Université de Lille
CRIStAL, UMR 9189
École Doctorale MADIS-631
Mathématiques, Sciences du numérique et leurs applications

Extraction de connaissance pour les métaheuristiques à partir des solutions appliquée aux problèmes de tournées de véhicules bi-objectif

Solution-based Knowledge Discovery in Metaheuristics for Bi-Objective Vehicle Routing Problems

Clément LEGRAND LIXON

Thèse préparée et soutenue publiquement le ***24 Septembre 2024*** *en vue de l'obtention du grade de* **Docteur en Informatique et Applications**

Membres du Jury:

Diego CATTARUZZA (*MCF*)	Centrale Lille	Co-encadrant
Jin-Kao HAO (*PR*)	Université d'Angers	Rapporteur
Lhassane IDOUMGHAR (*PR*)	Université de Haute-Alsace	Examinateur
Laetitia JOURDAN (*PR*)	Université de Lille	Co-directrice
Nicolas JOZEFOWIEZ (*PR*)	Université de Lorraine	Rapporteur
Marie-Éléonore KESSACI (*PR*)	Université de Lille	Co-directrice
Caroline PRODHON (*PR*)	Université Technologique de Troyes	Examinatrice
Jean-Stéphane VARRÉ (*PR*)	Université de Lille	Examinateur (Président)

Acknowledgements

At first I would like to thank all the jury members and especially Laetitia Jourdan, Marie-Eléonore Kessaci, and Diego Cattaruzza for their support during my PhD, but also during the internships I performed along my studies. I remember well our first collaboration six years ago, when I discovered for the first time the world of research. At that time, I did not know anything about metaheuristics and combinatorial optimization, but I quickly liked these topics, and it gave me the taste of research. Many thanks to Jean-Stéphane Varré, Lhassane Idoumghar, and Caroline Prodhon for accepting to be part of my jury, and to Jin-Kao Hao and Nicolas Jozefowiez for also reporting my manuscript; I am grateful for the interest you gave to my work.

I would like to thank also all the members of the ORKAD team. You all contribute to the unforgettable atmosphere of the laboratory, and everyone is involved in the life of the team, making it a great working space. It was a real pleasure to go to my office every day, and it undoubtedly positively contributed to the work I realized in the team. Many thanks to the administrative staff of CRIStAL, contributing to the working of the laboratory, and making the research life a bit easier.

I would not be here without the formation received at the ENS Rennes, specialized in both research and teaching, in particular in computer science. I really enjoyed the years spent there, following courses given on various topics, which contributed to the discovery of several aspects of the computer science. I am also grateful to all the teachers who prepared me to obtain the aggregation in mathematics and computer science. This brought to me more than I could have expected.

I thank all my friends who supported me during my PhD but also along my studies, for all the good time spent together. Special thanks to Victor, Arnaud, and Guillaume, friends from the ENS Rennes, with whom holidays are eventful and incredible. Special thanks also to Thomas, Agathe, Meyssa and Antoine, friends and former PhD students of the ORKAD team, with whom I shared so many good memories during my PhD, and especially during our “peak productivity” of the week: Wednesday afternoon.

My final thanks go to my family who have supported me for already 26 years. I hope this will continue for as long as possible. Thanks to my parents, who believed in my projects and without whom I would not be here. Many thanks also to my grandparents, the pillars of my childhood who have contributed to making me who I am today. To Florence, my godmother, and Raphael, her husband, for all the events and memories we share together. To Eric, my godfather, and his family, with whom I always have a good time whenever possible. Finally, thanks to my two sisters, Léa and Camille, and my brother, Antoine, for everything we share everyday.

Contents

Introduction

This thesis entitled "Solution-based Knowledge Discovery in Metaheuristics for Bi-Objective Vehicle Routing Problems" focuses on the integration of learning mechanisms, extracting knowledge from solutions, within existing multi-objective algorithms. This thesis was conducted within the ORKAD (Operational Research, Knowledge, And Data) team, in collaboration with the INOCS (Integrated Optimization with Complex Structure) team, at the CRIStAL laboratory in Lille. Both teams belong to OPTIMA, a large group of CRIStAL, experts in solving optimization problems with real-life applications. The ORKAD team develops new algorithms to solve combinatorial optimization problems by combining established resolution methods from operational research with artificial intelligence techniques. For example, by using data-mining techniques to understand the behavior of the different components of an algorithm, or to focus the exploration of the algorithm on promising regions from past explorations. In particular, the area of expertise of the ORKAD team encompasses various domains, including single and multi-objective combinatorial optimization problems, algorithm configuration, landscape analysis, and graph theory. The INOCS team models and develops innovative solution methods for complex structure problems according to three types of optimization paradigms: mathematical optimization, bilevel optimization, and robust/stochastic optimization. Complex structure problems are pervasive. In particular, they appear in the energy sector and supply chain management.

The machine learning field has become incredibly popular in the last decade. Machine learning can take different forms, depending on the task performed. For example, the learning can be supervised or unsupervised, it can be performed online or offline, and it can be based on linear regressions, decision trees, clustering, neural networks, or reinforcement learning among other mechanisms. In particular, machine learning is known to share a good synergy with combinatorial optimization problems. Such problems aim to optimize one or several objective functions, by exploring a discrete (but often huge) solution space, which can not be entirely explored in practice. Hence, it requires using heuristics (and more generally metaheuristics, which are generic optimization strategies) to quickly approximate an optimal solution of the problem. Metaheuristics aim to efficiently explore the solution space with different strategies, mixing diversification (where new regions of the space are explored), and intensification (where the focus is made on a small region to improve the solutions).

The notion of knowledge discovery covers the field of machine learning exploiting data that can be extracted from solutions (and from the problem) to guide metaheuristics during the exploration of the solution space. The knowledge discovery can be integrated at different levels in metaheuristics. At a problem-level, the knowledge is extracted from instances of the problem to characterize them and this knowledge can be used to adapt the strategies employed in the metaheuristic. At a low-level, the knowledge is extracted from solutions to the problem, in order to better understand the structure of good-quality solutions, and/or to better understand the structure of the solution space. The knowledge can also be integrated at a high-level, meaning that feedback from other heuristics is available for that problem, which can help the metaheuristic to select the next heuristic to apply.

Multi-objective optimization problems aim to simultaneously optimize several objective functions. Considering more than one objective function is interesting to understand how the different objectives impact the solutions found, and more precisely, to understand how the objective functions interact together during optimization. In general, solving a multi-objective problem helps to better understand the problem considered. However, in a multi-objective context, some solutions are incomparable, meaning that they present different trade-offs between the optimized objectives, but one solution can not be considered globally better than the other. We say that one solution dominates another one whenever its evaluation of all objectives is better than those of the other solution. The goal of solving a multi-objective problem is to find all non-dominated solutions (i.e., solutions for which there does not exist any solution dominating them). Consequently, it requires manipulating a set of solutions instead of a single solution during the execution of the algorithm. Usually, the objectives to optimize are in conflict with each other, meaning that they can not be optimized simultaneously, forcing the presence of trade-off solutions.

Integrating machine learning components into existing metaheuristics has led to significant improvements (both in terms of performance and execution time) in single and multi-objective optimization. More precisely, learning from the structure of solutions is known to be efficient in a single-objective context, since all high-quality solutions optimize the same objective. However, in a multi-objective context, learning from the structure of solutions generated during the execution remains a challenge due to the optimization of conflicting objectives. In particular, there exists high-quality solutions for each objective, and a structure that is interesting for optimizing one objective, may not be interesting for optimizing another one.

In this thesis, we focused on a vehicle routing problem with time windows (VRPTW), which finds its application in logistics. Its aim is to determine optimized routes to deliver parcels to customers during a precise time period, symbolized by a time window. Solving this type of problem is a challenge for many companies. While many different objectives can be optimized in a VRPTW, we decided to minimize the two following objectives: the total cost of transport and the total waiting time for the delivery driver. The waiting time is caused by the driver arriving before the start of the delivery period. Using both of these objectives is interesting in many real-life situations, like food delivery, where the waiting time of a driver impacts the heat of the meals of the next customers, and consequently customer satisfaction. Another typical situation concerns the transportation of people, more precisely when a patient has a medical appointment, we do not want them to wait too long. To solve this problem, we propose to exploit the sequences of customers delivered consecutively within a tour. These sequences are extracted from generated solutions during algorithm execution. The most promising sequences are then integrated into other solutions to improve them. While learning sequences to solve this type of problem has proved effective in single-objective settings, it remains a challenge to exploit them in a multi-objective context. Indeed, some sequences that are interesting for one objective may prove useless for the other. More specifically, in this thesis, we are looking at the best ways to exploit the sequences available in solutions. In particular, this led us to ask the following questions: how to manage these sequences in a multi-objective solver? From which solutions should we extract sequences, and into which solutions should we inject them? At what point in the runtime should the injection and extraction steps be carried out? The answers to these questions led us to develop a learning model exploiting solution sequences in a multi-objective context, where knowledge groups are created to store sequences related to a part of the search space. This model was then integrated into two popular algorithms: a multi-objective evolutionary algorithm based on decomposition (MOEA/D) and a multi-objective local search MOLS, demonstrating the effectiveness of the proposed model.

This manuscript is divided into three parts. The first part (Chapter 1 and Chapter 2) introduces the necessary scientific context and background to understand the work realized in this thesis. Moreover, the VRPTW is formally introduced, as well as the objectives considered. The second part (Chapters 3 to 5) is dedicated to the development of a knowledge discovery mechanism integrated into MOEA/D. This mechanism is based on the construction of knowledge groups, each associated with a region of the objective space. A knowledge group gathers sequences obtained from solutions belonging to the associated region of the objective space. The third part (Chapters 6 and Chapter 7) develops a more generic knowledge discovery mechanism, namely Solution-based Knowledge Discovery (SKD), which is integrated into two algorithms a MOEA/D and a MOLS. In particular, the integration of SKD is facilitated thanks to the proposition of a unified view between multi-objective evolutionary algorithms and multi-objective local search. Finally, we conclude this thesis by summarizing the different contributions obtained and drawing several research perspectives. In particular, we propose perspectives to improve the knowledge groups (to make them more adaptive to the instance solved), to improve the knowledge learned (through the update of scores), and to better exploit what is learned.

Chapter 1 Chapter 1 presents the scientific context of the thesis. Several themes are developed, like combinatorial optimization, multi-objective optimization, and knowledge discovery. The main concepts related to combinatorial optimization are introduced, like the notion of objective function, global and local optima, and neighborhood operators. The two most commonly used families of metaheuristics to solve combinatorial problems are presented: the family of local search algorithms, exploiting neighborhood of solutions, and the family of evolutionary algorithms. Then, multi-objective combinatorial optimization problems are formalized. In particular, the dominance relation between solutions is introduced, and the Pareto front, which is a set of incomparable solutions (i.e., representing different trade-offs between the objectives). We provide some tools to assess the quality of the Pareto fronts returned by the algorithms. Among the most popular quality indicators, we find the hypervolume and the generational distance. Different strategies used to solve multi-objective problems are also described. Following that, knowledge discovery mechanisms are introduced, and details about their integration into metaheuristics are provided. In particular, we give examples of integration at problem-, low- and high-level. Finally, the vehicle routing problem with time windows is formally introduced. An overview of the existing objectives that can be considered to solve the problem is presented, leading to our choice concerning the two objectives optimized. We also describe existing algorithms to solve this problem with existing knowledge discovery mechanisms.

Chapter 2 Chapter 2 formalizes the bi-objective vehicle routing problem with time windows considered throughout this thesis. Solutions to this problem can be represented as permutations of customers, but it requires the use of the *split* algorithm to evaluate the solutions. Common neighborhood operators are also defined and will be used in the different local searches performed. On the other hand, we motivate our choice of objectives by presenting the most common ones that are optimized for this problem. In addition, we present an overview of the existing works for the multi-objective VRPTW. Finally, we focus on a knowledge discovery mechanism, called Pattern Injection Local Search (PILS), developed for another routing problem. However, with a few modifications, it can be applied to our problem.

Chapter 3 Chapter 3 presents the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D). The idea is to decompose the multi-objective problem into several single-objective problems by aggregating the different objectives. There exists different strategies to aggregate the objectives, but a common way is to define a weighted sum of the objectives. Each weight vector produces a single-objective problem that is solved by using single-objective optimization strategies (e.g., a local search). An optimal solution for a single-objective problem generated produces a solution for the optimal Pareto front. Each subproblem is associated with its current best solution found and genetic mechanisms are used to improve and generate new solutions during the execution. A discussion about its main components, and how they have been improved (using learning mechanisms or not) is also provided.

Chapter 4 Chapter 4 focuses on our first version of the knowledge discovery mechanism for multi-objective optimization. It introduces the notion of knowledge groups, gathering sequences of solutions belonging to the same region of the objective space. In MOEA/D, we decided to define a knowledge group per subproblem. However, putting together the structure of close solutions (i.e., solutions with close objective values) requires that close solutions share similar structures. Thus, we provide details about the structural similarity between solutions found when solving instances of the VRPTW. Then, we propose new neighborhood exploration strategies to define a local search adapted to the resolution of our problem. In fact, using a local search has the advantage of generating more interesting solutions during the exploration, and it allows a faster convergence towards local optima. Finally, we analyze the benefit of extracting sequences from local optima only, since local optima generally contain more interesting structural information. This study shows that learning from local optima improves the quality of the solutions returned.

Chapter 5 Chapter 5 describes an enhanced version of our knowledge discovery mechanism introduced in Chapter 4, making it more independent from MOEA/D, but still integrated into MOEA/D only. The enhanced version associates a representative (being a weight vector) to each knowledge group, delimiting a specific region of the objective space. The main difference with the former construction is the increased control we have over the groups: we can control the number of groups defined and the region associated with a group. Moreover, intensification and diversification strategies are introduced for extraction and injection procedures. Finally, we perform an analysis of different parameters used by MOEA/D and our knowledge discovery mechanism by using the irace tool.

Chapter 6 To generalize the knowledge discovery mechanism introduced in the former chapters, it is necessary to better understand what are the main components of a multi-objective algorithm. Thus, in Chapter 6, we decide to recall the main components of multi-objective evolutionary algorithms (MOEA) and to give more details about multi-objective local search (MOLS). In particular, a unification of MOLS is presented, to highlight its main components. The comparison of the main components of MOEA and MOLS reveals that they both use similar intensification (generally a local search) and diversification (perturbation) mechanisms when exploring the search space. Moreover, they both use three main sets of solutions during the search: one keeps the best solutions found, another one maintains a current population from which solutions are selected, and the last one manages the selected solutions during their exploration. These considerations led us to propose a unified view of MOEA and MOLS, in order to facilitate the integration of knowledge discovery mechanisms in different multi-objective algorithms.

Chapter 7 In Chapter 7, we develop a solution-based knowledge discovery (SKD) model. The SKD model is based on three main components, being the creation of the knowledge groups, the extraction, and the injection procedures. In particular, a new construction strategy for the knowledge groups is introduced, which adapts to the current Pareto front, and more generic extraction and injection procedures are described. The SKD model is then instantiated into a MOEA/D and a MOLS, by using the unified view proposed in Chapter 6. Finally, the performances of the algorithms are evaluated, showing the efficiency of our knowledge discovery mechanism, and an additional discussion of the SKD model is provided.

Part I

Scientific Context

Chapter 1

Metaheuristics and Knowledge Discovery

Contents

1.1 Introduction

This chapter gathers all the concepts required to clearly understand the scope of the thesis. Several elements related to combinatorial optimization, machine learning, and routing problems are covered.

Many real-life problems (like logistic problems) can be modeled as combinatorial optimization problems. In Section 1.2, the notions of combinatorial optimization and metaheuristics (i.e., algorithmic strategies adapted to the resolution of combinatorial problems) are introduced. In particular, the concepts of local search and evolutionary algorithms are developed.

When one objective function is not enough to accurately model the problem to solve, a possible extension is to consider several objective functions. *Multi-objective* combinatorial problems are defined in Section 1.3. One interest in solving a multi-objective problem is to provide to a decision-maker, a set of *non-dominated* solutions representing different trade-offs between the objectives. However, solving a multi-objective problem is more difficult than solving a single-objective problem, and it requires the use of specific approaches. To compare the sets of solutions returned, many quality indicators have been developed but we focus on the most common ones (e.g., hypervolume, generational distance, and epsilon indicator). In addition, the most frequent approaches used to solve multi-objective problems are described.

The field of machine learning has received much interest in the last few years. In particular, recent works try to integrate machine learning elements in existing algorithms to improve their adaptability to new problems. Using machine learning mechanisms can also help to focus on more promising regions of the space of solutions, which may guide the algorithm towards more interesting solutions. Section 1.4 presents the taxonomy of knowledge discovery (i.e., machine learning at the service of optimization) and existing works in that field.

1.2 Combinatorial Optimization and Metaheuristics

Optimization is omnipresent in our lives. Every day, we optimize the time to go to work or school. We plan events, trying to limit the cost or find trade-offs to respect a given budget. In mathematics, the optimization field has received high interest for centuries. Thanks to the progress made in that field, and the development of computer science, many tools are today available to solve various optimization problems. One category of problems, known as combinatorial optimization problems, regroups most logistic problems that have to be tackled in our society (supply chains, parcel delivery, time-tabling conception, task planning, and many more).

