehrgott, M. and Gandibleux, X. (2008). Hybrid metaheuristics for multiobjective combinatorial optimization. In *Hybrid metaheuristics*, pages 221–259. Springer.

[Ehrgott and Klamroth, 1997] Ehrgott, M. and Klamroth, K. (1997). Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research*, 97(1):159–166.

[Eliahou et al., 2013] Eliahou, S., Fonlupt, C., Fromentin, J., Marion-Poty, V., Robilliard, D., and Teytaud, F. (2013). *Investigating monte-carlo methods on the weak schur problem*. Springer.

[Feo and Resende, 1989] Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71.

[Feo and Resende, 1995] Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

[Finkel and Bentley, 1974] Finkel, R. A. and Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9.

[Finnsson and Björnsson, 2008] Finnsson, H. and Björnsson, Y. (2008). Simulation-based approach to general game playing. In *AAAI*, volume 8, pages 259–264.

[Florios and Mavrotas, 2014] Florios, K. and Mavrotas, G. (2014). Generation of the exact pareto set in multi-objective traveling salesman and set covering problems. *Applied Mathematics and Computation*, 237:1–19.

[Fonseca et al., 2005] Fonseca, C. M., Knowles, J. D., Thiele, L., and Zitzler, E. (2005). A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 216, page 240.

[Fonseca et al., 2006] Fonseca, C. M., Paquete, L., and López-Ibánez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *Congress on Evolutionary Computation*, pages 1157–1163. IEEE.

[Gabrel and Vanderpooten, 2002] Gabrel, V. and Vanderpooten, D. (2002). Enumeration and interactive selection of efficient paths in a multiple criteria graph for scheduling an earth observing satellite. *European Journal of Operational Research*, 139(3):533–542.

[Galand et al., 2010] Galand, L., Perny, P., and Spanjaard, O. (2010). Choquet-based optimisation in multiobjective shortest path and spanning tree problems. *European Journal of Operational Research*, 204(2):303–315.

[Gandibleux et al., 1997] Gandibleux, X., Mezdaoui, N., and Fréville, A. (1997). A tabu search procedure to solve multiobjective combinatorial optimization problems. In *Advances in multiple objective and goal programming*, pages 291–300. Springer.

[Gandibleux et al., 1998a] Gandibleux, X., Morita, H., and Katoh, N. (1998a). A genetic algorithm for 0-1 multiobjective knapsack problem. In *NACA98*.

[Gandibleux et al., 2001] Gandibleux, X., Morita, H., and Katoh, N. (2001). The supported solutions used as a genetic information in a population heuristics. In *EMO*, pages 429–442. Springer.

[Gandibleux et al., 1998b] Gandibleux, X., Vancoppenolle, D., and Tuyttens, D. (1998b). A first making use of grasp for solving moco problems. Technical report, University of Valenciennes, France.

[García-Martínez et al., 2007] García-Martínez, C., Cordón, O., and Herrera, F. (2007). 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):116–148.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). A guide to the theory of np-completeness. *WH Freemann, New York*, 70.

[Gelly and Silver, 2007] Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM.

[Gendreau and Potvin, 2005] Gendreau, M. and Potvin, J.-Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213.

[Geoffrion, 1967] Geoffrion, A. M. (1967). Proper efficiency and the theory of vector maximization. Technical report, DTIC Document.

[Ghoseiri and Nadjari, 2010] Ghoseiri, K. and Nadjari, B. (2010). An ant colony optimization algorithm for the bi-objective shortest path problem. *Applied Soft Computing*, 10(4):1237–1246.

[Ginsberg, 2001] Ginsberg, M. L. (2001). Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358.

[Glover, 1989] Glover, F. (1989). Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic algorithms in search optimization and machine learning*, volume 412. Addison-wesley Reading Menlo Park.

[Gonçalves et al., 2006] Gonçalves, E. N., Palhares, R. M., Takahashi, R. H. C., and Mesquita, R. C. (2006). Algorithm 860: Simples - an extension of freudenthal's simplex subdivision. *ACM Transactions on Mathematical Software (TOMS)*, 32(4):609–621.

[Gorski et al., 2006] Gorski, J., Klamroth, K., and Ruzika, S. (2006). *Connectedness of efficient solutions in multiple objective combinatorial optimization*. Inst. für Angewandte Mathematik.

[Greco et al., 2005] Greco, S., Figueira, J., and Ehrgott, M. (2005). Multiple criteria decision analysis. *Springer's International series*.

[Guerreiro et al., 2012] Guerreiro, A. P., Fonseca, C. M., and Emmerich, M. T. (2012). A fast dimension-sweep algorithm for the hypervolume indicator in four dimensions. In *CCCG*, pages 77–82.

[Gutin and Punnen, 2006] Gutin, G. and Punnen, A. P. (2006). *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media.

[Guttman, 1984] Guttman, A. (1984). *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM.

[Habenicht, 1983] Habenicht, W. (1983). Quad trees, a datastructure for discrete vector optimization problems. In *Essays and Surveys on Multiple Criteria Decision Making*, pages 136–145. Springer.

[Haimes et al., 1971] Haimes, Y. Y., Ladson, L., and Wismer, D. A. (1971). Bicriterion formulation of problems of integrated system identification and system optimization. *IEEE Transactions on Systems Man and Cybernetics*, (3):296.

[Hamacher and Ruhe, 1994] Hamacher, H. W. and Ruhe, G. (1994). On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52(4):209–230.

[Hansen, 1997] Hansen, M. P. (1997). Tabu search for multiobjective optimization: Mots. In *Proceedings of the 13th International Conference on Multiple Criteria Decision Making*, pages 574–586. Citeseer.

[Hansen, 2000] Hansen, M. P. (2000). Use of substitute scalarizing functions to guide a local search based heuristic: The case of motsp. *Journal of heuristics*, 6(3):419–431.

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

[Hastings, 1970] Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.

[Helsgaun, 2000] Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

[Helsgaun, 2009] Helsgaun, K. (2009). General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation*, 1(2):119–163.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

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

[Hoos and Stützle, 2004] Hoos, H. H. and Stützle, T. (2004). *Stochastic local search: Foundations and applications*. Elsevier.

[Hwang et al., 2003] Hwang, S., Kwon, K., Cha, S. K., and Lee, B. S. (2003). Performance evaluation of main-memory r-tree variants. In *International Symposium on Spatial and Temporal Databases*, pages 10–27. Springer.

[Inja et al., 2014] Inja, M., Kooijman, C., de Waard, M., Roijers, D. M., and Whiteson, S. (2014). Queued pareto local search for multi-objective optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 589–599. Springer.