More precisely, a combinatorial optimization problem is characterized by discrete decision variables and a finite search space ($\mathcal{D}$), containing all feasible solutions. To solve a combinatorial optimization problem, we look for the *best* solution(s) according to one (or more) criteria. The criteria are often modeled through an objective function (noted f), that assigns a score to a solution, reflecting how the solution found respects the specified criteria. Hence, the goal is to determine (at least) one solution $x^* \in \mathcal{D}$ that is optimum (either maximum or minimum, depending on the problem) for f, i.e., such that $\forall x \in \mathcal{D}, f(x^*) \leq f(x)$ (in a minimization context).

All combinatorial optimization problems are not equally difficult. There exists a field in computer science, known as complexity theory [Garey and Johnson, 1979], that classifies the different algorithmic problems according to their complexity. Two classes are widely known: P and NP. The P (Polynomial) class contains all problems that can be solved in polynomial time on a deterministic machine (i.e., without using randomness). The NP (Non-deterministic Polynomial) class groups together decision problems that can be solved in polynomial time on a non-deterministic machine. In the NP class, we find NP-hard problems, which represent the most difficult problems of the class. A problem is said NP-hard when every other problem of the NP class can be (polynomially) reduced to it. There exist problems in NP that are not NP-hard. Without going into the details, many combinatorial optimization problems are known to be NP-hard: the traveling salesman problem (TSP), the knapsack problem (KP), the permutation flowshop scheduling problem, and in particular, all routing problems.

Attempting to find an optimum solution to a combinatorial optimization problem by performing an exhaustive search is an impractical choice [Korte et al., 2011]. Although the field of exact algorithms is not able to solve large instances, it remains an active and important field in the optimization community. An efficient strategy used in exact algorithms is the branch-and-cut [Padberg and Rinaldi, 1991], which combines the branch-and-bound strategy [Lawler and Wood, 1966] and the cutting plane method [Kelley, 1960]. Moreover, for vehicle routing problems, the state of the art are branch-price-and-cut methods [Desrosiers and Lübbecke, 2011]. To tackle large instances, other tools have been developed, like metaheuristics. A heuristic is an algorithmic strategy used to generate a reasonably good solution in a short amount of time. In practice, heuristics are often a good choice to solve a specific problem, however, they are not designed to tackle various optimization problems. For example, an efficient heuristic for the TSP can not be used to solve the KP, since the two problems are structurally too different (the TSP uses solutions represented as permutations, while the KP uses binary solutions). A metaheuristic is generally based on a strategic approach, where the components problem-specific are abstracted. Thus, a metaheuristic can be applied to different problems but may require the implementation of problem-specific components or heuristics. Many kinds of metaheuristics, using various strategies, have been developed, see for example the recent survey of Hussain et al. [2019] for more details.

With the use of metaheuristics, it is possible to approximate the optimum solution of the optimization problem considered in a reasonable amount of time. However, due to the no-free-lunch theorem (stating that a general-purpose, universal optimization strategy is impossible), a single metaheuristic can not be the best on all optimization problems. In the literature, we find two main families of metaheuristics, neighborhood-based metaheuristics presented in Section 1.2.1, and nature-inspired metaheuristics, like evolutionary algorithms, presented in Section 1.2.2.

1.2.1 Neighborhood-based Metaheuristics

This section presents the concepts related to neighborhood-based metaheuristics. Behind the concept of neighborhood (formally presented in Section 1.2.1.1), hides the notion of local search. The basic idea underlying local search (LS) is that high-quality solutions to an optimization problem can be found by iteratively improving a solution using small (local) modifications, called *moves*.

To perform a local search, three key ingredients are required: an initial solution (which can be generated randomly if the problem is not too constraint, or with a greedy algorithm), a neighborhood to explore, and an exploration strategy (indicating when the exploration of the neighborhood stops).

The concept of neighborhood and some examples are introduced in Section 1.2.1.1. Section 1.2.1.2 presents commonly used exploration strategies. Finally, Section 1.2.1.3 features some neighborhood-based metaheuristics.

1.2.1.1 Concept of Neighborhood

The neighborhood of a solution is defined with a *neighborhood operator* (or *local search operator*), *op*. This operator generates a set of possible moves that can be applied to a solution x. The set of solutions that can be obtained by applying one move defined by *op* to a solution x is called the *neighborhood* of x, noted $\mathcal{N}^{op}(x)$. In Figure 1.1, a simple solution space is represented. The current solution, x, is colored red. The neighborhood of a solution, represented in blue, contains all the closest feasible solutions that are in the same row or column (in the figure).

In the neighboring solutions, we distinguish between two types of solutions: the improving solutions, which are closer (in terms of the objective function) than x to the optimum solution x^*, in green, and the non-improving solutions, which are the other ones. It is possible that the neighborhood of a solution (for a specific operator) does not contain any improving solution, without being the best optimum solution. In that case, the solution is called a *local optimum*. Formally, a solution x is a local optimum for the operator *op* if $\forall x' \in \mathcal{N}^{op}(x), f(x) \leq f(x')$ (in a minimization context). This situation is represented in Figure 1.2, where the neighborhood is the same as defined in Figure 1.1. Non-improving solutions are crossed in the figure.

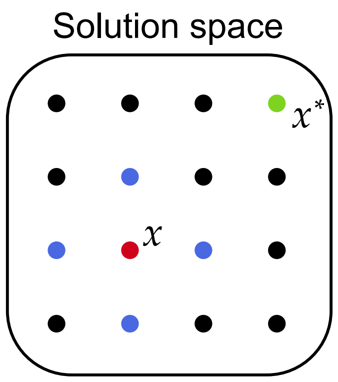


Figure 1.1: Neighborhood (blue dots) of a solution x (red dot). The best solution is x^* (green dot). In this situation, some neighboring solutions are closer to the optimum solution.

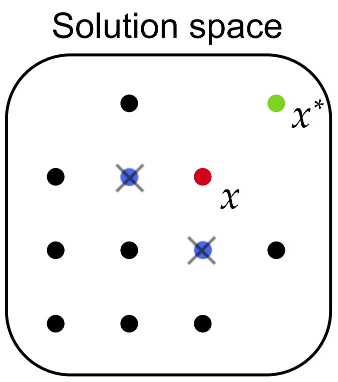


Figure 1.2: The current solution x (red dot) is a local optimum since no neighboring solutions (blue dots) are closer to the optimum solution x^* (green dot).

An important observation is that a local optimum for one local search operator is generally not a local optimum for another one. For this reason, it is interesting to use several local search operators in a single algorithm. However, the usage of several local search operators is only beneficial as long as they explore different neighborhoods. In the best case, the pairwise intersections of the considered neighborhoods are empty, i.e, given two operators op_1 and op_2 we have $\mathcal{N}^{op_1}(x) \cap \mathcal{N}^{op_2}(x) = \varnothing$ for every solution x.

1.2.1.2 Exploration Strategies

In this section, more details are given about the exploration of the neighborhood of a solution $\mathcal{N}^{op}(x)$. The exploration strategy decides which neighbor to accept.

Two classical neighborhood exploration strategies commonly adopted in the literature are the *best* and the *first* improvement strategies [Hoos and Stützle, 2004]. The *best* strategy (represented in Figure 1.3) entirely explores $\mathcal{N}^{op}(x)$ to find the neighbor maximizing the improvement. This strategy guarantees the best possible progress at each iteration of the local search but the time allotted to the exploration may become an issue when a fast exploration is wanted. The *first* strategy (represented in Figure 1.4) evaluates the neighboring solutions one by one and stops as soon as it finds one improving (or non-deteriorating, i.e., with the same fitness) neighbor. Accepting non-deteriorating solutions is beneficial when many solutions share the same fitness, to explore different regions of the search space. This strategy allows a fast exploration but many improving neighbors are set aside. Moreover, the convergence rate of the *first* strategy is slower than the one of the *best* strategy, when the exploration is performed several times.

Finally, the two strategies can be mixed by generating first a subset of the neighborhood and taking the best-improving neighbor from this subset. This strategy was initially proposed by Ruiz and Stützle [2007] to create an iterative greedy heuristic for the permutation flowshop scheduling problem. More generally, Tari et al. [2022] studied different *Partial Neighborhood Local Search*

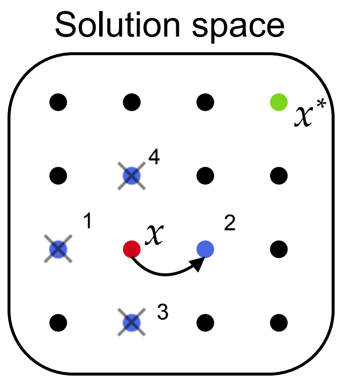


Figure 1.3: Illustration of the *best* exploration strategy. All the neighboring solutions are explored. Discarded solutions are crossed. Finally, one solution maximizing the improvement is selected.

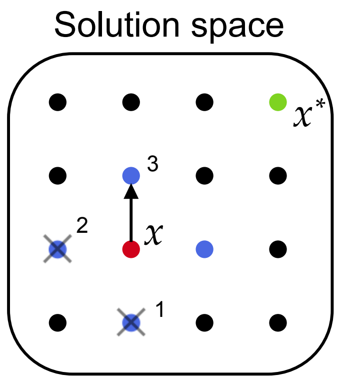


Figure 1.4: Illustration of the *first* exploration strategy. The neighboring solutions are explored in a random order. Non-improving solutions are crossed. The first improving solution is selected.

techniques, which consist of adaptive walks where the moves are chosen in a random subset of the current solution neighborhood. In particular, the balance between intensification and diversification is controlled by a single parameter impacting the size of the subset.

1.2.1.3 Classical Neighborhood-based Metaheuristics

Many metaheuristics based on neighborhood have emerged over the years, however most of them are based on strategies developed decades ago. Among the most popular strategies we find the Hill Climbing, the Simulated Annealing [Kirkpatrick et al., 1983], the Tabu Search [Glover and Laguna, 1998], the Iterated Local Search [Lourenço et al., 2003], and the Variable Neighborhood Search [Mladenović and Hansen, 1997]. All these strategies use different trade-offs between intensification (focus on a specific region to reach a local optimum) and diversification (largely exploring the search space).

The Hill Climbing (HC) is presented in Algorithm 1. The algorithm starts with an initial solution x and a neighborhood operator op, entirely explores the neighborhood of x, replaces it with the best improving solution (*best* strategy), and repeats the process until a local optimum is found. Given any neighborhood operator, the HC is the fastest (in number of iterations) strategy to reach a local optimum. This algorithm does not use any diversification strategy. Moreover, once a local optima is reached, to pursue the search, strategies to escape local optima are required. Such strategies may accept non-improving solutions to explore other regions of the solution space.

Contrarily to the HC, the Simulated Annealing (SA) integrates a diversification strategy since it may accept neighboring solutions that do not improve the current solution with a probability. The interest in accepting non-improving solutions is to avoid getting stuck in local optima. It allows exploring other regions of the solution space, that would have been discarded with a HC. The original SA employs the Metropolis function, which considers the fitness difference and a *temperature* to decide the replacement of the current solution by a non-improving solution. The temperature gradually decreases every iteration, according to a cooling rate, and the probability

Algorithm 1: Hill Climbing.

```
Input: Initial solution $x$, neighborhood operator $op$
Output: A local optimum $bestSolution$
1 $improved \leftarrow True$
2 $bestSolution \leftarrow x$
3 while $improved$ do
4     $improved \leftarrow False$
5     for $x' \in \mathcal{N}^{op}(bestSolution)$ do
6         if $f(x') < f(bestSolution)$ then
7             $bestSolution \leftarrow x'$
8             $improved \leftarrow True$
9 return $bestSolution$
```

of accepting a non-improving solution decreases with the temperature. The neighborhood of the solution is explored until a new solution is accepted. The algorithm stops when a local optimum is reached or when the temperature becomes 0.

Once a local optimum is reached (for all the operators considered), it is no longer possible to improve the solution by applying a local search. To continue the exploration of the search space, an idea is to realize one (or more) deteriorating moves (diversification) before starting a new local search (intensification). More generally, this step is called a *perturbation* step. This strategy is called Iterated Local Search (ILS). Algorithm 2 illustrates the steps of the ILS. In particular, the `LS` step can be a Hill Climbing, and the `Perturbation` can apply random moves.

Algorithm 2: Iterated Local Search.

```
Input: Initial solution $x$
Output: A local optimum $bestSolution$
1 $bestSolution \leftarrow x$
2 $currentSolution \leftarrow x$
3 while $not\ termination\ criterion$ do
4     $currentSolution \leftarrow$ LS($currentSolution$)
5     if $f(currentSolution) < f(bestSolution)$ then
6         $bestSolution \leftarrow currentSolution$
7     $currentSolution \leftarrow$ Perturbation($currentSolution$)
8 return $bestSolution$
```

When multiple neighborhood operators are available, an interesting strategy is the Variable Neighborhood Search (VNS), which returns a local optimum for all the neighborhood operators. The idea is to change the neighborhood whenever a local optimum is reached and restart the exploration of all the neighborhoods when an improving solution is found. The main steps of the strategy are described in Algorithm 3. Some existing variants are presented by Hansen et al. [2019]. In general, they differ from the order of exploration of the neighborhoods, the `Exploration` step, or the backtrack choice (i.e., the next neighborhood selected when an improving solution is found).

Algorithm 3: Variable Neighborhood Search.

```
Input: Initial solution $x$, $k$ operators $op_1, \dots, op_k$
Output: A local optimum $bestSolution$
1  $bestSolution \leftarrow x$
2  $currentSolution \leftarrow x$
3  $currentNeighborhood \leftarrow 1$
4  while $currentNeighborhood \leq k$ do
5      $currentSolution \leftarrow$ Exploration$(currentSolution, op_{currentNeighborhood})$
6      if $f(currentSolution) < f(bestSolution)$ then
7          $bestSolution \leftarrow currentSolution$
8          $currentNeighborhood \leftarrow 1$
9      else
10         $currentNeighborhood \leftarrow currentNeighborhood + 1$
11 return $bestSolution$
```

When non-improving solutions are accepted during the search, there is a possibility to cycle between the same solutions, preventing the algorithm to explore new regions of the solution space and to find new better solutions. To avoid cycling between the same regions of the search space, it is possible to associate a *tabu* list with specific solutions or components. This tabu list is a memory of what has already been explored. All the solutions that do not belong to the tabu list or do not have components in the tabu list can still be selected. This strategy is called a tabu Search. In practice, it favors the exploration of new regions of the search space. Moreover, it is possible to remove elements of the tabu list after some iterations.

1.2.2 Nature-inspired Metaheuristics

Nature-inspired metaheuristics emulate behaviors observed in natural systems to solve complex problems. Many nature-inspired metaheuristics have emerged in the last decades [Yang, 2020], but the most popular ones remain the evolutionary algorithms, and more particularly the Genetic Algorithms (GA) introduced by Holand [1975], the Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995], and the Ant Colony Optimization (ACO) [Dorigo and Di Caro, 1999].

Briefly, PSO solves a problem by moving solutions, named particles, in the search space according to a mathematical formula over the particle's position and velocity. The movement of each particle is influenced by its local best-known position but is also guided toward the best-known position(s) in the search space. ACO is a multi-agent (each agent is an ant) method inspired by the behavior of real ants, which use pheromones to indicate the best paths to follow to reach their objective (e.g., food). Algorithmically each ant, represented by a solution, locally explores its neighborhood, with a bias induced by pheromones of other ants when the path is better (more chance to go there) or worse (less chance) than the others.

Evolutionary algorithms use principles of natural selection, recombination, and mutation to evolve a population of potential solutions. A Genetic Algorithm (GA) is a specific evolutionary algorithm using genetic operators to evolve the population of candidate solutions [Goldberg, 1989]. Algorithm 4 details the framework of a GA. An initial population is generated using an operator `Initialize`. Solutions can be randomly generated or constructed via a heuristic when it is available.

Then, solutions are evaluated (`Evaluate`) to determine their fitness. Most of the time the fitness is obtained with the objective function. If it is costly to evaluate a solution, a surrogate should be used to approximate the objective function, so that the evaluation step remains as fast as possible. Members of the population evolved for a maximum number of iterations (I_{max}), called *generations* (however, we keep the term iteration, even when a genetic algorithm is considered). During each generation, a few solutions are selected (`Select`) to inherit their characteristics to new solutions. Among the most used selections, we find the elitist selection, where the best individuals are selected, the roulette wheel selection, where the probability of being selected depends on the contribution of the fitness of the solution to the sum of all fitnesses, and the tournament selection, where solutions compete for two by two and the solution with the best fitness makes it to the next round. A set of new solutions, called *offspring*, is created by applying a crossover mechanism (`Crossover`) to the selected individuals. The crossover step tries to generate a new solution by mixing the best components of the parents. In addition, a mutation step (`Mutation`) can occur to bring more diversity to the solutions. However, the mutation step can also be used to exploit the solutions found, i.e., intensify the search around the solutions to improve them. Note that, both crossover and mutation steps depend on the representation of the solutions. We provide an example of crossover and mutation in the case of solutions represented as permutations. In Figure 1.5, the Partially Mapped Crossover (PMX) is illustrated. First, two points are selected to divide each permutation into three fragments. The middle fragments are exchanged, but it may create redundant elements. To repair the permutation, we use a mapping induced by the exchange of the middle fragments. For example, the value 9 became redundant in Child 1 after the exchange, thus 9 is mapped to 3, but 3 is already in the middle fragment, leading to a second mapping 3 to 5. The last mapping sends 7 to 4, allowing the correction of Child 1. The process is similar for Child 2. In Figure 1.6, the permutation swap mutation is presented, where two elements of the permutation are simply swapped. Finally, some members of the population are replaced by the new generation of solutions (`Update`). Update strategies are the same as selection strategies since the update is equivalent to selecting M solutions in the set containing the current population and the offspring. At the end of the execution, the best individual of the population is returned.

In the field of evolutionary computation, the term *exploration* (resp. *exploitation*) designs diversification (resp. intensification) mechanisms. In that context, exploration means finding new members for our current population, while exploitation means optimizing the current population to reach the best individuals. As for neighborhood-based metaheuristics, where a good trade-off between intensification and diversification is required to reach high-quality solutions, a good trade-off between exploration and exploitation is necessary to obtain an efficient GA. From now on, when we mention exploration of solution, in the context of neighborhood exploration (like in local search), it is equivalent to an intensification mechanism (and thus to exploitation). In general, neighborhood exploration (which is an intensification step) should not be confused with exploration strategy (which refers to diversification mechanisms). In particular, in the second part of the thesis, we use the term exploration for neighborhood exploration, even if it is considered as an exploitation mechanism.

1.3 Multi-objective Optimization

Single-objective optimization is sometimes not enough to model more complex problems encountered in real life. Indeed, minimizing a single objective function may bias the search towards solutions that are finally not such interesting in practice. By considering more objectives, it is

Algorithm 4: Genetic Algorithm (GA).

```
Input: Size of the population $M$, maximum number of iterations $I_{max}$
Output: The best solution found
1  $Population \leftarrow$ Initialize($M$)
2  Evaluate($Population$)
3  $I \leftarrow 0$
4  while $I < I_{max}$ do
5      $Parents \leftarrow$ Select($Population$)
6      $Offspring \leftarrow$ Crossover($Population$)
7      Mutation($Offspring$)
8      Evaluate($Offspring$)
9      Update($Population, Offspring$)
10     $I \leftarrow I + 1$
11 $bestSolution \leftarrow$ BestOf($Population$)
12 return $bestSolution$
```

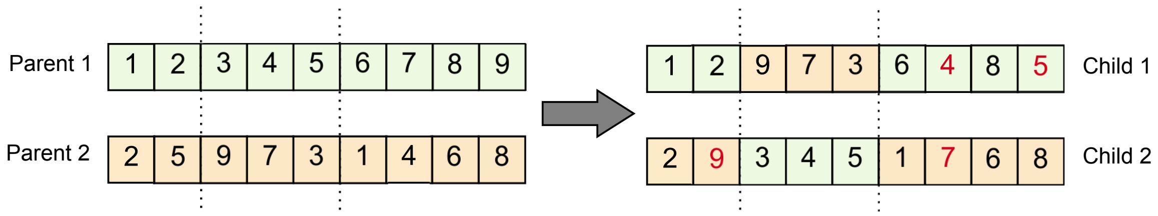


Figure 1.5: Partially mapped crossover (PMX).

possible to handle more aspects of the optimization problem and provide more possibilities to a decision-maker. To turn a single-objective problem into a multi-objective problem, it is possible to relax one or several constraints by turning them into objective functions to optimize (e.g., in routing problems, the number of vehicles usually appears as a constraint of the problem to limit the maximum number of vehicles used, but this constraint can be relaxed by putting the number of vehicles as an objective function to minimize). This has the advantage of considering a less constrained problem, whose solution space is easier to explore. Another possibility to transform a single-objective problem into a multi-objective one is to define new objective functions considering aspects of the original problem that were put aside (e.g., in routing problems, the fairness between drivers, customer satisfaction, and the environmental aspect). Since it is not possible to present all aspects of the multi-objective optimization field, we refer to the book of Deb et al. [2016] for more detailed information.

In the following, we mainly focus on multi-objective combinatorial optimization, whose concepts are presented in Section 1.3.1. We present in Section 1.3.2 various quality indicators that are commonly used to assess the quality of the solutions returned by multi-objective algorithms. Section 1.3.3 is dedicated to approaches used to turn a multi-objective problem into a single-objective problem. Finally, Section 1.3.4 features resolution methods adapted to multi-objective optimization.

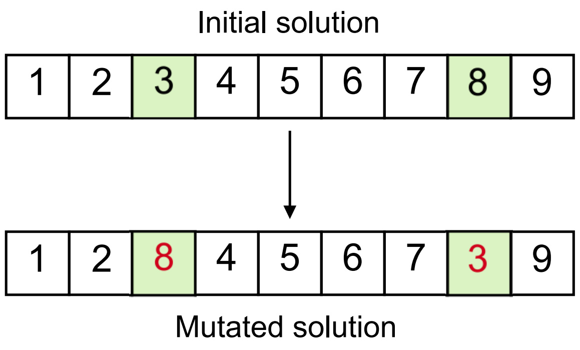


Figure 1.6: Permutation swap mutation.

1.3.1 Definitions

A *Multi-objective Combinatorial Optimization Problem* (MoCOP) is commonly formalized as follows [Coello et al., 2010]:

$$(MoCOP) = \begin{cases} Optimize & F(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ s.t. & x \in \mathcal{D}, \end{cases} \tag{1.1}$$

where $n \geq 2$ objective functions f_i have to be optimized, x is a vector of decision variables, and $\mathcal{D}$ is the (discrete) set of feasible solutions. The *objective space*, noted $\mathcal{Z}$, is the image of F. In particular, each solution $x \in \mathcal{D}$ is associated with a point $F(x) \in \mathcal{Z}$. When four (or more) objectives are considered, we talk about *many-objective optimization*. With such problems comes another issue: the visualization of the Pareto front obtained.

We say that a solution x *dominates* a solution y, noted $x \succ y$ in a minimization context, if and only if for all $i \in \{1 \ldots n\}$, $f_i(x) \leq f_i(y)$ and there exists $j \in \{1 \ldots n\}$ such that $f_j(x) < f_j(y)$. We say that x *weakly dominates* y, noted $x \succeq y$, if and only if for all $i \in \{1 \ldots n\}$, $f_i(x) \leq f_i(y)$. The ϵ-dominance [Laumanns et al., 2002] naturally extends the dominance relation: for any real $\epsilon > 1$, x ϵ-dominates y, noted $x \succeq_\epsilon y$, if and only if for all $i \in \{1 \ldots n\}$, $f_i(x) \leq \epsilon \cdot f_i(y)$, in a minimization context and assuming that all points are positive in all objectives. The dominance relation induces a partial order in the solution space. Indeed, there exist pairs of solutions that cannot be compared to each other. Such solutions are said *incomparable*. Intuitively, two solutions are incomparable when there exists a set of objectives for which one solution is better and another set for which the other solution is better. Figure 1.7 illustrates the different categories of neighbors of a solution in a bi-objective context. The dominated solutions are worse than the current solution, the non-dominated ones are incomparable, and the dominating ones are better. For example, let be three solutions x_1, x_2, x_3, with their corresponding bi-objective vectors $F(x_1) = (2, 5)$, $F(x_2) = (4, 2)$, $F(x_3) = (5, 6)$. x_1 and x_2 dominates x_3, x_1 and x_2 are incomparable, x_2 ϵ-dominates x_1 with $\epsilon \geq 2$.

A *Pareto front* (or *approximation set*) is defined as a set of non-dominated solutions. The set of all approximation sets is denoted Ω. The Pareto dominance relations defined above can be extended to relations between objective vector sets [Zitzler et al., 2003]. A set S_1 dominates (resp. weakly dominates) another set S_2, noted $S_1 \succ S_2$ (resp. $S_1 \succeq S_2$) when: $\forall x_2 \in S_2, \exists x_1 \in S_1, x_1 \succ x_2$ (resp. $\forall x_2 \in S_2, \exists x_1 \in S_1, x_1 \succeq x_2$).

A feasible solution $x^* \in \mathcal{D}$ is called *Pareto optimal* if and only if there is no solution $x \in \mathcal{D}$ such that $x \succ x^*$. Extending the notion of local optimum introduced in Section 1.2.1.1 for single-objective optimization, in a multi-objective context, a *Pareto local optimum* is a solution whose neighborhood (for a specific operator) does not contain any dominating solution. To solve a MoCOP it is required to find all the Pareto optimal solutions, which form together the *Pareto optimal set*. The image of the Pareto optimal set by the objective function F provides the *optimal Pareto front*. The *ideal* (resp. *nadir*) point associated with an optimal Pareto front is the point obtained by taking the best (resp. worse) values, considering all the solutions of the front, for each objective. Figure 1.8 represents a small bi-objective space. The blue dots represent the optimal Pareto front. The ideal and nadir points are represented too. Note that, they only depend on the optimal Pareto front, and not on all the solutions of the objective space. In the remainder of the thesis, if nothing is mentioned, the objectives are minimized by default.

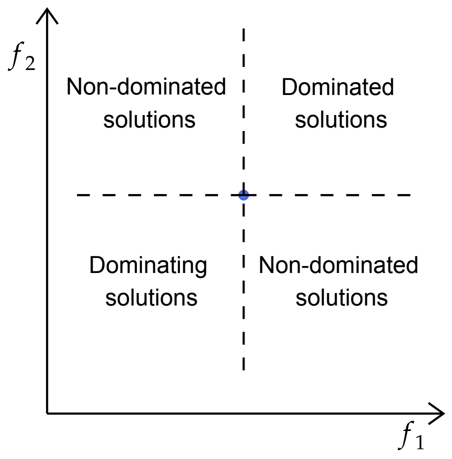


Figure 1.7: The different types of neighbors of a solution in a bi-objective context, when minimizing both f_1 and f_2.

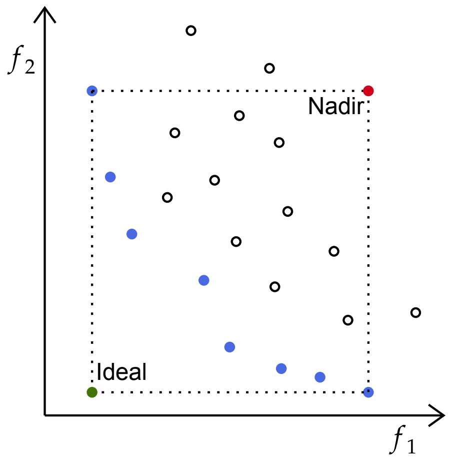


Figure 1.8: A bi-objective space in minimization context. Each dot is associated with a solution, except the nadir and ideal points which are virtual. The blue dots represent the optimal Pareto front.

1.3.2 Quality Indicators

Comparing the performance between different multi-objective algorithms requires metrics (or indicators) to analyze the quality of the fronts returned. A unary indicator is a unary function $I : \Omega \mapsto \mathbb{R}$, attributing a real value to an approximation set (Ω denotes the set of all approximation sets). Binary indicators are binary functions that directly compare two approximation sets (provided in input), by returning a real value symbolizing the difference between the two sets. Many metrics are reviewed and ranked by Riquelme et al. [2015]. The quality of a Pareto front is based on three characteristics: its accuracy (measuring the closeness to the optimal Pareto front), its diversity (measuring how solutions are spread along the front), and its cardinality (measuring if enough solutions have been found to represent the front). In the following, the cardinality of a front S_1 is noted $|S_1|$. Given a unary indicator I, an approximation set S_1 is preferable to another approximation set S_2, when $I(S_1) > I(S_2)$, in a case where I has to be maximized.

Section 1.3.2.1 presents the monotonicity property of indicators. Then, each following section describes an indicator or a family of indicators: Section 1.3.2.2 focuses on the hypervolume, Section 1.3.2.3 on the generational distance, and Section 1.3.2.4 on the epsilon indicator.

1.3.2.1 Monotonicity

Depending on the situation and the comparison wanted, some properties about unary indicators can be considered. Among the most important properties, we find the *monotonicity* of the indicator, and its associated *computation effort*. An indicator I is said to be *monotonic* if and only if for any approximation set S_1 that is compared to another approximation set S_2, if S_1 is at least as good as S_2 in terms of the weak dominance relation ($S_1 \succeq S_2$), then the quality of S_1 is at least as good as the quality of S_2, measured by I (i.e., $I(S_1) \geq I(S_2)$, when the indicator is to be maximized). This property is formally expressed as follows:

$$\forall S_1, S_2 \in \Omega : S_1 \succeq S_2 \implies I(S_1) \geq I(S_2). \tag{1.2}$$

The monotonicity guarantees that an indicator does not contradict the partial order of the set of the approximation sets, imposed by the weak dominance relation. However, it is not enough to guarantee a unique optimum with respect to the indicator values. In other words, an approximation set with the same indicator value as the optimal Pareto front does not necessarily contain only Pareto optimal solutions. To remedy this, the property of *strict monotonicity* is required, which is based on Pareto dominance (instead of weak Pareto dominance), turning Property (1.2) in:

$$\forall S_1, S_2 \in \Omega : S_1 \succ S_2 \implies I(S_1) > I(S_2). \tag{1.3}$$

1.3.2.2 Hypervolume

The unary hypervolume (I_{uHV}, or simply abbreviated uHV) [Zitzler et al., 2003], is the most used metric in the literature. It is defined relatively to a reference point Z_{ref}, generally $(1.001, \ldots, 1.001)$, and requires that the objectives of the solutions are normalized between 0 and 1. A reference point slightly further than $(1, \ldots, 1)$ is often preferred to avoid computational issues. This indicator is to be maximized and allows a good evaluation of the front's accuracy, diversity, and cardinality. Geometrically, see Figure 1.9, I_{uHV} represents the volume of the objective space (bounded by Z_{ref}) covered by the members of a non-dominated set of solutions. The larger the hypervolume, the better the set of solutions. It is possible to turn I_{uHV} into a binary indicator I_{HV}, by taking two approximation sets S_1 and S_2 (with S_2 a reference set, that may replace the optimal Pareto front), and computing $I_{HV}(S_1, S_2) = I_{uHV}(S_1) - I_{uHV}(S_2)$. The unary hypervolume is known to be a strictly monotonic metric, and currently this is the only metric verifying this property (i.e., currently, no other metric is known to be strictly monotonic). However, this metric is hard to compute in high dimensions, and the computation cost grows exponentially with the number of objectives. To be more precise, it requires about $O(s_f \ log \ s_f + s_f^{n/2})$ comparisons [Beume et al., 2009] with n being the number of objectives and s_f the number of points in the front.

1.3.2.3 Generational Distance

The second most-used indicator is the Generational Distance (I_{GD}). It calculates how far an approximation set (S_1) is from the optimal Pareto front (or any reference set R). The value $I_{GD}(S_1, R)$

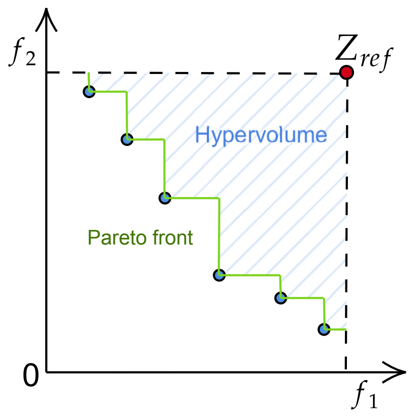


Figure 1.9: Hypervolume of a Pareto front in a bi-objective minimization context. The reference point Z_{ref} bounds the area.

is formally obtained with Equation (1.4). In practice, the value $p = 1$ is considered, making the interpretability easier: it considers the average Euclidean distance between the members of S_1 (noted as x_i) and their nearest member in the reference set R, noted $d_E(x_i, R)$.

$$I_{GD}(S_1, R) = \frac{1}{|S_1|}\left(\sum_{i=1}^{|S_1|} d_E(x_i, R)^p\right)^{1/p}. \tag{1.4}$$

Note that this indicator only considers one aspect of the front: its accuracy. Moreover, this indicator is fast to compute, since it requires $O(|S_1| \cdot |R|)$ comparisons.

The Inverted Generational Distance (I_{IGD}) is a commonly used indicator too. It exchanges the roles of the approximation set and the Pareto front in the calculation of I_{GD}. In addition to the accuracy, I_{IGD} is able to measure the diversity of the approximation set. The indicator I_{IGD} is easily computed by using the relation $I_{IGD}(S_1, R) = I_{GD}(R, S_1)$.

The metrics I_{GD} and I_{IGD} can be seen as an error to be minimized. However, when a solution x and its closest reference point z are non-dominated, moving x to z does not improve the two objectives of x (in a bi-objective context, one is improved in the detriment of the other one). Thus, decreasing the distance between the two solutions does not always improve the objectives of x. To remedy this issue Ishibuchi et al. [2015] proposed to consider a metric d^+, given by Equation (1.5) (in a minimization context), to evaluate the distance between solutions. In a maximization context the quantity $\max\{f_i(z) - f_i(x), 0\}$ is used instead of $\max\{f_i(x) - f_i(z), 0\}$.

$$d^+(x, z) = \sqrt{\sum_{i=1}^{n} (\max\{f_i(x) - f_i(z), 0\})^2}. \tag{1.5}$$

Replacing the Euclidean distance in the (inverted) generational distance leads to metrics I_{GD+} and I_{IGD+}. The I_{IGD+} has the advantage of being monotonic, contrary to the metrics I_{GD}, I_{IGD}, and I_{GD+}.