[Ishibuchi and Murata, 1996] Ishibuchi, H. and Murata, T. (1996). Multi-objective genetic local search algorithm. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 119–124. IEEE.

[Ishibuchi and Murata, 1998] Ishibuchi, H. and Murata, T. (1998). 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):392–403.

[Jaszkiewicz, 1999] Jaszkiewicz, A. (1999). Improving performance of genetic local search by changing local search space topology. *Foundations of Computing and Decision Sciences*, 24(2):77–84.

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

[Jaszkiewicz, 2017] Jaszkiewicz, A. (2017). Many-objective pareto local search. *arXiv preprint arXiv:1707.07899*.

[Jaszkiewicz and Lust, 2016] Jaszkiewicz, A. and Lust, T. (2016). Nd-tree: a fast online algorithm for updating a pareto archive and its application in many-objective pareto local search. *arXiv*.

[Jaszkiewicz and Lust, 2017] Jaszkiewicz, A. and Lust, T. (2017). Proper balance between search towards and along pareto front: biobjective tsp case study. *Annals of Operations Research*, pages 1–20.

[Jaszkiewicz and Zielniewicz, 2006] Jaszkiewicz, A. and Zielniewicz, P. (2006). 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*.

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

[Kamel and Faloutsos, 1993] Kamel, I. and Faloutsos, C. (1993). Hilbert r-tree: An improved r-tree using fractals. Technical report.

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

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

[Kennedy, 2011] Kennedy, J. (2011). Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer.

[Kennedy and Eberhart, 1997] Kennedy, J. and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, volume 5, pages 4104–4108. IEEE.

[Kinny, 2012] Kinny, D. (2012). A new approach to the snake-in-the-box problem. In *ECAI*, volume 242, pages 462–467.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[Klamroth et al., 2015] Klamroth, K., Lacour, R., and Vanderpooten, D. (2015). On the representation of the search region in multi-objective optimization. *European Journal of Operational Research*, 245(3):767–778.

[Knuth, 1998] Knuth, D. E. (1998). *The art of computer programming: sorting and searching*, volume 3. Pearson Education.

[Kocsis and Szepesvàri, 2006] Kocsis, L. and Szepesvàri, C. (2006). Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer.

[Korf, 1985] Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109.

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

[Kung et al., 1975] Kung, H.-T., Luccio, F., and Preparata, F. P. (1975). On finding the maxima of a set of vectors. *Journal of the ACM (JACM)*, 22(4):469–476.

[Lacour, 2014] Lacour, R. (2014). *Exact and approximate solving approaches in multi-objective combinatorial optimization, application to the minimum weight spanning tree problem.* PhD thesis, Université Paris Dauphine-Paris IX.

[Landau and Lifshitz, 1980] Landau, L. and Lifshitz, E. (1980). Statistical physics, vol. 5. *Course of theoretical physics*, 30.

[Larrañaga and Lozano, 2001] Larrañaga, P. and Lozano, J. A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media.

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

[Li and Landa-Silva, 2009] Li, H. and Landa-Silva, D. (2009). An elitist grasp metaheuristic for the multi-objective quadratic assignment problem. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 481–494. Springer.

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

[Li and Zhang, 2009] Li, H. and Zhang, Q. (2009). Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE Transactions on Evolutionary Computation*, 13(2):284–302.

[Li et al., 2014] Li, K., Deb, K., Zhang, Q., and Kwong, S. (2014). Efficient non-domination level update approach for steady-state evolutionary multiobjective optimization. *Department of Electtrical and Computer Engineering, Michigan State University, East Lansing, USA, Tech. Rep. COIN Report*, (2014014).

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

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

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

[Loukil et al., 2005] Loukil, T., Teghem, J., and Tuyttens, D. (2005). Solving multi-objective production scheduling problems using metaheuristics. *European journal of operational research*, 161(1):42–61.

[Lourenço et al., 2003] Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). *Iterated local search*. Springer.

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

[Lust and Rolland, 2013] Lust, T. and Rolland, A. (2013). Choquet optimal set in biobjective combinatorial optimization. *Computers & Operations Research*, 40(10):2260–2269.

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

[Lust and Teghem, 2012] Lust, T. and Teghem, J. (2012). The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 19(4):495–520.

[Lust and Tuyttens, 2014] Lust, T. and Tuyttens, D. (2014). Variable and large neighborhood search to solve the multiobjective set covering problem. *Journal of Heuristics*, 20(2):165–188.

[Madakat et al., 2013] Madakat, D., Morio, J., and Vanderpooten, D. (2013). Biobjective planning of an active debris removal mission. *Acta Astronautica*, 84:182–188.

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

[Martí et al., 2015] Martí, R., Campos, V., Resende, M. G., and Duarte, A. (2015). Multiobjective grasp with path relinking. *European Journal of Operational Research*, 240(1):54–71.

[Martin et al., 1991] Martin, O., Otto, S. W., and Felten, E. W. (1991). Large-step markov chains for the traveling salesman problem. 5:299–326.

[Martins and Ribeiro, 2006] Martins, S. L. and Ribeiro, C. C. (2006). Metaheuristics and applications to optimization problems in telecommunications. In *Handbook of optimization in telecommunications*, pages 103–128. Springer.

[Moscato, 1989] Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989.

[Mostaghim and Teich, 2005] Mostaghim, S. and Teich, J. (2005). Quad-trees: A data structure for storing pareto sets in multiobjective evolutionary algorithms with elitism. In *Evolutionary Multiobjective Optimization*, pages 81–104. Springer.

[Murata and Ishibuchi, 1995] Murata, T. and Ishibuchi, H. (1995). Moga: Multi-objective genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 289. IEEE.

[Murata et al., 2001] Murata, T., Ishibuchi, H., and Gen, M. (2001). Specification of genetic search directions in cellular multi-objective genetic algorithms. In *Evolutionary multi-criterion optimization*, pages 82–95. Springer.

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

[Özpeynirci and Köksalan, 2010] Özpeynirci, Ö. and Köksalan, M. (2010). An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12):2302–2315.

[Papadimitriou and Yannakakis, 2000] Papadimitriou, C. H. and Yannakakis, M. (2000). On the approximability of trade-offs and optimal access of web sources. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 86–92. IEEE.

[Paquete et al., 2004] Paquete, L., Chiarandini, M., and Stützle, T. (2004). Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, pages 177–199. Springer.

[Paquete and Stützle, 2003] Paquete, L. and Stützle, T. (2003). A two-phase local search for the biobjective traveling salesman problem. In *Evolutionary Multi-Criterion Optimization*, pages 479–493. Springer.

[Paquete and Stützle, 2009a] Paquete, L. and Stützle, T. (2009a). Clusters of non-dominated solutions in multiobjective combinatorial optimization: An experimental analysis. In *Multiobjective Programming and Goal Programming*, pages 69–77. Springer.

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

[Pelikan et al., 2006] Pelikan, M., Sastry, K., and Goldberg, D. E. (2006). Multiobjective estimation of distribution algorithms. In *Scalable optimization via probabilistic modeling*, pages 223–248. Springer.

[Perez et al., 2015] Perez, D., Mostaghim, S., Samothrakis, S., and Lucas, S. M. (2015). Multiobjective monte carlo tree search for real-time games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):347–360.

[Perez et al., 2013] Perez, D., Samothrakis, S., and Lucas, S. (2013). Online and offline learning in multi-objective monte carlo tree search. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE.

[Poulding and Feldt, 2014] Poulding, S. and Feldt, R. (2014). Generating structured test data with specific properties using nested monte-carlo search. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1279–1286. ACM.

[Powley et al., 2013] Powley, E. J., Whitehouse, D., and Cowling, P. I. (2013). Monte carlo tree search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE.

[Przybylski and Gandibleux, 2017] Przybylski, A. and Gandibleux, X. (2017). Multi-objective branch and bound. *European Journal of Operational Research*, 260(3):856–872.

[Przybylski et al., 2010a] Przybylski, A., Gandibleux, X., and Ehrgott, M. (2010a). A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS Journal on Computing*, 22(3):371–386.

[Przybylski et al., 2010b] Przybylski, A., Gandibleux, X., and Ehrgott, M. (2010b). A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3):149–165.

[Qi et al., 2014] Qi, Y., Ma, X., Liu, F., Jiao, L., Sun, J., and Wu, J. (2014). Moea/d with adaptive weight adjustment. *Evolutionary computation*, 22(2):231–264.

[Rego et al., 2011] Rego, C., Gamboa, D., Glover, F., and Osterman, C. (2011). Traveling salesman problem heuristics: leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427–441.

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

[Rimmel et al., 2011] Rimmel, A., Teytaud, F., and Cazenave, T. (2011). Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows. In *Applications of Evolutionary Computation*, pages 501–510. Springer.

[Rosin, 2011] Rosin, C. D. (2011). Nested rollout policy adaptation for monte carlo tree search. In *IJCAI*, pages 649–654.

[Roy, 1985] Roy, B. (1985). *Méthodologie multicritère d" aide à la décision*. Economica.

[Schadd et al., 2008] Schadd, M. P. D., Winands, M. H. M., van den Herik, H. J., Chaslot, G., and Uiterwijk, J. W. H. M. (2008). Single-player monte-carlo tree search. In *Computers and Games*, pages 1–12.

[Schwarz and Ocenasek, 2001] Schwarz, J. and Ocenasek, J. (2001). Multiobjective bayesian optimization algorithm for combinatorial problems: Theory and practice. *Neural Network World*, 11(5):423–442.

[Serafini, 1994] Serafini, P. (1994). Simulated annealing for multi objective optimization problems. In *Multiple criteria decision making*, pages 283–292. Springer.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Sleator and Tarjan, 1985] Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686.

[Sörensen, 2015] Sörensen, K. (2015). Metaheuristics-the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18.

[Sourd and Spanjaard, 2008] Sourd, F. and Spanjaard, O. (2008). A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484.

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

[Sun and Steuer, 1996] Sun, M. and Steuer, R. E. (1996). Quad-trees and linear lists for identifying nondominated criterion vectors. *INFORMS Journal on Computing*, 8(4):367–375.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[Talbi, 2009] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

[Talbi et al., 2001] Talbi, E.-G., Rahoual, M., Mabed, M. H., and Dhaenens, C. (2001). A hybrid evolutionary approach for multicriteria optimization problems: Application to the flow shop. In *EMO*, volume 1, pages 416–428. Springer.

[Tricoire, 2012] Tricoire, F. (2012). Multi-directional local search. *Computers & operations research*, 39(12):3089–3101.

[Ulungu and Teghem, 1995] Ulungu, E. and Teghem, J. (1995). The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165.

[Ulungu and Teghem, 1992] Ulungu, E. L. and Teghem, J. (1992). Heuristic for multiobjective combinatorial optimization problems with simulated annealing. In *EURO XII conference*.

[Vanderpooten and Vincke, 1989] Vanderpooten, D. and Vincke, P. (1989). Description and analysis of some representative interactive multicriteria procedures. *Mathematical and computer modelling*, 12(10-11):1221–1238.

[Verel et al., 2011] Verel, S., Liefooghe, A., Jourdan, L., and Dhaenens, C. (2011). Analyzing the effect of objective correlation on the efficient set of mnk-landscapes. pages 116–130.

[Vianna and Arroyo, 2004] Vianna, D. S. and Arroyo, J. E. C. (2004). A grasp algorithm for the multi-objective knapsack problem. In *Computer Science Society, 2004. SCCC 2004. 24th International Conference of the Chilean*, pages 69–75. IEEE.

[Wang and Sebag, 2012] Wang, W. and Sebag, M. (2012). Multi-objective monte-carlo tree search. In *ACML*, pages 507–522.

[Wang and Sebag, 2013] Wang, W. and Sebag, M. (2013). Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search. *Machine Learning*, 92(2-3):403–429.

[While et al., 2012] While, L., Bradstreet, L., and Barone, L. (2012). A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86–95.

[Woodruff and Zemel, 1993] Woodruff, D. L. and Zemel, E. (1993). Hashing vectors for tabu search. *Annals of Operations Research*, 41(2):123–137.

[Wu et al., 2013] Wu, T.-Y., Wu, I.-C., and Liang, C.-C. (2013). Multi-objective flexible job shop scheduling problem based on monte-carlo tree search. In *Technologies and Applications of Artificial Intelligence (TAAI), 2013 Conference on*, pages 73–78. IEEE.

[Xue et al., 2013] Xue, B., Zhang, M., and Browne, W. N. (2013). Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE transactions on cybernetics*, 43(6):1656–1671.

[Zavala et al., 2014] Zavala, G. R., Nebro, A. J., Luna, F., and Coello, C. A. C. (2014). A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, 49(4):537–558.

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

[Zhang et al., 2016] Zhang, X., Tian, Y., Cheng, R., and Jin, Y. (2016). Empirical analysis of a tree-based efficient non-dominated sorting approach for many-objective optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE.