1.3.2.4 Epsilon Indicator

Finally, the third most-used indicator is the epsilon-indicator, noted I_ϵ. It is a binary indicator, whose value $I_\epsilon(S_1, S_2)$ gives the minimum factor ϵ by which objective vector associated with S_2 can be multiplied such that the resulting transformed approximation set is weakly dominated by the approximation set S_1. Formally, let S_1 and S_2 two approximation sets (S_2 is usually considered as a reference set), then:

$$I_\epsilon(S_1, S_2) = \inf_{\epsilon \in \mathbb{R}} \{\forall y \in S_2, \exists x \in S_1, x \succeq_\epsilon y\} = \max_{y \in S_2} \min_{x \in S_1} \max_{1 \leq i \leq n} \epsilon(f_i(x), f_i(y))$$

where $\epsilon(f_i(x), f_i(y))$ is equal to $f_i(x)/f_i(y)$ in a minimization context, and $f_i(y)/f_i(x)$ in a maximization context. In both cases, a lower value corresponds to a better approximation set. The unary epsilon indicator (obtained by fixing the reference set for the problem), is monotonic, but not strictly monotonic. It is cheap to compute since the runtime complexity is of order $O(n \cdot |S_1| \cdot |S_2|)$, with S_1 the approximation set and S_2 the reference set.

In the same manner, it is possible to define an *additive* ϵ-indicator, noted $I_{\epsilon+}$, with Equation (1.6). This indicator is monotonic too.

$$I_{\epsilon+}(S_1, S_2) = \inf_{\epsilon \in \mathbb{R}} \{\forall y \in S_2, \exists x \in S_1, x \succeq_{\epsilon+} y\} \tag{1.6}$$

where $x \succeq_{\epsilon+} y$ if and only if $\forall\, 1 \leq i \leq n,\ f_i(x) \leq \epsilon + f_i(y)$.

1.3.3 Modelling the Multi-objective Problem by a Single-objective One

Three main strategies exist to model multi-objective problems [Jozefowiez et al., 2008]: *a priori*, *a posteriori*, and interactive strategies. The *a priori* approach is discussed in this section, while the two other ones are discussed in Section 1.3.4.

The idea behind an *a priori* approach is to model the multi-objective problem as a single-objective problem by using information provided by the decision-maker. It constrains the objective space, allowing a more focused exploration. For example, the decision-maker may want to ponder the different objectives to show that one objective is more important than the others, without totally discarding the other objectives. Using weighted aggregations of objectives is a common way to model the multi-objective problem by an *a priori* approach [Jin et al., 2001]. More generally, the idea is to use mathematical transformations, *scalarizations*, to turn the multi-objective problem into a single-objective problem, that is easier to solve. In particular, such transformations divide the objective space into small regions, to approximate the Pareto front more efficiently. Although scalarizations are not so accurate and exhaustive, they are simple to implement.

There exist many ways to generate scalar problems, but in every case, it requires a weight vector $w = (w_1, \ldots, w_n)$ such that, $\forall i \in \{1, \ldots, n\}$ $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$, where n is the number of objectives considered. The two most commonly used scalar approaches are the *weighted sum* (WS) and the *Tchebycheff* approach (TCH).

To define a scalar problem with a WS only a weight vector is required. The fitness associated with a problem with n objective functions $f_1, \ldots, f_n$ is computed with Equation (1.7).

$$g_{fit}^{WS}(x, w) = \sum_{i=1}^{n} w_i \cdot f_i(x). \tag{1.7}$$

Concerning the Tchebycheff approach, it requires a reference point z^0 in the objective space, in addition to the weight vector to define a scalar problem. Then the fitness associated with the problem is obtained with Equation (1.8).

$$g_{fit}^{TCH}(x, w, z^0) = \max_{1 \leq i \leq n} \{w_i \cdot (f_i(x) - f_i(z^0))\}. \tag{1.8}$$

Note that, the objective functions f_i should be normalized if their range highly differ. Indeed, when an objective function takes small values and another one very high values, without normalization, it will not be possible to obtain evenly spread trade-offs with weights in $[0, 1]$.

Another common scalar approach uses goal programming methods, where a goal (i.e. an objective vector) is chosen, and then a search is conducted to minimize the distance between the current solution and the goal. Defining an interesting goal is the main difficulty of this method, but it can be provided by the decision-maker, to represent its ideal solution.

Once the problem is reduced to a single-objective problem, any method described in Section 1.2 can be used to solve the problem. *A priori* approaches are relevant only when the decision-maker knows exactly what they are looking for in terms of objectives. However, it is known that in multi-objective optimization very different solutions can produce close objective vectors. For that reason, we would recommend using population-based metaheuristics (like evolutionary algorithms presented in Section 1.2.2) to maintain diversity among high-quality solutions. If several final solutions seem interesting, the final choice can be left to the decision-maker.

1.3.4 Strategies to Approximate the Optimal Pareto Front

The *a posteriori* strategy is the opposite of the *a priori* strategy, in the sense that the problem is solved without further information from the decision-maker. The decision-maker operates at the end of the process to choose its preferred solution among the solutions returned [Branke et al., 2004]. An overview of the most commonly used algorithms to solve a multi-objective problem is provided in the remainder of this section. We first introduce the concept of diversity and bounded archives in Section 1.3.4.1. Indeed, for problems with a huge objective space and where the optimal Pareto front is large, the notions of bounded archives and diversity of solutions have to be envisaged. More precisely, in practice, a decision-maker can have difficulties making his/her choice if hundreds of solutions are provided, hence the size of the Pareto set returned is often bounded. Furthermore, the approximation set must fairly represent the optimal Pareto front, which requires diverse solutions. Then, some nature-inspired multi-objective algorithms are presented in Section 1.3.4.2, where we highlight the main differences with the algorithms presented in Section 1.2.2. The concept of multi-objective local search is introduced in Section 1.3.4.3, and a few hybrid algorithms are detailed in Section 1.3.4.4. Some of these algorithms use the notion of Pareto dominance to evaluate the quality of a solution or to compare solutions, while others use decomposition strategies described in the former section.

Finally, interactive approaches integrate the decision-maker in the execution process. Interactive strategies mix *a priori* (to integrate the knowledge of the decision-maker, allowing the exploration of a restrained objective space) and *a posteriori* (to obtain more diverse solutions or to provide an insight of the possible solutions to the decision-maker) strategies, by giving the decision-maker an important place in the discussions. In fact, solving a multi-objective optimization problem with an interactive approach is a constructive process where the decision-maker discovers what kind of solutions are available and confronts this knowledge with their preferences, which also evolve during the process. Such methods are not further developed in the remainder of the thesis, but we refer to the book of Deb et al. [2016] for the interested reader.

1.3.4.1 Notions of Diversity and Bounded Archives

Since the goal of a multi-objective problem is generally not to find a single solution but a set of (non-dominated) solutions, the question of the diversity of the returned solutions arises. We distinguish between two kinds of diversity: the *genotypic* diversity, which is related to the internal structure of the solutions (e.g., the routes for a vehicle routing problem), and the *phenotypic* diversity, which refers to the observable characteristics of a solution (in general, its associated objective vector). In other words, the goal of the multi-objective problem is to optimize the phenotype, which is obtained by evaluating the genotype, with the corresponding objective functions. Generally, the constraints of a problem concern the genotype (e.g., constraints over the length of a route, or the presence of a specific pattern in the solution). It is also possible to consider constraints over the phenotype when specific objective values have to be discarded. It is important for the decision-maker to have access to a set of non-dominated solutions, that fairly represent the diversity of the solutions to the problem in both *phenotype* and *genotype*.

When solving MoCOPs, it is common to limit the size of the *archive* to store the current Pareto set. In the remainder of the thesis, the term *archive* refers to the structure storing a set of non-dominated solutions (i.e., a Pareto front). In some multi-objective problems, the size of the optimal Pareto front can grow exponentially with the number of objectives considered and with the size of the instance. This is the case for instances of the bi-objective TSP (bTSP). A bTSP instance is defined by using two distinct distance matrices, so that, allowing the evaluation of a solution in two different manners, leading to two objective functions to be optimized. Since it is not interesting for a decision-maker to deal with such large fronts, different techniques have emerged to limit the size of an archive during the execution. The bounded archive has to respect some properties of the original front, such as its convergence, its diversity, and its extreme points. To control the diversity of the bounded archive, several methods have been proposed, and they can be classified into two categories depending on the space considered (either the objective space or the solution space). Concerning the methods related to the objective space, we find the adaptive grid archiving strategy suggested by Knowles and Corne [2003] and a hypervolume archiving strategy presented by Knowles et al. [2003]. The grid archiving strategy splits the objective space into a homogeneous grid, and when the archive is full, to add a new non-dominated solution, one solution from the cell with the highest density is removed. The hypervolume archiving strategy eliminates the solution that contributes the least to the hypervolume when a new solution is added. Another strategy consists of evaluating the density of the current population by attributing a *crowding-distance* attribute to each solution reflecting how close are the closest solutions for each objective direction. The procedure is described by Deb et al. [2002]. It requires sorting the population according to each objective function value in ascending order of magnitude. Considering each objective function, the boundary solutions (i.e., with the smallest and largest function values) received an infinite distance value, ensuring they are always kept (indeed, we do not want to lose the extremal solutions). All other intermediate solutions received a distance value equal to the absolute normalized difference in the function values of two adjacent solutions (the one before and the one after for the considered objective). This calculation is performed for each objective function. Finally, the overall crowding-distance value is expressed as the sum of individual distance values corresponding to each objective. Preferred solutions are those with the highest crowding distance, indicating that the region of the front contains fewer solutions.

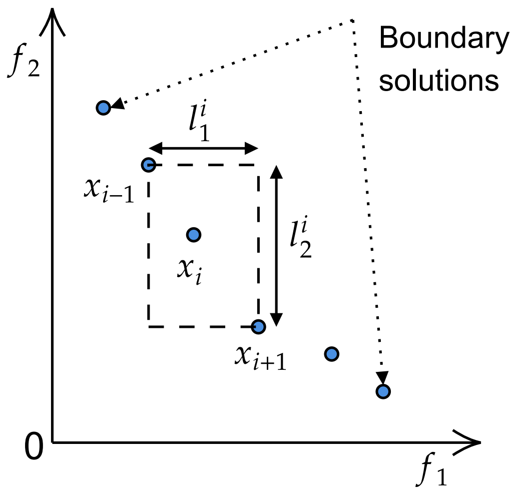


Figure 1.10: The crowding-distance of x_i is $\tilde{l_1^i}+\tilde{l_2^i}$, where $\tilde{l_k^i}$ is the normalized value of l_k^i. Boundary solutions receive an infinite crowding-distance.

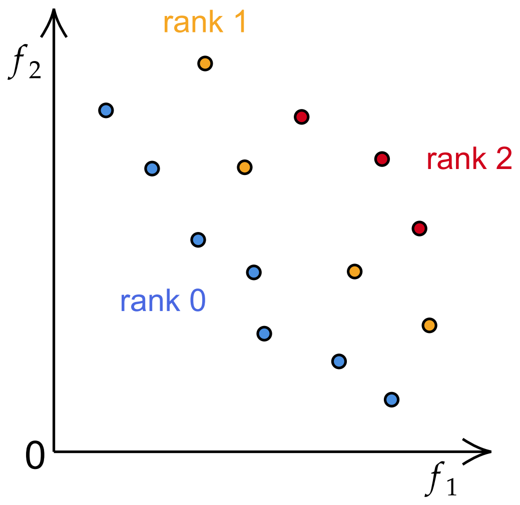


Figure 1.11: Pareto ranks of a population of solutions, in a minimization context. Solutions of rank 0 are non-dominated.

Figure 1.10 illustrates the concept of crowding-distance in a bi-objective context, although it can be used for any number of objectives.

When considering the diversity in the solution space, new metrics are necessary to evaluate the closeness between solutions, like the Hamming distance (which counts the number of differences between the representation of two solutions), or operator-based distances. An operator-based distance requires a neighborhood operator that can produce any solution starting from any solution in a finite number of applications, then it is possible to define a distance between two solutions based on that operator, corresponding to the minimum number of times that the operator has to be applied to go from one solution to the other one. Once a neighborhood is defined in the solution space, leading to a measure of the decision space diversity, it is possible for example, to integrate that measure of diversity into the hypervolume indicator to optimize both convergence and diversity in the solutions found [Ulrich et al., 2010].

1.3.4.2 Multi-Objective Evolutionary Algorithms

Nature-inspired algorithms are population-based metaheuristics, making them a natural choice for solving a multi-objective problem. However, since there are (in general) no *best* solutions, strategies have been developed to overcome the difficulty of maintaining a good population of solutions. Moreover, most of the time the size of the population considered is limited, requiring the use of bounded mechanisms to keep the most interesting solutions.

One first strategy is to consider the different objectives separately. For instance, the *Vector Evaluated Genetic Algorithm* (VEGA) is a genetic algorithm designed to solve multi-objective problems. The approach was proposed by Schaffer [1985]. In VEGA, at each iteration, the population is divided into n subpopulations, where n is the number of objectives. Then, solutions from each subpopulation are selected (with usual selection mechanisms since only one objective is considered) and mixed to obtain a smaller population to which genetic operators are applied.

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Deb et al., 2002], uses Pareto ranks to stratify the solutions found. These ranks are sequential integer values that represent the layers of stratification in the population obtained via dominance testing. Consequently, individuals assigned to rank 1 are non-dominated, and inductively, those of rank $i + 1$ are dominated by all individuals of rank 1 through i. The concept is presented in Figure 1.11. In practice, attributing a Pareto rank to a new solution requires a high computational effort (particularly if the population contains many individuals). Non-dominated solutions have a rank of 0, among the remaining solutions, the non-dominated ones have a rank of 1, and so on. In addition to the Pareto ranks, the NSGA-II algorithm benefits from the use of the crowding distance to select the most promising solutions for the next generation: the solutions are sorted by ascending Pareto rank, and with equal ranks, solutions with higher crowding distance are preferred. Once the solutions are selected, the next population is obtained by using genetic mechanisms (crossover and mutation).

Instead of evaluating each solution individually, one possibility is to evaluate the set of solutions directly by using an indicator introduced in Section 1.3.2. Then, it is possible to use that indicator to guide the search and to update the population. The GD-MOEA [Menchaca-Mendez and Coello Coello, 2015] is a Multi-Objective Evolutionary Algorithm that uses the Generational Distance indicator to update the population. The diversity in the decision space can be additionally controlled by using an appropriate measure with the hypervolume indicator [Ulrich et al., 2010].

Many more evolutionary algorithms adapted to multi-objective optimization are presented in the book of Deb [2001].

1.3.4.3 Multi-objective Local Search Algorithms

In multi-objective optimization, the notion of neighborhood remains as important as in single-objective optimization. Indeed, using an intensification step allows algorithms to converge faster toward non-dominated solutions. More precisely, the idea is to intensify the search in a region of a solution previously found. This concept is exploited by *Multi-Objective Local Search* (MOLS) [Blot et al., 2017]. MOLS are further developed in Chapter 6.

Briefly, a MOLS is interesting in optimizing simultaneously many solutions, allowing a better exploration of Pareto sets. Like neighborhood-based metaheuristics, it requires defining neighborhood operators to intensify the search. However, the acceptance criterion of a neighbor solution has to be adapted to the multi-objective context. In general, one can accept a dominating solution, or a non-dominated solution considering the current population. Among the MOLS we find the *Pareto Local Search* (PLS) [Paquete et al., 2004], which starts from an initial Pareto set, explores neighborhoods of solutions in this set, and then updates the Pareto set with new solutions found. Liefooghe et al. [2012] proposed the concept of Dominance-based MOLS (DMLS), generalizing the PLS.

1.3.4.4 Hybrid Algorithms

Finally, we find algorithms that solve multi-objective problems by considering decomposition strategies or one objective at a time. Such algorithmic strategies can be seen as hybrid strategies. Moreover, these strategies can also be adapted to an interactive resolution context, where the decision-maker can provide feedback during the process.

The Multi-Objective Evolutionary Algorithm based on Decomposition [Zhang and Li, 2007], called MOEA/D, is an algorithm widely studied in the literature [Xu et al., 2020]. MOEA/D is a genetic algorithm approximating the Pareto front by decomposing the multi-objective problem into several scalar objective subproblems. MOEA/D is further detailed in Chapter 3. Briefly, instead of using a single scalarization to model the multi-objective problem as explained in Section 1.3.3, several scalarizations are considered (whose choice may be influenced by the decision-maker) to approximate a region of the objective space.

We can also find *lexicographic* methods, where objectives are each assigned a priority value (randomly or by the decision-maker), and the problems are solved in order of decreasing priority. Since it is not easy to define an interesting priority between objectives, Castro-Gutiérrez et al. [2009] proposed a dynamic strategy where the priority of each objective is adapted during the resolution.

A last common scalar approach is the ϵ-constraint method, described by Chankong and Haimes [2008] in a bi-objective context. In this case, the focus is made on one objective function which is optimized, while the other objective function is considered as a constraint expressed as $f_i(x) \leq \epsilon_i$. When the solution can not be further improved the constraint is updated by modifying the ϵ value with a constant value. This method has been improved by Laumanns et al. [2006] to be used when an arbitrary number of objectives is considered, and where the constraints are dynamically adjusted. Epsilon constraint methods can be used as exact algorithms (if an exact solver is available for the single-objective problem) to obtain the optimal Pareto front.

1.4 Knowledge Discovery

The machine learning field has become incredibly popular in the last decade. Moreover, it is known that machine learning and combinatorial optimization have a good synergy [Corne et al., 2012]. For example, when performing a clustering task several criteria (like specificity and sensitivity) can be considered to improve the classification capability [Ostaszewski et al., 2009]. In that case, multi-objective tools are used to obtain interesting clusters. On the other side, knowledge discovery can be beneficial to the resolution of combinatorial optimization problems. In particular, the use of knowledge can help to speed up the search process (e.g., by detecting promising areas of the search space to explore), improve the quality of the results (e.g., by learning characteristics of good solutions), or tune the algorithms (i.e., choose the best parameter values for an algorithm).