[Zhao et al., 2012] Zhao, S.-Z., Suganthan, P. N., and Zhang, Q. (2012). Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes. *IEEE Transactions on Evolutionary Computation*, 16(3):442–446.

[Zhou et al., 2011] Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P. N., and Zhang, Q. (2011). Multi-objective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49.

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

[Zitzler and Künzli, 2004] Zitzler, E. and Künzli, S. (2004). Indicator-based selection in multiobjective search. In *International Conference on Parallel Problem Solving from Nature*, pages 832–842. Springer.

[Zitzler et al., 2001] Zitzler, E., Laumanns, M., Thiele, L., et al. (2001). Spea2: Improving the strength pareto evolutionary algorithm.

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

[Zobrist, 1970] Zobrist, A. L. (1970). A new hashing method with application for game playing. *ICCA journal*, 13(2):69–73.

Recherche Locale, structures de données et Recherche Monte-Carlo pour les problèmes d'optimisation combinatoire Multi-Objectif

Résumé en français

Introduction

Les solutions de certains problèmes combinatoires peuvent être évaluées sur plusieurs objectifs, souvent contradictoires. Les conflits entre les objectifs conduisent généralement à la différence fondamentale suivante avec l'optimisation mono-objectif : deux solutions peuvent avoir des valeurs d'objectifs différentes, représentant des compromis différents des objectifs, et par conséquent, aucune n'est meilleure que l'autre. L'optimisation conjointe de plusieurs objectifs au sein d'un problème combinatoire a donné naissance au domaine de l'optimisation combinatoire multi-objectif (MOCO).

Un exemple classique d'objectifs conflictuels concerne des problèmes tels que le problème du plus court chemin ou le problème de tournée de véhicule. Pour de tels problèmes, plusieurs objectifs peuvent être envisagés : le temps de parcours, l'empreinte écologique et naturellement le coût du trajet en fonction du moyen de transport choisi. Nous pouvons facilement constater qu'un certain nombre de ces objectifs sont généralement contradictoires, comme le temps de déplacement par rapport au coût du trajet ou à l'empreinte écologique. Cependant, à titre d'exemple, il peut être intéressant pour une entreprise soucieuse de l'environnement (ou même une personne) de choisir un itinéraire optimisant à la fois le temps de parcours et son empreinte écologique. Considérer un tel objectif écologique supplémentaire serait bénéfique pour l'entreprise en promouvant son respect de l'environnement, et ce afin attirer plus de clients. Concernant le problème d'ordonnancement rencontré par toute compagnie aérienne, outre la minimisation des coûts, nous pourrions considérer comme deuxième objectif le bien-être du personnel de bord, en tenant compte de leurs préférences de jours de travail. Bien que potentiellement en conflit avec une minimisation des coûts, l'optimisation prenant en aussi compte le bien-être des employés pourrait potentiellement réduire le nombre de mouvements sociaux mais également améliorer l'image de marque de l'entreprise. Un autre exemple d'objectifs contradictoires apparaît dans un problème rencontré en finance, consistant à choisir un portefeuille parmi un ensemble de propositions d'investissement tout en optimisant deux objectifs contradictoires : la valeur attendue des rendements du portefeuille à maximiser, et le risque inhérent aux rendements du portefeuille à minimiser. Plus généralement, MOCO présente de nombreuses applications dans les domaines de la finance, du transport, de la médecine et des télécommunications, des problèmes de routage et des télécommunications et des problèmes de conception structurelle.

MOCO est une alternative intéressante pour les décideurs par rapport à l'optimisation combinatoire mono-objectif, car elle couvre différents points de vue existants pour un

même problème. En revanche, MOCO est confronté à des problèmes computationnels lorsque de multiples objectifs contradictoires sont considérés. D'ailleurs, la plupart des problèmes MO sont théoriquement difficiles, dans le sens où leur version de décision est NP-difficile, même si la version mono-objectif sous-jacente peut être résolue en temps polynomial.

Les méthodes MOCO sont classées en méthodes exactes, méthodes d'approximation avec garantie de performance et (méta-)heuristique. En plus de cette classification, trois approches principales existent dans MOCO et chacune est plus ou moins exigeante pour le décideur. La première est l'approche interactive, qui interagit itérativement avec le décideur en lui demandant ses préférences à travers la sélection ou la classification des différentes solutions suggérées par la méthode afin de guider efficacement la recherche, et d'obtenir finalement une solution qui lui convient. La seconde est l'approche a priori qui vise à d'abord interroger le décideur sur ses préférences puis à guider le processus d'optimisation en fonction de ces préférences. À la fin, la méthode produit un unique ou un petit ensemble de solutions. Ces deux types d'approches agrègent souvent les objectifs en un seul, de sorte que le problème peut alors être résolu comme problème mono-objectif. Une grande variété de fonctions d'agrégation existe dans la littérature, de la somme pondérée ou de l'agrégation Tchebychev pondérée, à des méthodes plus complexes capables de modéliser les préférences complexes du décideur comme l'intégrale de Choquet. Le troisième est l'approche a posteriori, pour laquelle les préférences du décideur ne sont pas connues a priori, c'est-à-dire non connues avant le processus d'optimisation. Cette approche est fortement reliée à la notion de dominance de Pareto : une solution domine une autre solution si elle est égale ou meilleure sur tous les objectifs et strictement meilleure sur au moins un objectif. En considérant cette notion de dominance de Pareto, le décideur ne serait pas intéressé par des solutions dominées et, par conséquent, le but d'une approche a posteriori est de trouver l'ensemble des solutions qui ne sont dominées par aucune autre solution, appelée ensemble des solutions efficaces (ou plus simplement ensemble efficace). Malgré le fait que dans la pratique nous recherchons un ensemble efficace réduit, c'est-à-dire un ensemble efficace de sorte qu'une seule solution est mémorisée parmi toutes les solutions équivalentes, l'ensemble peut être extrêmement grand, en particulier dans le cas où plusieurs objectifs conflictuels sont considérés. Ainsi, même pour des problèmes de taille modérée, il est généralement prohibitif d'identifier un ensemble efficace réduit. En particulier, plusieurs problèmes MOCO sont intractables, en ce sens que le nombre de points peut être exponentiel dans la taille de l'instance. C'est pourquoi, en plus de la difficulté inhérente des problèmes MO précédemment mentionnée, les méthodes exactes permettant de trouver un ensemble efficace réduit sont généralement utilisées pour des problèmes faciles, ou des problèmes plus difficiles de petite taille. D'autre part, étant donné une tolérance prédéfinie dans la relation de dominance, les méthodes d'approximation avec garantie de performance seront capables de trouver une approximation de l'ensemble efficace dès le moment où les méthodes exactes deviennent impraticables. Ces méthodes d'approximation semblent être efficaces en théorie comme en pratique pour certains problèmes. Enfin, les méta-heuristiques ne fournissent aucune garantie de performance sur la qualité de l'ensemble d'approximations trouvé, mais en raison de leur efficacité pratique, elles sont actuellement massivement utilisées pour optimiser des problèmes difficiles de grandes tailles, là où les méthodes d'approximation avec garantie de

performance sont dépassées. Les méta-heuristiques mono-objectif ont en fait été adaptées à MOCO, comme les algorithmes évolutionaires MO, la recherche locale MO, les méta-heuristiques MO inspirées de la nature, et plus récemment la recherche Monte-Carlo MO. Une fois que l'ensemble efficace a été trouvé (ou une approximation), le décideur choisit la solution qui correspond au mieux à ses préférences en utilisant une méthode d'aide à la décision multi-crière.