Knowledge discovery mechanisms can be integrated into metaheuristics to solve optimization problems at three levels according to Talbi [2021]: at a problem-level (Section 1.4.1), a low-level (Section 1.4.2), and a high-level (Section 1.4.3).

Each integration can be realized either *online* or *offline* [Corne et al., 2012]. The learning is performed online when it occurs during the execution of the algorithm. Otherwise, the learning is said offline.

Concerning the learning mechanism itself, it is generally composed of two steps: an *extraction* step, where some knowledge is extracted from something related to the problem studied (e.g., solutions, instances), and an *injection* step, where the extracted knowledge is used to guide the algorithm towards promising solutions. In addition, we call *memorization* the step performed during or after the extraction, where the knowledge is stored or updated. The analysis of different hybridization between learning mechanisms and heuristics leads to four questions that have to be considered: *What*/*Where*/*When*/*How* is the knowledge extracted/injected?

Question *What* is problem-dependent, since each problem may have specific relevant knowledge. Question *Where* is algorithm-dependent, since the extraction and injection steps have to be integrated into the process of the algorithm. More precisely, the position of the extraction step in the algorithm highly influences what is learned. Question *When* is algorithm-dependent as well and deals with the frequency of applying the extraction and injection steps. But it also concerns *when* the learning mechanism starts. Indeed, starting the learning mechanism too early may result in poor learning, since initial solutions are generally of low quality. Question *How* corresponds to the design of the memorization step, requiring appropriate data structures. It also considers which knowledge is exploited during the injection step.

1.4.1 Problem-Level Integration

The idea behind a problem-level integration of knowledge is to help the heuristic to better understand the problem to solve. In particular, it takes into account the characteristics of the problem itself (e.g. the format of instances) to guide the algorithm. Knowing the characteristics of an instance can provide additional information when tuning the algorithm if results are available on other (similar) instances. The analysis can also help to better understand the behavior of heuristics. In particular, to detect the similarity between instances, it is possible to extract features from the problem, which depend only on the instance solved. By analyzing these features (e.g., with a principal component analysis), useful information can be discovered about a benchmark, which may help to generate new instances with different characteristics to complete the existing benchmarks. In the context of routing problems, some features have been proposed by Arnold and Sörensen [2019], to better understand the structures of good solutions depending on the instance features. Such analyses are generally performed offline (i.e., before the resolution of the considered problem).

Problem-level approaches can also be helpful when the objective function and/or constraints are not explicitly defined or are not easily computable (e.g., in black-box optimization problems). Such situations arise when real-life problems are modeled. In such cases, surrogates are generated to approximate the functions.

With a problem-level integration, one can learn to decompose a problem into several (easier) subproblems, considering either the decision space (e.g., to remove inter-relationship between variables) or the objective space (e.g., to reduce the number of objectives in multi-objective optimization).

When considering dynamic problems, the concept of transfer learning can be used to speed up the search process, since solutions of a close problem are already known. In a multi-objective context, it can be used to approximate the new Pareto front knowing the former one [Jiang et al., 2017a].

1.4.2 Low-Level Integration

Low-level integration of knowledge into a metaheuristic is useful to improve the components of the metaheuristic itself (e.g., it can help with intensification and diversification). Low-level mechanisms learn from the solutions generated when solving the problem (e.g., they can exploit the structure of the solutions).

The first component that can be improved in a metaheuristic is the initialization procedure. By using appropriate knowledge, it is possible to create interesting solutions from the beginning of the executions. For example, [Joshi et al., 2019] trained a Graph Convolutional Network with valid TSP tours to generate a heat map that represents the probability of each edge to be taken in a TSP tour. Valid solutions can then be constructed by following the probabilities obtained, which can serve as initial solutions. On the other hand, when many solutions are available, and when it is possible to exploit the structure of the solutions easily, the concept of vocabulary building can be considered [Taillard, 2023]. It consists of grouping in a dictionary pieces of solutions (called *words*) and exploiting the pieces to form new solutions, based on what has already been seen. For routing problems, the Pattern Injection Local Search (PILS) mechanism proposed by Arnold et al. [2021], exploits sequences of consecutive customers, recalling the concept of vocabulary building. The PILS mechanism is further developed in Section 2.8.

Adaptive mechanisms, reacting to the search space explored, can be considered as low-level mechanisms. For example, in adaptive variable neighborhood search, the selection of a neighborhood operator is conditioned to its performance. With a tabu list, elements of solutions to avoid are learned. In ant colony optimization, the ants (i.e., the solutions learn from their neighborhood). Bandit mechanisms and Monte Carlo search can also be used to have a good trade-off between exploration (i.e., diversification) and exploitation (i.e., intensification). Arnold and Sörensen [2019] developed a knowledge-guide local search for the capacitated vehicle routing problem where *bad* edges are detected and removed during the search to explore solutions more efficiently. In particular, the objective function is changed periodically to penalize the bad edges.

Over the past years, reinforcement learning has been widely studied to create efficient operators for single-objective problems such as the TSP or VRP's [Kool et al., 2018, Xing and Tu, 2020, Falkner and Schmidt-Thieme, 2020]. However, the training requires a lot of computational resources (in time and memory), and for now, such methods do not overcome classical metaheuristics on large instances.

The discovery of relevant features can also be helpful during the search phase. Knowing which features characterize good solutions or good regions of the exploration space is the key to guiding the algorithms toward promising areas. One can learn such features in an online manner with a decision tree, producing rules to guide the current solution into an interesting space (or to avoid a specific space). This approach was proposed by Lucas et al. [2020] for a routing problem.

The Estimation Distribution Algorithms (EDA) estimate the relations between the variables of a problem by learning a probability distribution associated with each solution of the population. The probability distribution is generally estimated by using Bayesian networks [Lozano, 2006]. More recently, the concept has been adapted to permutation-based solutions, by using linkage genes [Guijt et al., 2022].

Bandaru et al. [2017] review many knowledge discovery concepts used in the continuous multi-objective optimization field. Among the concepts presented, we find the automated *innovization* [Bandaru and Deb, 2013], which is an unsupervised learning algorithm that can extract knowledge from multi-objective optimization sets of solutions in the form of analytical relationships between the decision variables and objective functions. The term innovization, short for innovation through optimization, was coined by Deb and Srinivasan [2006].

1.4.3 High-Level Integration

The interest behind a high-level approach is to design new heuristics by selecting the most relevant operators available. In practice, we define a set of common operators used in the literature, and at each step, we try to select the best operator to pursue optimization.

This idea has already been investigated for single-objective problems. For example, an online integration is proposed by McClymont and Keedwell [2011]. They use hidden Markov models to simulate the selection of operators, and then choose the one with the highest probability. An offline integration is also proposed by Yates and Keedwell [2019]. They have data representing many sequences of operators and the result obtained when applying the corresponding sequence. Hence, they can learn which sequences are the most relevant to solving a given problem. Such ideas can be extended to multi-objective optimization since it only requires operators for the problem and an evaluation function (e.g., the gap of one indicator presented in Section 1.3.2).

1.5 Conclusion

In this chapter, we introduced all the concepts required to fully understand the next chapters. More precisely, we started with a description of (single-objective) combinatorial optimization problems, which are a classical manner to formalize real-life problems. This model can be enriched by considering several objective functions together, leading to the concept of multi-objective combinatorial optimization. Such problems require new tools to compare and evaluate the quality of the returned solutions. In order to improve the efficiency of existing algorithms, knowledge discovery mechanisms can be integrated. Such mechanisms use machine learning tools in algorithms to exploit specificities of the problem (online or offline, and at different levels).

Chapter 2

Vehicle Routing Problem with Time Windows

Contents

2.1 Introduction

With the increasing demand in terms of services (transportation, delivery, storage), comes an explosion of new challenges that have to be faced. Most of them require taking into account several conflicting aspects (some are ecological, economical, or societal), each one characterized by an objective function to optimize. The Vehicle Routing Problem with Time Windows (VRPTW) is a routing problem, where vehicles have to serve customers within a precise time interval. This problem was introduced decades ago as a case study [Solomon, 1987], and since then many objectives have been investigated for this problem.

The Traveling Salesman Problem (TSP) is probably the simplest, yet difficult, routing problem to solve. Given a set of N customers, the objective is to design a cycle going through all customers exactly once (i.e., a *Hamiltonian* cycle) minimizing the total cost of the tour. Generally, the cost between two customers is the Euclidean distance between them. The m-TSP variant offers the possibility to use up to K vehicles to serve all the customers exactly once. The Capacitated Vehicle Routing Problem (CVRP), considers, in addition, a demand for each customer, and each vehicle has a maximum capacity that must not be exceeded when delivering all the customers of a tour. Finally, the VRPTW adds a time window for each customer indicating when the customer should be served.

All of these problems are known to be NP-complete (i.e., NP and NP-hard), meaning that there currently does not exist a deterministic algorithm that finds an optimal solution to these problems in polynomial time. In particular, an instance of size N of the TSP already contains $(N-1)!$ distinct solutions (when the origin of the tour is fixed), preventing the use of a naive exhaustive search. Using exact algorithms to solve a large instance of the VRPTW is often intractable in practice. That is why we need to use metaheuristics to quickly approximate the optimal solutions.

The VRPTW is formally introduced in Section 2.2. Experimental benchmarks are presented in Section 2.3. We discuss the bi-objective formulation of the problem in Section 2.4. Section 2.5 describes how solutions are represented and evaluated. The neighborhood associated with a solution is presented in Section 1.2.1. Related works are described in Section 2.7. Finally, the Pattern Injection Local Search (PILS) mechanism, which learns structures of solutions, is detailed in Section 2.8.

2.2 Model and Notations

The VRPTW can be formally described as follows [Toth and Vigo, 2014]. Let N denote the number of destinations that need to be visited from one point of origin. In the following, we will use the standard terminology and call destinations *customers* and the point of origin *depot*. Every customer is usually identified with a non-negative integer in $\{1, \ldots, N\}$ and the depot with 0. The routing problem can be modeled as a graph $G = (V, A)$, where V denotes the set of nodes (i.e., the customers and the depot) of size $N + 1$. A represents the set of arcs (usually the graph is considered *complete* meaning that it is possible to visit any customer from any other one). Each arc (i, j) (reflecting the connection between nodes i and j) is annotated with a cost $c(i, j) = c_{ij}$ that is realized if the arc is traversed by a vehicle. This cost usually corresponds to the Euclidean distance between two nodes, however, it can also be replaced by any other metrics. In general, the instance is *symmetric*, that is, the cost to go from i to j is the same of the cost to go from j to i. All the costs are stored in a cost matrix C. A *path* in G is a sequence of adjacent arcs $((i, j), (j, k), \ldots)$ so that all visited nodes are distinct. A *route* in G is then defined as a path that starts at 0, and has an additional edge back to 0, i.e., $((0, i), (i, j), \ldots, (k, l), (l, 0))$, which is also written $(v_0, v_1, \ldots, v_{|r|}, v_{|r|+1})$ with

$v_0 = v_{|r|+1} = 0$ and $|r|$ denotes the length of the route. In the existing literature on the VRPTW, the number K of vehicles available to serve the customers is usually considered unlimited, which is represented by a large value of K.

In many applications, the vehicles have to respect limitations. For instance, a truck can only transport a given amount of parcels. Formally, these constraints are modeled by allocating a demand q_i to each customer i. The sum of the demand of all customers on a route may not exceed a certain value $\mathcal{Q}$. This limitation is commonly called *capacity constraint.* For instance, q_i could represent the number of parcels that have to be delivered to customer i and $\mathcal{Q}$ could define the limit of parcels that can be transported in one vehicle. In particular, the fleet of vehicles is generally considered *homogeneous*, meaning that all vehicles have the same capacity.

Concerning the time constraints, the service at each customer must start within an associated time interval, called a *time window.* The two most frequent time windows studied in the literature are hard and soft time windows. With *hard* time windows, a vehicle that arrives too early at a customer must wait until the customer is ready to start the service. On the contrary, vehicles are not allowed to arrive late. With *soft* time windows, a vehicle can violate a time window by paying a penalty. With soft time windows, it is easier to find feasible solutions to the problem, since the violation of the time constraint is added to the function to optimize. In the following, we only consider hard time windows. Formally a time window $[a_i, b_i]$ is associated with each customer i. In a time window, a_i, called *ready date*, is the earliest delivery time for customer i, and b_i, called *due date*, is the latest possible time to deliver customer i. Moreover, each customer i has a service time t_i^s which represents the duration of the service asked (for instance the duration of the delivery, it may estimate the time needed to get the parcel in the truck and then bring it to the customer). We assume that the depot has a zero service time $t_0^s = 0$. A time matrix $T = (t_{ij})$, where coefficient t_{ij} denotes the time needed to go from customer i to customer j, is provided for each VRPTW instance. Note that for some instances, we consider $t_{ij} = c_{ij}$.

To properly define the objective function and the constraints, the following notations are required. We define x_{ij} which is equal to 1 if the customer j is served exactly after the customer i, and 0 otherwise. Note that according to this definition we can not have $x_{ij} = 1$ and $x_{ji} = 1$. In addition to that, we define a time variable T_i^r for each customer i and route r specifying the start of service time at customer i when serviced by route r. A solution x, that is, a set of routes, is now linked with a matrix $\mathcal{M}$ of size $N+1$ such that $\mathcal{M}(x)_{ij} = c_{ij} \cdot x_{ij}$. Hence the total transportation cost of a solution x for the graph G is defined as follows:

$$f_1(x) = \sum_{i=0}^{N} \sum_{j=0}^{N} \mathcal{M}(x)_{ij} = \sum_{i=0}^{N} \sum_{j=0}^{N} c_{ij} \cdot x_{ij} \tag{2.1}$$

The problem can be expressed as the following mathematical model where subtour elimination constraints come from the Miller-Tucker-Zemlin (MTZ) formulation:

$$\min_x f_1(x) \tag{2.2}$$

$$s.t. \sum_{j=0, j\neq i}^{N} x_{ij} = 1 \qquad \forall i \in \{1, \dots, N\} \tag{2.3}$$

$$\sum_{i=0, i\neq j}^{N} x_{ij} = 1 \qquad \forall j \in \{1, \dots, N\} \tag{2.4}$$

$$\sum_{j=1}^{N} x_{0j} \leq K \tag{2.5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \tag{2.6}$$

$$u_i - u_j + \mathcal{Q} x_{ij} \leq \mathcal{Q} - q_j \qquad \forall (i, j) \in A \tag{2.7}$$

$$q_i \leq u_i \leq \mathcal{Q} \qquad \forall i \in \{1, \dots, N\} \tag{2.8}$$

$$T_i^r + (t_i^s + t_{ij}) x_{ij} \leq T_j^r + b_0(1 - x_{ij}) \qquad \forall r \in \{1, \dots, K\}, (i, j) \in A \tag{2.9}$$

$$a_i \leq T_i^r \leq b_i \qquad \forall r \in \{1, \dots, K\}, i \in V \tag{2.10}$$

Constraints (2.3) and (2.4) indicate that each customer is connected to two other vertices of the graph. Constraint (2.5) ensures that at most K vehicles are used. In constraints (2.7) the additional variables $u_1, \dots, u_N$ indicate the accumulated demand u_i already distributed by the vehicle when arriving at customer i. Constraints (2.9) premise that the service at customer j can not start before the service at customer i is finished, if i and j are served one after the other by the same route. Note that if $x_{ij} = 0$ this constraint is always verified because b_0 corresponds to the closing of the depot. The Time windows are respected with constraints (2.10). Again with these two constraints, we prevent the creation of subtours.

Finally, solving a VRPTW instance over a complete graph G, aims to find a set of at most K routes that serve all customers exactly once, respect capacity and time constraints, and minimize the sum of the costs of the involved arcs.

An instance of the VRPTW is illustrated in Figure 2.1. The depot contains $K = 3$ vehicles, with a capacity of $\mathcal{Q} = 10$. There are $N = 8$ customers. Customers have their own demand and time window. Between each customer i and j there is a cost $c_{ij} = 1$ and a travelling time $t_{ij} = 1$. Here the service time of each customer i is set to $t_i^s = 1$. A feasible solution of this instance is illustrated in Figure 2.2. This solution is even optimal regarding the number of vehicles and the total transportation cost.

Many variants of routing problems have been studied over the years, most are described by Laporte [2009], Toth and Vigo [2014], and Braekers et al. [2016]. Laporte presents 50 years of work on routing problems and regroups all existing variants before 2009. Toth and Vigo present classical and new routing problems, with recend advances to solve them. Braekers et al. present a taxonomy to classify recent routing problems, so that, between 2009 and 2016. In particular, the VRPTW can also be extended to form more complex problems, by considering, for example, more than one depot (MDVRPTW), or multiple time windows for each customer.

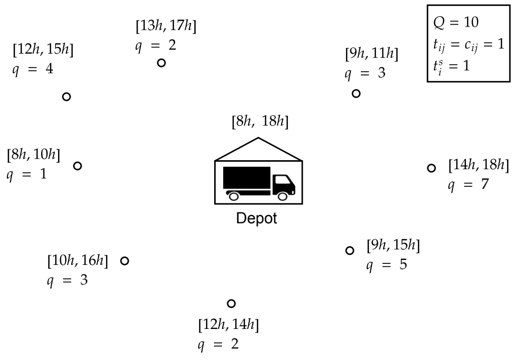


Figure 2.1: Example of VRPTW instance.

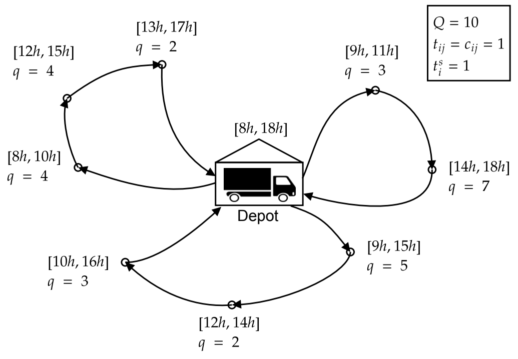


Figure 2.2: A feasible solution.

2.3 Benchmarks

In this section, we present the different benchmarks of instances used. There exist two main benchmarks for the VRPTW, the Solomon one (Section 2.3.1) and the Gehring and Homberger one (Section 2.3.2). A set of real-life instances is described in Section 2.3.3. Additionally, we generated a set of instances, presented in Section 2.3.4, to perform additional experiments related to tuning different algorithms.

2.3.1 Solomon

Solomon's benchmark [Solomon, 1987] is a set of VRPTW instances, frequently used in the literature to evaluate the performance of multi-objective algorithms [Ghoseiri and Ghannadpour, 2010, Qi et al., 2015, Moradi, 2020]. The benchmark contains 56 instances, each with 100 customers, but restrictions of 25 and 50 customers are available too. The instances are divided into three categories according to the type of generation used: R (random), C (clustered), or RC (random-clustered). The R category (23 instances) contains instances where customers are randomly located in a 100×100 grid, while the instances of category C (17 instances) contain clusters of customers. The category RC (16 instances) contains instances where half the customers are randomly located and the other half is clustered. Each category is divided into two classes, either 1 or 2, according to the width of time windows. Instances of class 1 have tighter time windows than instances of class 2, meaning that instances 1 are more constrained.

2.3.2 Gehring and Homberger

The benchmark of Gehring and Homberger [Gehring and Homberger, 1999] extends Solomon's benchmark and considers a larger number of customers. Indeed, it contains instances of size 200, 400, 600, 800, and 1000. These instances are very close (in terms of generation) to the instances of Solomon. Again, there are three categories R, C, and RC, and for each category, two classes (1 or 2), depending on the width of the time windows. Contrarily to Solomon's set, each category of instance is equally represented, meaning that each one contains 10 instances with tight time windows (class 1) and 10 instances with wide time windows (class 2).

2.3.3 Real life Instances

Castro-Gutierrez et al. [2011] developed a set of instances based on data from a real-world distribution company. The generator of instances is available at `https://github.com/psxjpc/`. Concerning the generation of the time windows, five profiles are considered. In every case, the opening time of the depot is 8 hours (i.e., 480 minutes). Three types of customers are considered. Early customers, who want to be served in the morning. Midday customers, who want to be served at midday. And late customers, who want to be served the latest. Each type of customer has a specific time window, depending on the profile chosen for the instance. All the customers have the same probability of having any of the time windows in a specific profile. Note that the service time of a customer is randomly chosen among $\{10, 20, 30\}$. The five profiles are:

1. All the customers are available all the day: only one time window $[0, 480]$;
2. Early customers: $[0, 160]$, midday customers: $[160, 320]$, and late customers: $[320, 480]$;
3. Early customers: $[0, 130]$, midday customers: $[175, 305]$, and late customers: $[350, 480]$;
4. Early customers: $[0, 100]$, midday customers: $[190, 290]$, and late customers: $[380, 480]$;
5. The time window of a customer is chosen among the ten above time windows.

Locations of customers and their demands are provided by the company. It is possible to adjust the capacity of the vehicles with a parameter in $[0, 100]$. With a value close to 0, the capacity of the vehicles will be close to the maximum demand of the customers, consequently, a larger number of vehicles will be required to satisfy the demand constraint. On the other hand, with a value close to 100, the capacity will be close to the total demand, which may lead to the use of fewer vehicles.

Finally, 45 instances were generated in total, 15 for each possible number of customer (i.e., 50, 150 and 250). The authors showed that these instances were more realistic and challenging for multi-objective VRPTW than the Solomon instances.

2.3.4 Generated Instances

In order to allow a fair tuning of the studied algorithms and not to bias the experimental results obtained on Solomon and Gehring and Homberger benchmarks, we decided to generate a new set of instances. We did not use the real-life inspired instances, since they were very different from the two other benchmarks. We followed the generation strategy proposed by Uchoa et al. [2017] (for the demands), with the generation strategy of Solomon [1987] (for the time windows). The generation procedure is defined as follows.

All the customers are located in a 1000×1000 grid. The generation of the customers is *random* (R), *clustered* (C), or *mixed* (M). The random generation randomly selects the location of the customers in the grid. The clustered generation creates clusters of customers. The number of clusters s is randomly selected between 3 and 8. Then s customers are randomly located in the grid, to define the centers of the clusters. A new customer is accepted only if it can be added to one cluster. A customer v is added to the cluster of center s_i with probability $exp(-d_E(v, s_i)/40)$, where d_E is the Euclidean distance. If the customer is not added to any cluster, then it is discarded and a new customer is generated, until the number of customers wanted is reached. For the mixed generation, half of the customers follow the random generation, and the other half the clustered generation.

The depot is located either in the center of the grid or randomly. If the strategy is not specified, the choice is made randomly.

The width of the time window of the depot, that is $b_0 - a_0$ is either *large* (i.e., chosen between 2500 and 5000) or *small* (i.e., chosen between 1500 and 2500). For the creation of the time windows of the customers, we consider that the time required to perform the travel between two customers corresponds to the cost between them (i.e., the Euclidean distance). Naturally, there is no interest in taking an upper bound of the time window exceeding the horizon of the depot, and the upper bound can not be lower than the time required to go from the depot to the customer $\underline{t}$ (i.e., the Euclidean distance between the depot and the customer). A time window can be either *wide* or *tight*, which impacts its width t_w. For wide time windows, the width is picked between 200 and 500, while the width is picked between 50 and 200 for tight time windows. The center t_c of the time window is chosen uniformly between 0 and the horizon of the depot. Then the due date (b_i) of the i-th customer is set as $max(min(t_c + t_w/2), b_0), (\underline{t} * 11)/10))$. The ready date ($a_i$) is obtained with the width of the time window.

The service time of a customer is either *long*, between 100 and 200, or *short*, between 50 and 100. Each customer has the same probability of having a long or a short service time.

The demand of a customer is either *large*, between 50 and 100, or *small*, between 1 and 50. Each customer has the same probability of having a large or a small demand.

To define the maximal capacity $\mathcal{Q}$ allowed for all vehicles, we first estimate the average number of customers per route by picking an integer between $N/15$ and $N/5$. The value is then multiplied by the average demand of the customers generated, and if the value obtained is lower than the maximal demand of the customers, then the capacity is set to the maximal demand increased by 10%.

Finally, we generate 8 instances of 100 customers for each triplet of parameters containing the generation of the customers (R, C, or M), the horizon of the depot (L or S), and the width of time windows (W or T). It leads to the creation of 96 new instances. As an example, the instance $MLW2$ is the second instance with a mixed generation of customers, a large horizon, and wide time windows.

2.4 Additional Objectives

A wide range of objectives has already been investigated in the VRPTW context. This section provides a broad overview of the most common objectives found in the literature. It leads to many possibilities to solve a bi-objective problem, but our choice is to consider the total transportation cost and the total waiting time of drivers.

Following the classification of Jozefowiez et al. [2008], these objectives are divided into three categories according to the component of the problem they are associated with: the route (Section 2.4.1), the node or arc activity (Section 2.4.2), and the resources (Section 2.4.3). Section 2.4.4 formally describes the total waiting time objective, and finally, the two objectives optimized are motivated in Section 2.4.5.

2.4.1 Objectives Associated with the Route

Among the objectives related to the route, the most common is surely minimizing the cost of the solutions generated [Schneider et al., 2017]. For instance, the cost can be the distance traveled or the time required. It can also be something more complex, like CO_2 emissions. More generally, minimizing cost is linked to an economic criterion. When the total cost is ignored, the makespan (i.e., minimizing the length of the longest route) is an objective frequently minimized [Zhou and Wang, 2014]. These objectives are motivated by the applications of the problem.

2.4.2 Objectives Associated with Nodes or Arcs

Concerning the objectives related to the node or arc activity, they often involve time windows, since it is the most constraining part of the problem in general. When considering *hard* time windows (i.e., that cannot be violated), the driver's waiting time due to earliness can be optimized [Zhang et al., 2019]. If we consider *soft* time windows (i.e., that can be violated with a penalty), the delay time of drivers is more likely to be optimized [Zhou and Wang, 2014]. More generally, the number of violated time windows can also be minimized [Geiger, 2003].

2.4.3 Objectives Associated with Resources

The last category of objectives concerns resources. The main resources encountered in the literature are vehicles and goods. The minimization of the number of vehicles often appears in classical benchmarks and can be interpreted economically, in that fewer vehicles means less monetary investment. More precisely, this objective is often minimized first in Solomon's and Gehring and Homberger's benchmarks [Solomon, 1987, Gehring and Homberger, 1999], since buying a truck or hiring a driver is the most expensive part. Finally, some objectives try to erase disparities between tours, to bring a *fairness* aspect into the problem. To define a balancing objective it is necessary to define a route's workload, which can be expressed as the number of customers visited or the number of goods delivered [Melián-Batista et al., 2014, Baños et al., 2013].

2.4.4 Focus on the Total Waiting Time

When drivers arrive before the opening of a time window they must wait until the opening time. It increases the time of the route for the driver and may incur satisfaction issues.

We formalize the notion of waiting time as follows. The waiting time W_i at a customer i is given as the maximum between 0 and the difference between the opening of the time window a_i and the arrival time T_i at location i, that is $W_i = \max\{0, a_i - T_i\}$. Note that each route $r = (v_0, v_1, \dots, v_{|r|}, v_{|r|+1})$ is associated with a feasible (i.e., consistent with traveling times and time windows) arrival time vector $T_r = (T_{v_0}, T_{v_1}, \dots, T_{v_{|r|}}, T_{v_{|r|+1}})$ and the total waiting time $W_r(T_r)$ on route r, with respect to T_r is given by $W_r(T_r) = \sum_{i=1}^{|r|} W_{v_i}$. Thus the total waiting time of a solution $x = \{r_1, \dots, r_K\}$ on a graph G, given a time arrival vector for each route in the solution, i.e., $T_x = (T_{r_1}, \dots, T_{r_K})$, is given by the following formula:

$$f_2(x) = \sum_{k=1}^{K} W_{r_k}(T_{r_k}) \tag{2.11}$$

2.4.5 Bi-objective VRPTW Considered

As presented above, there exist plenty of objectives that can be optimized when solving a VRPTW. Originally, the VRPTW was a bi-objective problem, where the number of vehicles is minimized first and then the total transportation cost is minimized. Indeed, using an additional vehicle has a bigger impact on the total cost for the company.

Castro-Gutierrez et al. [2011] studied how five common objectives (the total transportation cost, the number of vehicles, the makespan, the total waiting time, and the total delay time in case of soft time windows) were correlated. In particular, they showed that in Solomon's instances, the total transportation cost and the total waiting time were weakly correlated in clustered instances with tight time windows and in random instances with wide time windows. In addition, the objectives are in harmony (i.e., the minimization of one tends also to decrease the other) on clustered instances with wide time windows, whereas they are conflicting (i.e., the minimization of one tends to increase the other) on random instances with tight time windows. The total waiting time due to early arrival has also been studied by Zhou and Wang [2014], where a five-objective VRPTW is solved with an objective-wise local search.

Moreover, it seems interesting to study together two continuous objectives (here the total transportation cost and the total waiting time) instead of using a discrete objective, like the number of used vehicles. Indeed, with two continuous objectives, the fronts generated by the algorithm contain, generally, much more non-dominated solutions, that are relevant when using knowledge discovery methods.

As a consequence, we propose to focus on a bi-objective VRPTW (bVRPTW) where one objective has been highly studied, that is the total transportation cost, and a less studied objective, that is, the total waiting time. Using both of these objectives is interesting in many real-life situations, like food delivery, where the waiting time of a driver impacts the heat of the meals of the next customers, and consequently customer satisfaction. Another typical situation concerns the transportation of people, more precisely when a patient has a medical appointment, we do not want them to wait too long.

2.5 Representation of Solutions and Evaluation

When solving a combinatorial optimization problem, the representation of a solution is a key element. Sometimes, the naive structure adapted to a problem is not well adapted to existing operators. In routing problems, a solution is easily represented as a set of routes, each one containing a subset of customers, being the customers served in the route. However, the combination (by a crossover) of two solutions with such a structure is difficult, preventing the use of genetic algorithms. The development of the Split algorithm by Prins [2004] overcomes this difficulty and allows the representation of a solution as a permutation of the N customers of the problem. This algorithm is described in Section 2.5.1.

Another common issue when dealing with combinatorial optimization problems concerns neighborhood evaluation. Indeed, when exploring the neighborhood of a solution, it is possible not to entirely evaluate a neighbor, if a small move is performed. This is the idea behind the incremental evaluation of a solution presented in Section 2.5.2.

2.5.1 Split Algorithm

The objective of the Split algorithm [Prins, 2004] is to optimally cut the given permutation of customers, called *giant tour*, in a subset of routes. The problem is then reduced to a shortest path problem between the nodes 0 and N of an acyclic graph $G^S = (V, A^S)$, where $V^S = \{0, \ldots, N\}$ and A^S contains one arc (i, j) with cost $c^S(i, j) = c_{0,i+1} + \sum_{k=i+1}^{j-1} c_{k,k+1} + c_{j,0}$ for any feasible route visiting customers $i+1$ to j in the order they appear in the giant tour. The shortest path is computed in $O(NB)$, where B is the average out-degree of a node of G^S (i.e., the average number of feasible trips from one node of the giant tour), with a variant of Bellman's algorithm using dynamic programming. Split is presented in Algorithm 5. During the *while* iteration, the longest route starting from t is built. The construction stops when the load on the route exceeds the capacity $\mathcal{Q}$ of the vehicle. Note that, at the end of each *for* iteration, $p[t]$ contains the cost of the shortest path from 0 to t. The cost of the solution is contained in $p[N]$. The value $pred[i]$ contains the index of the customer starting the route containing the i-th customer in the giant tour. Thus, the *pred* array allows us to retrieve the routes of the solution.

Algorithm 5: Split algorithm (CVRP).

```
   Input: A giant tour (i.e., a permutation of the N customers)
   Output: The cost of the tour and the list of the cuts
1  $p = [0, \infty, \ldots, \infty]$
2  for $t = 0$ to $N-1$ do
3      $(load, i) \leftarrow (0, t+1)$
4      while $i \leq N$ and $load + q_i \leq \mathcal{Q}$ do
5          $load \leftarrow load + q_i$
6          if $i = t+1$ then
7              $cost \leftarrow c_{0,i}$
8          else
9              $cost \leftarrow cost + c_{i-1,i}$
10         if $p[t] + cost + c_{i,0} < p[i]$ then
11             $p[i] = p[t] + cost + c_{i,0}$
12             $pred[i] = t$
13         $i \leftarrow i+1$
14 return $p[N], pred$
```

Note that the version of split presented in Algorithm 5 is only suitable in a CVRP context. In a VRPTW context, when a customer is added to the current tour, we have to check if the time window of the customer is violated. It can be done by using the incremental evaluation equations from Section 2.5.2. Two conditions have to be added to l.4: $TW = 0$ and $D + E + t_{i,0} \leq b_0$. The condition $TW = 0$ checks that the time window of the new added customer is not violated, and the condition $D+E+t_{i,0} \leq b_0$ ensures that adding the customer allows a return to the depot before its closing time. Note that during the execution of Split, the additional variables of Section 2.5.2 are iteratively computed. Moreover, the algorithm has to be also adapted when multiple objectives are considered. In this case, $p[t]$ contains the *best* objective vector. The notion of best vector should be adapted according to the algorithm considered, e.g., in MOEA/D it is possible to consider the best vector regarding the aggregation associated with a solution.

The Split algorithm has been improved to reach a $O(N)$ execution in the size N of the problem [Vidal, 2016]. However, we considered the original version of Prins (adapted to a VRPTW context) in our experimentation, since the instances solved contained at most 200 customers and the capacity $\mathcal{Q}$ of the vehicles remains slower than 1000. Moreover, in a CVRP context, the length of routes grows when the capacity increases, which may give instances where the length of a route is very high. In a VRPTW context, the length of a route is quite small on average, with a maximum peak around 30 customers on a single route for bigger instances. Consequently, the use of the linear version of Split would not significantly improve the performance of our algorithms.

2.5.2 Incremental Evaluation

Incremental evaluation should be used, whenever neighborhoods correspond to the recombination of parts of a solution, called *subsequences*, and when values required to evaluate a solution are easy to compute through recombination. In particular, it allows the evaluation of a neighbor in constant time, after a preprocessing step in $O(N^2)$. In the following, we directly place ourselves in a VRPTW context.