Cadre de la thèse

Dans cette thèse, nous nous intéressons à la conception de structures de données et de méta-heuristiques pour trouver l'approximation d'un ensemble efficace de problèmes MOCO. Le point principal de la thèse est de proposer des méthodes nouvelles et efficaces (en termes de temps et de qualité d'approximation), indépendantes autant que possible du problème traité, et scalables tant dans la taille que dans le nombre d'objectifs de l'instance. En effet, nous ne nous sommes pas limités au cas bi-objectif et avons considéré des problèmes avec jusqu'à 5 objectifs. De plus, les méthodes proposées sont modulaires, en le sens où elles peuvent être utilisés indépendamment les unes des autres. En raison des récents succès de la recherche locale sur des problèmes MOCO durs, nous sommes particulièrement intéressés par l'introduction de nouveaux algorithmes pour la recherche locale MO, les méthodes d'optimisation mono-objectif et les structures de données. Comme axe secondaire, compte tenu de l'émergence récente de méthodes de recherche de Monte-Carlo efficaces sur de nombreux problèmes d'optimisation mono-objectif, nous nous intéressons également à la combinaison des méthodes de recherche Monte-Carlo avec la recherche locale MO.

Plan de la thèse

Le chapitre 1 rappelle les définitions et notions MOCO fondamentales. Le chapitre 2 présente une vue d'ensemble des méthodes d'optimisation des problèmes MOCO, en mettant l'accent sur les méta-heuristiques. Le chapitre présente également l'Etat de l'art des méthodes sur le Problème du Voyageur de Commerce MO (MOTSP), en mettant l'accent sur les méta-heuristiques, puis expose l'Etat de l'art des archives, qui sont des structures de données gérant un ensemble de solutions incomparables. Le chapitre 3 traite des nouvelles archives proposées : l'AVL-Archive pour les problèmes d'optimisation bi-objectif et la NDR*-Archive pour le cas général. Les résultats expérimentaux sur les benchmarks artificiels et sur le MOTSP sont présentés. Le chapitre 4 introduit les nouvelles méthodes proposées pour optimiser les problèmes MOCO : une nouvelle méta-heuristique MO appelée 2 Phase Iterated Pareto Local Search with Decomposition (2PIPLS/D), intégrant l'Adaptive Maximally Dispersed set of Weights (A-MDW) qui permet de générer une population initiale de solutions, la Partitioned Pareto Local Search (P-PLS) comme technique d'accélération pour la PLS, et quelques modifications sur les méthodes mono-objectif afin améliorer leur efficacité sur les problèmes MOCO. Le chapitre 5 est consacré

à l'application de 2PIPLS/D au MOTSP sur un benchmark d'instances bi-objectif et tri-objectif, et propose une preuve empirique de la convexité globale pour le MOTSP. Enfin, le chapitre 6 introduit la version MO du problème de cartographie des régions françaises et montre l'application de 2PIPLS/D à ce nouveau problème à cinq objectifs.

1 Concepts fondamentaux de MOCO : Résumé

Ce chapitre rappelle d'abord la définition formelle d'un problème MOCO, puis les notions de base telles que les relations de dominance, le concept de non-dominance, les points idéal et nadir et les solutions (non) supportées. Des techniques fondamentales sont également introduites, telles que les fonctions d'agrégation et la génération de poids. Enfin, certains problèmes MOCO classiques sont introduits.

2 Méthodes de MOCO : Résumé

Ce chapitre propose une vue d'ensemble des différentes méthodes existant dans MOCO, en mettant l'accent sur les méta-heuristiques. Tout d'abord, la notion d'indicateur de qualité pour un ensemble de points est introduite, et les indicateurs utilisés pour nos futures expériences sont détaillés. Ensuite, différentes classes de méta-heuristiques sont présentées parmi lesquelles la recherche locale et la recherche Monte-Carlo sont largement décrites. En particulier, nous proposons une catégorisation claire des différentes méthodes de recherche locale MO. Enfin, est réalisé un état de l'art des méthodes d'optimisation pour le MOTSP ainsi que sur les archives.

3 Nouvelles archives : Résumé

Nous avons proposé deux nouvelles archives, l'AVL-Archive spécialisée dans le cas bi-objectif, et la NDR *-Archive pour le cas général. Les deux structures sont auto-équilibrées et ont deux versions : une version générale adaptée à toute tâche d'optimisation MO ; et une version adaptative, spécialement conçue si une présomption de localisation temporelle et spatiale existe entre les solutions présentées à l'archive.

Ces archives ont été testées expérimentalement et comparées aux archives de l'état de l'art sur un grand nombre d'instances simulant la génération de points d'une méta-heuristique MO, et embarquées à l'intérieur de la recherche locale Pareto (PLS) appliquée au MOTSP avec jusque 5 objectifs.

L'AVL-Archive surpasse toutes les autres archives dans le cas bi-objectif et sa version adaptative fonctionne légèrement mieux que la liste triée lorsqu'elles est utilisée au sein de la PLS.

NDR * -Archive fonctionne mieux que les concurrents dans le cas général, en particulier lorsque 4 objectifs ou moins sont considérés et que le nombre de points dominés est plus grand que le nombre de points non dominés. La version adaptative de NDR*-Archive est beaucoup plus efficace que ses concurrents lorsqu'elle est embarquée dans la PLS, en particulier quand 3 ou 4 objectifs sont considérés.

Nous espérons que ces deux archives et leurs versions adaptatives pourront être utiles pour de futures méta-heuristiques, en particulier celles qui utilisent PLS, la programmation dynamique MO ou les méthodes basées sur les zones de recherche.

Concernant les perspectives de la NDR*-Archive, on pourrait trier les fils d'un nœud interne en utilisant leur clé de Hilbert (ou n'importe quelle clé provenant d'une courbe de remplissage, comme la courbe de Peano). Comme l'AVL-Archive qui utilise des propriétés spécifiques au cas bi-objectif, il pourrait être intéressant d'utiliser les propriétés spécifiques des espaces de dimensions fixes de tailles 3, 4 ou 5. D'autres recherches semblent nécessaires pour améliorer l'efficacité de NDR*-Archive dans le cas où 5 objectifs ou plus sont considérés.

Une amélioration possible de l'AVL-Archive est d'utiliser les mouvements de type splay pour réorganiser l'arborescence, afin que le nœud en cache soit placé à la racine de l'arbre. Cependant, cette modification pourrait rendre l'arbre déséquilibré.

4 Nouvelles méthodes d'optimisation : Résumé

Nous avons proposé dans ce chapitre un certain nombre d'améliorations de méthodes existantes et de nouvelles méthodes d'optimisation pour s'attaquer plus efficacement aux problèmes MOCO.

Nous avons tout d'abord suggéré trois modifications des méthodes d'optimisation mono-objectif pour améliorer leur capacité à trouver des solutions efficaces dans les problèmes MOCO : démarrer/guider systématiquement la recherche à partir d'une solution déjà trouvée, mémoriser les solutions générées pendant le processus d'optimisation et activer la perturbation des données. Sur la base de ces spécifications, nous avons proposé une nouvelle version de Nested Monte-Carlo Search (NMCS), appelée A-NMCS, mieux adaptée au MOCO que la version originale de NMCS.

Deuxièmement, nous avons présenté une nouvelle méthode généralisant l'algorithme Maximally Dispersed set of Weights, appelé A-MDW et le rendant plus adaptatif de sorte que la méthode continue de générer des poids jusqu'à ce que la méthode optimisant les problèmes de somme pondérée associés ne soit plus efficace.

Troisièmement, nous avons introduit la recherche locale Pareto partitionnée (P-PLS), qui vise à accélérer la recherche local Pareto avec descente variable (PLS-VND) à travers le partitionnement de l'ensemble de solutions à explorer, et une restriction intelligente de la

structure de voisinage de PLS-VND basée sur la présomption de convexité globale sur le problème traité.

Ensuite, nous avons proposé une nouvelle méta-heuristique MO appelée 2-Phase Iterated Pareto Local Search (2PIPLS/D) basée sur trois concepts importants : les méta-heuristiques à 2 phases, les méthodes de décomposition et la recherche locale Pareto itérée, et combinant A-MDW, P-PLS, ainsi que les modifications suggérées pour les méthodes d'optimisation mono-objectif.

Enfin, un nouveau système basé sur le concept d'ϵ-archive est présenté, permettant de limiter la taille d'une archive tout en garantissant une bonne répartition des points dans l'espace objectif.

5 Application de 2PIPLS/D au MOTSP : Résumé

Dans ce chapitre, nous avons d'abord proposé une implémentation des différents composants algorithmiques de 2PIPLS/D, en particulier nous avons proposé d'utiliser le Chained Lin Kernighan (C-LK) et l'A-NMCS comme méthodes d'optimisation mono-objectif, et suggéré une implémentation de la structure de restriction de voisinage de P-PLS. Ensuite, nous avons présenté une preuve empirique de convexité globale sur le MOTSP afin de légitimer l'utilisation du système de partitionnement de P-PLS. Nous avons constaté que les solutions (potentiellement) efficaces sont effectivement concentrées dans l'espace de décision et que les solutions (potentiellement) efficaces dans l'espace des objectif ont tendance à être également voisines dans l'espace de décision. Après une analyse de sensibilité de 2PIPLS/D sur ses paramètres, nous avons classé les différents paramètres en fonction de leur influence sur la performance de nôtre méthode. Nous avons constaté que 2PIPLS/D est sensible au nombre de problèmes de somme pondérée optimisés lors de l'initialisation avec A-MDW, mais aussi à la taille maximale autorisée d'une partie (de la partition générée) lors de l'initialisation de P-PLS. En particulier, nous avons noté que la P-PLS réduit drastiquement le nombre de solutions examinées par PLS-VND (-98% en moyenne sur les instances testées) grâce à son système de partitionnement. En outre, la mémorisation des solutions au cours de l'exécution d'une méthode mono-objectif semble être très efficace, à la fois pour C-LK et A-NMCS.

D'un autre côté, 2PIPLS/D ne semble pas particulièrement sensible à la perturbation des données, à la taille maximale de la structure du voisinage et aux différentes stratégies et modes d'exploration utilisés par PLS-VND. Enfin, nous avons comparé 2 versions de 2PIPLS/D (soit avec C-LK amélioré ou bien avec A-NMCS) avec les meilleures méthodes actuelles sur un grand nombre d'instances bi-objectif et tri-objectif.

2PIPLS/D (avec C-LK) obtient de bons résultats et trouve l'ensemble non-dominé de 7 instances bi-objectif de taille 100 en 20 runs, de sorte que 2PIPLS/D est, à notre connaissance, la première méta-heuristique capable de trouver un ensemble non-dominé

d'instances MOTSP d'une telle taille. 2PIPLS/D (avec C-LK) surpasse MoMad et PD-TPLS sur toutes les instances testées ; 2PIPLS/D surpasse également PDA sur 95% des instances testées et a des performances équivalentes pour les 5% d'instances restantes.

Enfin, 2PIPLS/D avec C-LK surpasse dans une grande majorité des cas 2PIPLS/D avec A-NMCS, ce qui renforce le fait que les performances 2PIPLS/D dépendent fortement de l'efficacité de la méthode mono-objectif utilisée.

Concernant les perspectives, il pourrait être intéressant de mettre en place des structures de voisinage (dans PLS-VND) avec une plus grande taille que le 3-exchange, ou mieux encore, de proposer des mouvements de type variable k-exchange comme dans Lin-Kernighan. Cependant, nous avons deux principales limitations à propos de cette perspective. Tout d'abord, les techniques qui ont rendu les méthodes de recherche locale si puissantes pour le TSP, comme les don't look bits ou les listes de candidats, ne sont pas aussi efficaces pour le MOTSP. Deuxièmement, il semble que l'étape de perturbation globale de 2PIPLS/D soit suffisamment efficace pour compenser une petite taille de structure de voisinage de PLS-VND.