A subsequence of customers is denoted σ. For each subsequence of customers, we compute its minimum duration $D(\sigma)$, minimum time-warp use $TW(\sigma)$, the earliest $E(\sigma)$, and the latest $L(\sigma)$ visit to the first vertex allowing a schedule with minimum duration and minimum time-warp use. The cumulated distance $C(\sigma)$, load $\mathcal{Q}(\sigma)$ and time $T(\sigma)$ are computed too. A *time-warp*, introduced by Nagata et al. [2010b], is a penalty paid to reach the end of the time window, in case of late arrival. These values are straightforward to obtain for a sequence σ^0 involving only one customer v_i. Indeed, $D(\sigma^0) = t_i^s$, $TW(\sigma^0) = 0$, $E(\sigma^0) = a_i$, $L(\sigma^0) = b_i$, $C(\sigma^0) = 0$, $\mathcal{Q}(\sigma^0) = q_i$, and $T(\sigma^0) = t_i^s$.

Now, knowing the characteristics of two sequences $\sigma = (v_i, \dots, v_j)$ and $\sigma' = (v'_{i'}, \dots, v'_{j'})$, it is possible to compute the characteristics of the concatenated sequence $\sigma \oplus \sigma'$ with the equations provided by Vidal et al. [2013].

$$\begin{aligned}
&D(\sigma \oplus \sigma') = D(\sigma) + D(\sigma') + t_{v_j v'_{i'}} + \Delta_{WT}\\
&TW(\sigma \oplus \sigma') = TW(\sigma) + TW(\sigma') + \Delta_{TW}\\
&E(\sigma \oplus \sigma') = max(E(\sigma') - \Delta, E(\sigma)) - \Delta_{WT}\\
&L(\sigma \oplus \sigma') = min(L(\sigma') - \Delta, L(\sigma)) + \Delta_{TW}\\
&C(\sigma \oplus \sigma') = C(\sigma) + C(\sigma') + c_{v_j v'_{i'}}\\
&Q(\sigma \oplus \sigma') = \mathcal{Q}(\sigma) + \mathcal{Q}(\sigma')\\
&T(\sigma \oplus \sigma') = T(\sigma) + T(\sigma') + t_{v_j v'_{i'}}
\end{aligned}$$

where $\Delta = D(\sigma) - TW(\sigma) + t_{v_j v'_{i'}}$, $\Delta_{WT} = max(E(\sigma') - \Delta - L(\sigma), 0)$, and $\Delta_{TW} = max(E(\sigma) + \Delta - L(\sigma'), 0)$.

Note that the value T is required to compute the waiting time, and it was not considered in [Vidal et al., 2013]. With these values, the cost of a route r is obtained with $C(r)$ and the waiting time associated is $D(r) - T(r) - TW(r)$. A route is feasible if $\mathcal{Q}(r) \leq \mathcal{Q}$ and $TW(r) = 0$.

Each time a solution is entirely evaluated, the associated subsequences are computed and stored in a matrix $\mathcal{S}$. Coefficient $\mathcal{S}_{i,j}$ contains the characteristics of the subsequence starting with i and ending with j. When a move is applied during the neighborhood exploration, only the coefficients involved in modified routes are updated in the matrix.

2.6 Neighborhood Operators

The neighborhood of a solution to the problem is generated by three operators: *relocate* (Section 2.6.1), *swap* (Section 2.6.2), and *2opt** (Section 2.6.3). These operators are commonly used

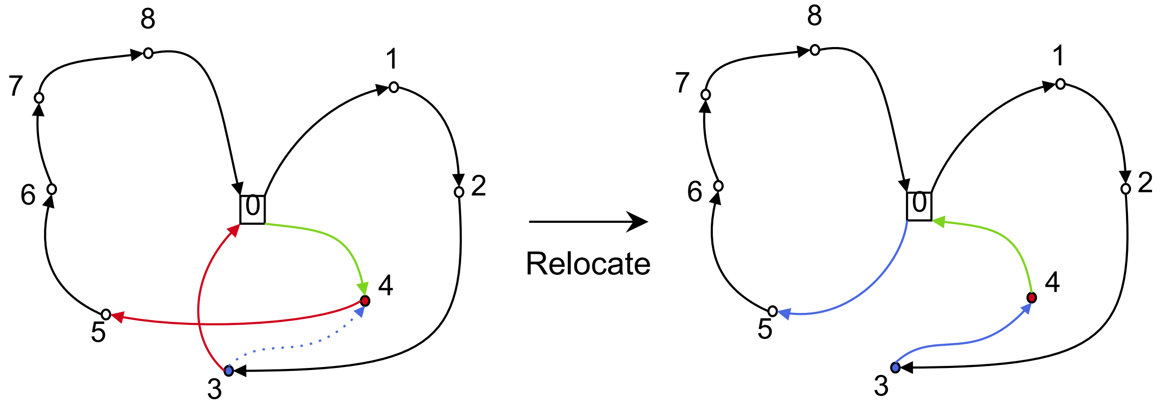


Figure 2.3: The `Relocate` operator transforms the two routes $(0, 1, 2, 3, 0)$ and $(0, 4, 5, 6, 7, 8, 0)$ into the two new routes $(0, 1, 2, 3, 4, 0)$ and $(0, 5, 6, 7, 8, 0)$, where customer 4 is relocated after customer 3. The red arcs are removed, the blue arcs are added, and the green arc is reversed.

in the routing community [Vidal et al., 2013, Schneider et al., 2017] since they involve a large and diverse neighborhood.

2.6.1 Relocate

The relocate operator moves one customer from one location to another one. The new location can be in the same route or not. The size of the neighborhood generated is a $O(N^2)$.

More formally, if the relocation is performed in the same route, let r be the route involved and v_i the customer that is moved after the customer in position j in r. The new route r' is: $r' = r_{0,j} \oplus v_i \oplus r_{j+1,i-1} \oplus r_{i+1,|r|}$, when $j < i$, otherwise it is $r' = r_{0,i-1} \oplus r_{i+1,j} \oplus v_i \oplus r_{j+1,|r|}$.

If the relocation moves v_i^1 from one route r^1 after v_j^2 in an other route r^2, then it produces two new routes $r'^1 = r^1_{0,i-1} \oplus r^1_{i+1,|r^1|}$ and $r'^2 = r^2_{0,j} \oplus v_i^1 \oplus r^2_{j+1,|r^2|}$.

2.6.2 Swap

The swap operator exchanges two customers from two (possibly distinct) routes. The size of the induced neighborhood is $O(N^2)$.

If the swap exchanges the two customers v_i and v_j in the same route r, the new route is $r' = r_{0,min(i,j)-1} \oplus v_{max(i,j)} \oplus r_{min(i,j),max(i,j)-1} \oplus v_{min(i,j)} \oplus r_{max(i,j),|r|}$

If the swap exchanges v_i^1 from r^1 and v_j^2 from r^2, then it produces the two new routes $r'^1 = r^1_{0,i-1} \oplus v_j^2 \oplus r^1_{i+1,|r^1|}$ and $r'^2 = r^2_{0,j-1} \oplus v_i^1 \oplus r^2_{j+1,|r^1|}$

2.6.3 2opt*

The 2opt* operator exchanges two sequences of customers, involving the extremities of two distinct routes. It generates a neighborhood of size $O(N^2)$.

More precisely, if $r^1 = (0, v_1^1, \dots, v^1_{|r^1|})$ and $r^2 = (0, v_1^2, \dots, v^2_{|r^2|})$ are two distinct routes, then 2opt* removes the arc (v_i^1, v_{i+1}^1) from r^1, the arc (v_j^2, v_{j+1}^2) from r^2, and merges the beginning of r^1

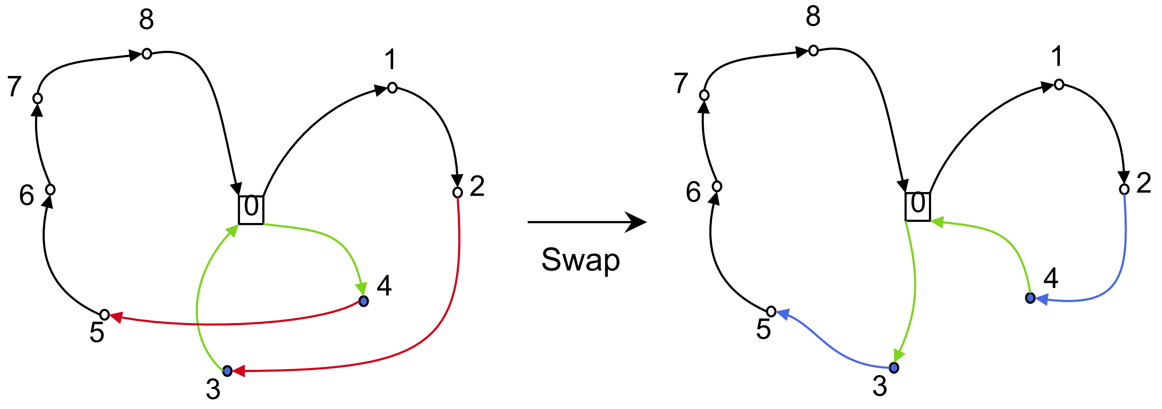


Figure 2.4: The swap operator transforms the two routes $(0, 1, 2, 3, 0)$ and $(0, 4, 5, 6, 7, 8, 0)$ into the two new routes $(0, 1, 2, 4, 0)$ and $(0, 3, 5, 6, 7, 8, 0)$, where customer 3 and 4 are swapped. The red arcs are removed, the blue arcs are added, and the green arcs are reversed.

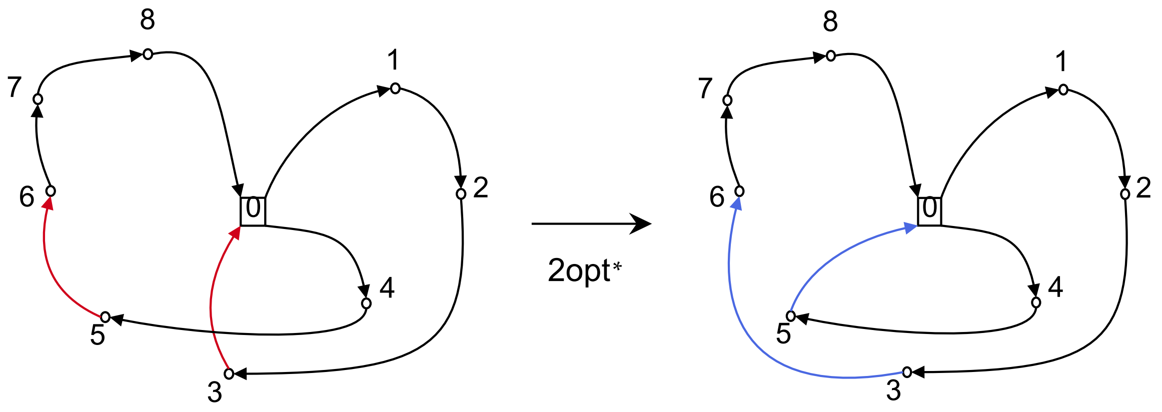


Figure 2.5: The 2opt* operator transforms the two routes $(0, 1, 2, 3, 0)$ and $(0, 4, 5, 6, 7, 8, 0)$ into the two new routes $(0, 1, 2, 3, 6, 7, 8, 0)$ and $(0, 4, 5, 0)$, where arcs $(3, 0)$ and $(5, 6)$ are removed to create arcs $(3, 6)$ and $(5, 0)$. The red arcs are removed and the blue arcs are added.

with the ending of r^2, and the beginning of r^2 with the ending of r^1, which forms two new routes $r'^1 = r^1_{0,i} \oplus r^2_{j+1,|r^2|}$ and $r'^2 = r^2_{0,j} \oplus r^1_{i+1,|r^1|}$

2.7 Works Related to Multi-objective VRPTW

In the following, we provide a broad overview of the existing works concerning the multi-objective VRPTW. Table 2.1 summarizes the existing works, each one associated with the objectives considered, the approach followed and the data set used for experimentation.

Hong and Park [1999], proposed a Goal programming approach and used heuristics to minimize the total travel time and the total customer waiting times. A genetic algorithm is described by Geiger [2001] and a Pareto approach by Geiger [2003]. The first one minimizes the total distance, the number of vehicles, and the total deviations from the time window bounds, while the second one focuses on the minimization of the number of violations instead of minimizing the total deviations. At the same time, an Ant colony system is developed by Barán and Schaerer [2003], which minimizes the total traveling time, the total delivery time, and the number of vehicles. Tan et al. [2006] developed a Pareto approach and a hybrid genetic algorithm to minimize the total route length as well as the number of vehicles. Ombuki et al. [2006] presented a Pareto approach and a weighted sum algorithm to minimize the same objectives. Their algorithm is quite similar to a genetic algorithm, but the fitness is evaluated with a weighted sum of the objectives. In addition to the fitness computed for each individual of the population, they attribute a Pareto rank (see Section 1.3.4.2) to these solutions too. A similar approach [Ghoseiri and Ghannadpour, 2010] is designed with Pareto ranks and weighted sums implemented in a genetic algorithm.

Another interesting algorithm is described by Baños et al. [2013]. It consists of a hybrid metaheuristic (a genetic algorithm and a simulated annealing), where a non-dominated set of solutions is built. More precisely, modified solutions are accepted in accordance with a modified Pareto-dominance criterion which considers the current temperature and the Metropolis function (i.e., a classical function for simulated annealing). Moreover, they initialize their solution with a time windows-based insertion heuristic (TWIH). This heuristic sequentially constructs the routes by first visiting those customers with the earliest time in their time windows (i.e. those available soonest) which are inserted in the vehicles whenever the time windows and capacity constraints are fulfilled. A *Mixed-Integer Linear Programming* (MILP) has been provided by Melián-Batista et al. [2014] to minimize the total distance and balance routes. In the last years, a multi-objective memetic algorithm based on adaptive local search chains was proposed by Zhang et al. [2019] to improve the performances of a multi-objective local search designed for the MO-VRPTW developed by Zhou and Wang [2014]. Briefly, the multi-objective local search is based on the use of objective-wise operators to improve one objective at a time of a random solution from the current population. The idea behind the adaptive local search chain is to use the solution generated during the current local search (which aimed to optimize a specific objective) as a starting point for the next operator, which focuses on another objective. Zografos and Androutsopoulos [2004] described an aggregation heuristic for the VRPTW for hazardous product transportation, which is a real-life extension of the problem, and minimized the risk in addition to the traveled distance. A recent work from Guo et al. [2017] also investigates a real-life extension, based on environmental protection and carbon emission. Qi et al. [2015] proposed a memetic algorithm based on MOEA/D to solve a bi-objective VRPTW. More recently, Moradi [2020] integrated a learnable evolutionary model into a Pareto evolutionary algorithm. The model is based on decision trees that are updated over time.

Authors	Objectives			Approach	Data Set
	Tour	Node	Resource		
Hong and Park [1999]	Total Travel Time	Total Waiting Time	-	Goal Programming	Solomon
Geiger [2001]	Total Distance	Total Deviations	N° of Vehicles	Genetic Algorithm	Solomon
Geiger [2003]	Total Distance	N° of Violations	N° of Vehicles	Pareto Approach	Solomon
Barán and Schaerer [2003]	Total Travel Time	Delivery Times	N° of Vehicles	Ant Colony System	Solomon
Tan et al. [2006]	Total Distance	-	N° of Vehicles	Genetic Algorithm	Solomon
Ombuki et al. [2006]	Total Distance	-	N° of Vehicles	Weighted Sums	Solomon
Ghoseiri and Ghannad-pour [2010]	Total Distance	-	N° of Vehicles	Genetic Algorithm	Solomon
Garcia-Najera and Bulli-naria [2011]	Total Distance	-	N° of Vehicles	EA	Solomon
Castro-Gutierrez et al. [2011]	Total Distance and Total Waiting Time	Makespan and Total Delay Time	N° of Vehicles	EA	Real life
Zhou et al. [2013]	Total Distance	-	Balance Distance	Genetic Algorithm	Solomon
Baños et al. [2013]	Total Distance	Work Load	-	EA	Solomon
Melián-Batista et al. [2014]	Total Distance and Balancing Routes	-	-	MILP	Real life
Chiang and Hsu [2014]	Total Distance	-	N° of Vehicles	EA	Solomon
Zhou and Wang [2014]	Total Distance	Makespan and Total Waiting/Delay Time	N° of Vehicles	Local Search	Real life
Qi et al. [2015]	Total Distance	-	N° of Vehicles	EA	Solomon
Zhang and Li [2007]	Total Distance	Makespan and Total Waiting/Delay Time	N° of Vehicles	Local Search	Real life Solomon
Moradi [2020]	Total Distance	-	N° of Vehicles	LEM (decision tree)	Solomon

Table 2.1: A summary of existing works for multi-objective VRPTW. EA: Evolutionary Algorithm, LEM: Learnable Evolution Model (i.e., EA guided by machine learning), MILP: Mixed-Integer Linear Programming.

2.8 Presentation of the Pattern Injection Local Search

The *pattern injection local search* (PILS) mechanism is an optimization strategy that uses pattern mining to explore high-order local-search neighborhoods. In a routing context, a pattern is a sequence of consecutive customers on a single route. The depot is not considered inside patterns since it belongs to all routes. PILS has introduced a few years ago in [Arnold et al., 2021]. They illustrated its efficiency on the CVRP by integrating the mechanism in two population-based metaheuristics.