Une deuxième perspective est l'application de 2PIPLS/D sur des instances plus importantes, à la fois en termes de tailles et de nombre d'objectifs. Selon nous, la principale limitation concernant cette proposition est que 2PIPLS/D, comme toute méthode basée sur le framework de la PLS à deux phases, sont très sensibles à la méthode d'optimisation mono-objectif utilisée pour (ré)démarrer la PLS. En effet, les méthodes d'optimisation mono-objectif basées sur l'algorithme de Lin-Kernighan semblent être la meilleure option que nous ayons actuellement, mais ne sont pas si efficaces sur les instances dans lesquelles il n'y a pas d'inégalité triangulaire. Par conséquent, des difficultés sont attendons pour ce type de méthodes sur des instances aléatoires, bi-objectif ou tri-objectif de taille 1000, ou sur des instances de grandes tailles considérant 4 objectifs ou plus.

6 Application de 2PIPLS/D au MOFRMP : Résumé

Dans ce chapitre, nous nous intéressons à la récente réforme territoriale des régions françaises qui s'est traduite par l'affectation des départements à de nouvelles régions plus grandes. A partir de l'observation que certains départements sont plus attirés par d'autres régions que leur région d'affectation actuelle, le but de ce chapitre était de tenter d'identifier ces départements en optimisant un nouveau problème multi-objectif appelé Problème de Cartographie des Régions Françaises MO (MOFRMP). Le problème consiste à trouver des cartes alternatives en optimisant cinq objectifs, basés sur des indicateurs économiques et financiers mesurant la force des interactions entre les départements affectés à une même région, ainsi que les poids économiques et démographiques de chaque région. Nous avons appliqué 2PIPLS/D à ce problème et testé la recherche locale itérée (ILS) et A-NMCS comme méthodes d'optimisation mono-objectif. Nous avons constaté que 2PIPLS/D avec ILS surpasse 2PIPLS/D avec A-NMCS. Une approximation de l'ensemble efficace a été générée avec les différentes exécutions de 2PIPLS/D. Cependant, nous avons trouvé que

les cartes (potentiellement) efficaces trouvées ont beaucoup trop de départements de différence avec la carte actuelle, ce qui rend irréaliste de les proposer comme alternative à la carte actuelle. Au lieu de cela, nous avons construit un indicateur basé sur l'approximation des solutions efficaces trouvées, qui vise à identifier les départements les plus attirés par d'autres régions que leur région courante à laquelle ils appartiennent. En particulier, nous avons constaté que le Gard (30), la Haute-Marne (52) et les Deux-Sèvres (79) sont particulièrement attirés par d'autres régions.

Le MOFRMP soulève une question importante : la carte actuelle des régions pourrait-elle être améliorée ? Le travail proposé dans ce chapitre correspond seulement à une réponse préliminaire à cette question. Nous pensons qu'une nouvelle formulation du MOFRMP est nécessaire pour obtenir des cartes alternatives réalistes. Premièrement, il nous semble important de ne pas limiter ce problème aux seuls critères économiques et financiers. A titre d'exemple, les échanges culturels entre départements pourraient être envisagés, ou l'intensité globale des mouvements de personnes entre les départements, et pas seulement les trajets domicile-travail. Deuxièmement, il est impératif d'obtenir les préférences des décideurs quant à l'importance relative des différents critères considérés. Troisièmement, nous pensons que toute carte alternative doit proposer un nombre très limité de réaffectations de départements par rapport à la carte actuelle. Si le nombre maximum de réaffectations est limité à 3 ou 4 par exemple, le problème devient assez facile et il semble qu'une méthode d'optimisation ne soit pas nécessaire. En effet, dans ce cas, une simple énumération des solutions est suffisante, puis un filtrage pour ne garder que les solutions efficaces, et enfin l'utilisation d'une méthode d'aide à la décision multi-critère en considérant les préférences des décideurs préalablement obtenues. En revanche, si un nombre plus élevé de réaffectations est accepté (c'est-à-dire 5 ou plus), le nombre de solutions réalisables semble trop important pour une simple énumération. Dans ce cas, une méthode d'optimisation MO est nécessaire et 2PIPLS/D pourrait être adapté à cette nouvelle formulation du MOFRMP.

Conclusion

S'inscrivant dans le cadre de l'optimisation combinatoire multi-objectif (MOCO), cette thèse a proposé un certain nombre de nouvelles méthodes d'approximation et de structures de données pour les problèmes de MOCO de grande taille. Tout d'abord, les contributions de la thèse sont résumées, puis un certain nombre de perspectives sont suggérées.

Contributions

1. Nous avons proposé deux nouvelles archives : l'AVL-Archive pour le cas bi-objectif et la NDR *-Archive pour un nombre quelconque d'objectifs ; ainsi que leurs versions adaptatives utilisant les concepts de localisation temporelle et spatiale, et donc spécialement conçues pour la PLS. Nous avons comparé ces archives aux

meilleures archives connues et nous obtenons de meilleurs résultats que les concurrents, faisant de l'AVL-Archive et de la NDR * -Archive et leurs versions adaptatives, les archives les plus efficaces jusqu'à 5 objectifs.

2. Nous avons présenté une nouvelle méthode généralisant le concept de MDW en le rendant plus adaptatif de sorte que la méthode continue de générer des poids jusqu'à ce que la méthode d'optimisation mono-objectif optimisant les problèmes de somme pondérée associés ne soit plus efficace.
3. Nous avons introduit la recherche locale Pareto partitonnée (P-PLS), qui vise à accélérer PLS-VND au travers d'un partitionnement de l'ensemble de solutions à explorer, et une restriction intelligente de la structure de voisinage de PLS-VND basée sur la présomption de convexité globale sur le problème traité.
4. Nous avons suggéré quelques modifications pour tout optimiseur mono-objectifs afin d'améliorer sa capacité à trouver des solutions efficaces dans les problèmes MOCO, en particulier en mémorisant les solutions générées au cours du processus d'optimisation. Sur la base de ces spécifications, nous avons proposé une nouvelle version de NMCS, appelée A-NMCS, mieux adaptée aux problèmes MOCO que NMCS.
5. Nous avons présenté une nouvelle méta-heuristique appelée 2PIPLS/D. Au cours de la première phase, A-MDW est utilisé comme méthode d'initialisation et P-PLS construit une partition pour accélérer PLS-VND. A chaque itération de la seconde phase, un ensemble de solutions est tout d'abord généré par l'optimisation de plusieurs problèmes de somme pondérée perturbés, puis PLS-VND est lancé à partir de cet ensemble. 2PIPLS/D gère différents modes et stratégies d'exploration pour PLS-VND, et intègre un système d'ϵ-archive qui permet une bonne distribution dans l'espace des objectifs des points générés et limite la taille de l'approximation.
6. Nous avons implémenté 2PIPLS/D pour le MOTSP. La méthode a été comparée aux méthodes de l'Etat de l'art sur des instances du TSP à deux objectifs et trois objectifs. Les résultats expérimentaux montrent que 2PIPLS/D surpasse ses concurrents sur toutes les instances testées, à l'exception de 5% d'entre elles où notre propre méthode PDA proposée dans un pécédent article et 2PIPLS/D ont des performances similaires. P-PLS fournit une performance étonnamment bonne car il réduit le nombre de solutions explorées de 98% en moyenne, comparé à une absence de partitionnement. Dans la continuation des travaux de Borges et Hansen, nous avons confirmé expérimentalement l'hypothèse de convexité globale pour le MOTSP sur un certain nombre d'instances bi-objectif et tri-objectif. Dans la suite des travaux de Lust et Teghem, nous avons constaté que les arêtes présentes dans les solutions efficaces sont de bonne qualité par rapport aux autres arêtes.
7. Nous avons implémenté 2PIPLS/D pour un nouveau problème lié à la récente réforme territoriale des régions françaises, le FRMP, pour lequel nous avons proposé une formulation MO avec 5 objectifs. Nous avons détecté un certain nombre de départements plus attirés par une région que leur région à laquelle ils appartiennent actuellement.
8. Pour le MOTSP comme pour le MOFRMP, 2PIPLS/D avec A-NMCS est surpassé dans la plupart des cas par 2PIPLS/D avec une méthode basée sur la recherche locale.

Perspectives à court et moyen termes

2PIPLS/D pourrait être testé sur d'autres problèmes MOCO. Le comportement du système de partitionnement de P-PLS sur des instances avec des objectifs corrélés négativement nous intéresse particulièrement. Car cela rend beaucoup plus difficile l'optimisation par les différentes méthodes de recherche locale dû à l'impact sur le niveau de clusterisation des solutions de telles instances.

En ce qui concerne PLS-VND en mode indépendant, il pourrait être intéressant de mettre en place un mécanisme tabu afin de ne pas réexaminer les solutions déjà explorées lors des runs précédents.

Perspectives à long terme

Concernant 2PIPLS/D, au lieu d'utiliser PLS(-VND), il pourrait être intéressant d'utiliser une méthode d'optimisation combinatoire exacte. En fait, la méthode de partitionnement introduite dans P-PLS est indépendante du concept de PLS (elle a en effet été initialement conçue comme telle). Par conséquent, l'idée consiste à exécuter un algorithme de Branch et Bound MO ou un algorithme de programmation dynamique MO (par exemple) à partir de l'espace de recherche réduit fourni par le système de partitionnement. Notez que les nouvelles archives que nous avons proposées seraient d'un grand intérêt si une méthode de programmation dynamique MO est utilisée. Par exemple, pour le MOTSP, on pourrait utiliser une version MO de la méthode de programmation dynamique basée sur la branch-decomposition de Cook et Seymour. Pour le MO 0-1 Knapsack Problem, la méthode utilisée pourrait être la méthode de Programmation Dynamique MO de Bazgan et al.. Nous croyons que cette direction de recherche est prometteuse étant donnée la grande réduction de l'espace de recherche permise par le système de partitionnement obtenue sur le MOTSP.

Résumé

De nombreux problèmes d'optimisation combinatoire considèrent plusieurs objectifs, souvent conflictuels. Cette thèse s'intéresse à l'utilisation de méthodes de recherche locale, de structures de données et de recherche Monte-Carlo pour la recherche de l'ensemble des solutions efficaces de tels problèmes, représentant l'ensemble des meilleurs compromis pouvant être réalisés en considération de tous les objectifs.

Nous proposons une nouvelle méthode d'approximation appelée 2-Phase Iterated Pareto Local Search based on Decomposition (2PIPLS/D) combinant les concepts de recherche locale Pareto (PLS) et de décomposition. La PLS est une descente de recherche locale adaptée au multi-objectif, et la décomposition consiste en la subdivision du problème multi-objectif en plusieurs problèmes mono-objectif. Deux méthodes d'optimisation mono-objectif sont considérées: la recherche locale itérée et la recherche Monte-Carlo imbriquée. Deux modules principaux sont intégrés à 2PIPLS/D. Le premier généralise et améliore une méthode existante et génère un ensemble initial de solutions. Le second réduit efficacement l'espace de recherche et permet d'accélérer la PLS sans négliger la qualité de l'approximation générée. Nous introduisons aussi deux nouvelles structures de données gérant dynamiquement un ensemble de solutions incomparables, la première est spécialisée pour le cas bi-objectif et la seconde pour le cas général.

2PIPLS/D est appliquée au Problème du Voyageur de Commerce bi-objectif et tri-objectif et surpasse ses concurrents sur les instances testées. Ensuite, 2PIPLS/D est appliquée à un nouveau problème avec cinq objectifs en lien avec la récente réforme territoriale d'agrandissement des régions françaises.

Abstract

Many Combinatorial Optimization problems consider several, often conflicting, objectives. This thesis deals with Local Search, data structures and Monte Carlo Search methods for finding the set of efficient solutions of such problems, which is the set of all best possible trade-offs given all the objectives.

We propose a new approximation method called 2-Phase Iterated Pareto Local Search based on Decomposition (2PIPLS/D) combining the notions of Pareto Local Search (PLS) and Decomposition. PLS is a local search descent adapted to Multi-Objective spaces, and Decomposition consists in the subdivision of the Multi-Objective problem into a number of Single-Objective problems. Two Single-Objective methods are considered: Iterated Local Search and Nested Monte Carlo Search. Two main components are embedded within the 2PIPLS/D framework. The first one generalizes and improves an existing method generating an initial set of solutions. The second one reduces efficiently the search space and accelerates PLS without notable impact on the quality of the generated approximation. We also introduce two new data structures for dynamically managing a set of incomparable solutions. The first one is specialized for the bi-objective case, while the second one is general.

2PIPLS/D is applied to the bi-objective and tri-objective Traveling Salesman Problem and outperforms its competitors on tested instances. Then, 2PIPLS/D is instantiated on a new five-objective problem related to the recent territorial reform of French regions which resulted in the reassignment of departments to new larger regions.

Mots Clés

Optimisation combinatoire multi-objectif
Méta-heuristique
Recherche locale
Structures de données
Décomposition
Recherche Monte-Carlo
Problème du Voyageur de Commerce

Keywords

Multi-objective combinatorial optimization
Meta-heuristic
Local search
Data structures
Decomposition
Monte Carlo search
Traveling Salesman Problem