The idea behind the mechanism is to discover the structural properties of high-quality solutions by extracting patterns from high-quality solutions generated during the search and then injecting the most frequent ones to improve the quality of a solution. Indeed PILS provides an interesting alternative to traditional local search. Instead of exhaustively exploring large neighborhoods, PILS relies on the information of frequent patterns to select and consider fewer moves that insert a pattern in the incumbent solution and optimally reconstruct the remaining edges to avoid large disruptions. It recalls the *vocabulary building* technique presented in Section 1.4.2, but where *words* are weighted with a score (being their appearance frequency) and where words are used to improve existing solutions.

PILS is based on two main steps: *pattern extraction* (Section 2.8.1) and *pattern injection* (Section 2.8.2). In Arnold et al. [2021], the injection phase was performed just after a perturbation step, while the pattern extraction phase was performed, with a probability, on local optima obtained after the local search. Since PILS was originally designed for the CVRP, it requires some changes for the VRPTW, described in Section 2.8.3.

2.8.1 Pattern Extraction

During the extraction process, only patterns with a size between 2 and s_p are considered. For example in a route $r = (v_0, \dots, r_{|r|})$ starting and ending at the depot each contiguous subsequence of $(r_2, \dots, r_{|r|-1})$ represents a pattern. Hence, a route $(0, 1, 2, 3, 4, 5, 0)$ contains two patterns of size four, that are, $(1, 2, 3, 4)$ and $(2, 3, 4, 5)$. Note that, in the symmetric CVRP, mirrored subsequences are identical (e.g. $(1, 2, 3, 4) = (4, 3, 2, 1)$). Any route r contains $max\{0, |r| - 1 - l\}$ patterns of size l, and any solution contains $O(N)$ patterns of a given size, where N refers to the number of customers in the instance, such that pattern extraction can be done by iterating over the routes as presented in Algorithm 6. An associative array monitors the extracted patterns, along with their frequency of appearance, through the execution of the algorithm, and it is updated each time the extraction step is performed on a solution x.

The success of PILS depends on its ability to extract diverse patterns from high-quality solutions. Indeed, a pool of diverse but low-quality patterns (similar to those found in random solutions) would mainly lead to random moves. In contrast, an overly-restricted set of patterns would lead to few possible injections and would guide towards the same solution characteristics which will result in a loss of diversity. To achieve a meaningful trade-off between these two extremes, the pattern extraction is applied with probability p_{ext} on each local optimum produced by the metaheuristic in which PILS is integrated. Moreover, the probability p_{ext} can also be useful to drive the computational effort needed for pattern extraction without changing the characteristics of the extracted patterns. In particular, the probability was set to $p_{ext} = 0.1$ in the hybridization with the *Hybrid Genetic Search* (HGS) of Vidal et al. [2013].

Algorithm 6: Pattern extraction procedure.

```
  Input: An associative array $\mathcal{A}$, the maximum size of patterns $s_p$, a solution $x$
  Output: The array $\mathcal{A}$ updated with the new patterns
1 for each pattern size $l \in \{2, \ldots, s_p\}$ do
2     for each route $r$ of $x$ do
3         for $i \in \{l+1, \ldots, |r|-1\}$ do
4             $p = (r_{i-l}, \ldots, r_i)$
5             if $p \in \mathcal{A}$ then
6                 Increment the frequency of $p$ by one unit
7             else
8                 Add new pattern $p$ in $\mathcal{A}$
9 return $\mathcal{A}$
```

2.8.2 Pattern Injection

During the injection phase, N_i frequent patterns are tentatively inserted in the current solution to define high-order local search moves. These moves are accepted only in case of improvement. The N_i patterns to inject are randomly chosen among the N_f most frequent patterns extracted. To not bias the injection towards smaller, more numerous, patterns, the size of the patterns injected is randomly selected before the patterns themselves.

The injection of a pattern p into a solution x is described in Algorithm 8. Figure 2.6 illustrates the injection of a pattern into a solution to the CVRP, where routes are not oriented. A pattern p is injected into a solution by connecting the vertices of p and removing all other interfering edges. This leads to the creation of different set of route fragments that need to be reconnected to obtain a feasible solution. $\mathcal{R}_{\text{BEG}}$ (resp. $\mathcal{R}_{\text{END}}$) contains fragments or routes starting (resp. ending) at the depot. $\mathcal{R}_{\text{MID}}$ contains the other fragments. $\mathcal{R}_{\text{INV}}$ contains routes that are not perturbed by the insertion of the pattern. The `Best-Reconnect` procedure, described in Algorithm 7, is optimized to find the optimal reconstruction. $\mathcal{R}_{\text{BEST}}$ is a global variable containing the best (current) set of reconstructed routes. $\mathcal{R}$ contains complete routes. The cost c of a set of pieces of routes is the sum of the cost of the fragments. When fragments are concatenated (operator $\oplus$) the attributes of the new fragment are obtained with the associated incremental evaluation formula from Section 2.5.2 (only total cost and total load are needed). If the optimal reconstruction does not lead to a better solution the pattern is discarded and the next one is tentatively injected. Note that, large patterns tend to create more fragments of route, leading to more important computational resources. Consequently, the maximum size of the patterns extracted should not be too high (e.g., it was $s_p = 5$ in [Arnold et al., 2021]).

2.8.3 Modifications for the VRPTW

Since PILS was originally developed for the CVRP, some adjustments are required to use it for the VRPTW. Reversed patterns can not be considered identical to original patterns since the routes of a solution to the VRPTW are oriented, due to time windows. Moreover, during the `Best-Reconnect` procedure of the injection step, reversed fragments are considered (l.8 of Algorithm 7). There is

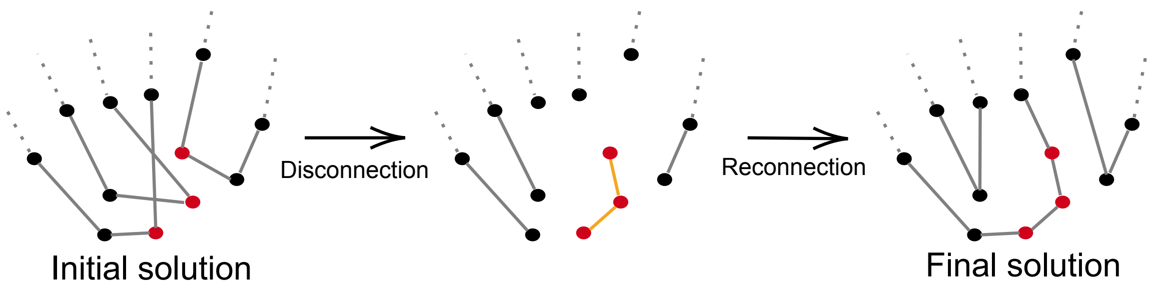


Figure 2.6: Injection of a pattern of size three into a solution to the CVRP. The pattern is represented by red dots and orange edges.

Algorithm 7: `Best-Reconnect` procedure.

```
   Input: $\mathcal{R}_{\text{BEG}}$, $\mathcal{R}_{\text{MID}}$, $\mathcal{R}_{\text{END}}$, $\mathcal{R}$
1  if $c(\mathcal{R}_{BEG} \cup \mathcal{R}_{MID} \cup \mathcal{R}_{END} \cup \mathcal{R}) < c(\mathcal{R}_{BEST})$ then
2      if $|\mathcal{R}_{BEG}| = 0$ then
3          $\mathcal{R}_{\text{BEST}} = \mathcal{R}$
4      else
5          $r_{\text{BEG}} \leftarrow$ Select($\mathcal{R}_{\text{BEG}}$)
6          for $r_{MID} \in \mathcal{R}_{MID}$ do
7              Best-Reconnect($\mathcal{R}_{\text{BEG}} - \{r_{\text{BEG}}\} \cup \{r_{\text{BEG}} \oplus r_{\text{MID}}\}, \mathcal{R}_{\text{MID}}, \mathcal{R}_{\text{END}}, \mathcal{R}$)
8              Best-Reconnect($\mathcal{R}_{\text{BEG}} - \{r_{\text{BEG}}\} \cup \{r_{\text{BEG}} \oplus$ REV($r_{\text{MID}}$)$\}, \mathcal{R}_{\text{MID}}, \mathcal{R}_{\text{END}}, \mathcal{R}$)
9          if $|\mathcal{R}_{BEG}| \neq 1$ or $|\mathcal{R}_{MID}| = 0$ then
10             for $r_{END} \in \mathcal{R}_{END}$ do
11                 Best-Reconnect($\mathcal{R}_{\text{BEG}} - \{r_{\text{BEG}}\}, \mathcal{R}_{\text{MID}}, \mathcal{R}_{\text{END}} - \{r_{\text{END}}\}, \mathcal{R} \cup \{r_{\text{BEG}} \oplus r_{\text{END}}\}$)
```

Algorithm 8: Pattern injection procedure.

```
   Input: The current solution $x$, the number of patterns to inject $N_i$, the number of most
          frequent patterns to consider $N_f$, the extracted patterns $\mathcal{A}$
   Output: A new solution best
1  best $\leftarrow x$
2  $l \leftarrow$ select at random the size of the patterns injected
3  $\mathcal{P} \leftarrow$ select at random $N_i$ patterns among the $N_f$ most frequent patterns of size $l$ in $\mathcal{A}$
4  for $p \in \mathcal{P}$ do
5      if $p$ does not occur in best then
6          $(\mathcal{R}_{\text{BEG}}, \mathcal{R}_{\text{MID}}, \mathcal{R}_{\text{END}}, \mathcal{R}_{\text{INV}}) \leftarrow$ disconnect edges adjacent to the vertices of $p$ in best
7          $x' \leftarrow$ Best-Reconnect($\mathcal{R}_{\text{BEG}}, \mathcal{R}_{\text{MID}} \cup \{p\}, \mathcal{R}_{\text{END}}, \mathcal{R}_{\text{INV}}$)
8      if $c(x') < c(best)$ then
9          best $\leftarrow x'$
10 return best
```

no need to consider them when solving the VRPTW since reversed fragments will probably lead to infeasible solutions (i.e., l.8 is removed). Finally, the incremental evaluation used during the reconnection step has to be adapted to match the VRPTW.

Furthermore, other modifications are required when considering a multi-objective problem. In particular, considering the comparison of two (partial) solutions (l.1 of Algorithm 7 and l.8 of Algorithm 8). When the multi-objective problem is modeled by one (or more) single-objective problem (like in MOEA/D), there is no problem. However, with algorithms that do not use an aggregation of objectives, like MOLS, the solution can be accepted if it dominates or if it is non-dominated regarding the current solution.

2.9 Conclusion

In this chapter, our case study is presented. It is a VRPTW where the total transportation cost and the total waiting time of drivers are simultaneously minimized. The problem has been formally introduced, solutions are represented as permutations, and they can be evaluated with the split algorithm. Moreover, the neighborhood of a solution is generated by the three operators `swap`, `relocate`, and `2opt*`. During neighborhood exploration, an incremental evaluation is performed to be faster. We have also provided an overview of the existing works related to the problem. In the following, we are interested in solving this problem by using knowledge discovery principles. More precisely, by using pattern mining strategies, as the PILS mechanism exploits to solve the capacitated vehicle routing problem in a single-objective format.

Part II

Solution-based Knowledge Discovery into MOEA/D

Chapter 3

Multi-Objective Evolutionary Algorithm based on Decomposition

Contents

3.1 Introduction

This chapter is dedicated to the presentation of the main algorithm used during the first part of the thesis work, that is the Multi-Objective Evolutionary Algorithm based on Decomposition, known as MOEA/D. This algorithm was originally introduced by Zhang and Li [2007]. It is a popular algorithm for solving multi-objective combinatorial optimization problems, and it has been widely studied in the literature [Xu et al., 2020]. MOEA/D is a genetic algorithm, that approximates the Pareto front by decomposing the multi-objective problem to solve into several single-objective subproblems. The framework allows the possibility of solving the subproblems in parallel, which is beneficial in practice. For simplicity purposes, in the following, we only consider that the subproblems are sequentially solved. The Section 3.2 presents MOEA/D and its components. Section 3.3 focuses on variants of MOEA/D in the literature, improving its main components. Section 3.4 presents variants exploiting knowledge during the execution in MOEA/D.

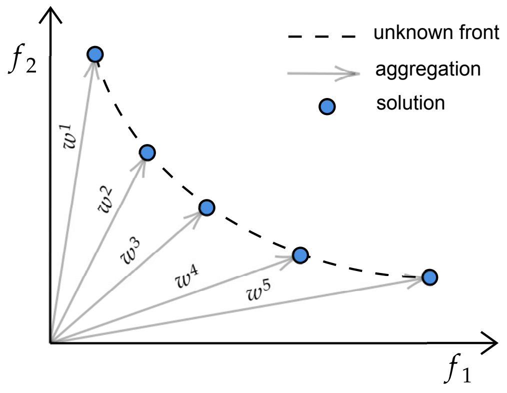


Figure 3.1: Concept of MOEA/D, in a minimization context, where a bi-objective problem is decomposed into five subproblems with weight vectors $w^1, \ldots, w^5$. Each subproblem is associated with its current best solution.

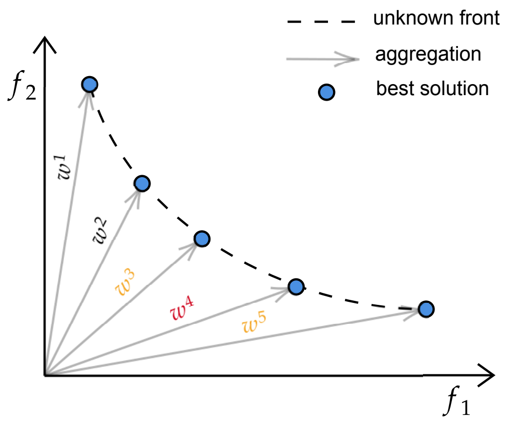


Figure 3.2: Neighborhood (of size $m = 3$) of w^4. It regoups vectors w^4, w^3, and w^5.

Our motivation concerning the choice of this algorithm was its trade-off between single-objective and multi-objective optimization. In MOEA/D, several single-objective problems are solved, allowing the use of mechanisms related to single-objective optimization. Indeed, the exploitation of knowledge from solutions generated during the execution is known to be efficient when dealing with a single-objective problem, but the transition to multi-objective optimization is not obvious. Consequently, relying on MOEA/D facilitated the development of a first knowledge discovery mechanism suited to multi-objective optimization.

3.2 Presentation of the MOEA/D

Algorithm 9 outlines the MOEA/D framework. The algorithm receives M weight vectors to generate the subproblems. Each weight vector is associated with a distinct subproblem, following a decomposition strategy described in Section 1.3.3. We assume that the production of uniformly distributed weight vectors yields a variety of subproblems sufficient to approximate accurately the optimal Pareto front of the multi-objective problem. In particular, with a bi-objective problem, $w^i = (\frac{i-1}{M-1}, \frac{M-i+1}{M-1})$ generates the i-th subproblem. Figure 3.1 illustrates the decomposition concept of MOEA/D in the particular case of a bi-objective problem, in a minimization context, where subproblems are evenly spread.

An initial population P of solution is created with the `Initialization` function. The i-th solution of the population is the current best solution associated with the i-th subproblem. `Initialization` creates random solutions, generates optimized solutions, or reads an existing Pareto front and then associates the best possible solution to each subproblem. The size of the population does not change during the execution and remains constantly equal to M, the number of subproblems. Moreover, inside the population, the order of the solutions is important, since the i-th solution is always associated with the i-th subproblem.

During the execution of MOEA/D, the current best solutions from neighboring subproblems

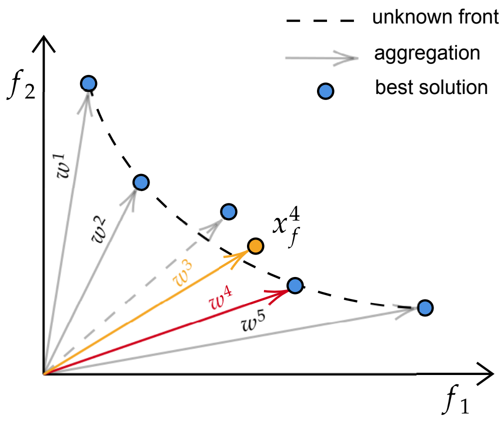


Figure 3.3: Replacement of the current best solution for w^3 by a better solution x_f^4 found during the resolution of the subproblem with w^4. The replacement is possible because w^3 belongs to the neighborhood of w^4.

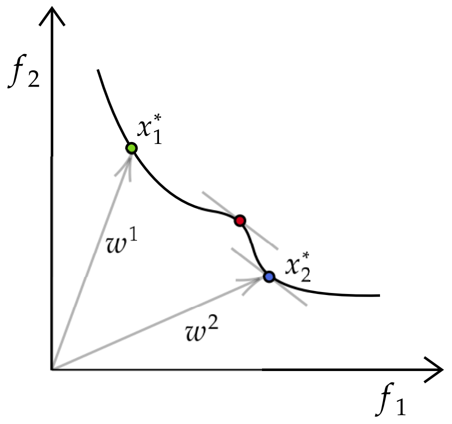


Figure 3.4: Situation where not all the solutions of the optimal Pareto front are optimal solutions of weighted sum problems. Solution x_1^* can be obtained with w^1, but the red solution can not be obtained, since using w^2 returns x_2^*.

can be used to generate a new starting solution for the considered subproblem, thus increasing the exploration capacity of the algorithm. More precisely, the *neighborhood*, of size $m \geq 1$, of a weight vector w^i is defined as the set of its m closest (for the Euclidean distance) weight vectors among $\{w^1, \ldots, w^M\}$. Consequently, the neighborhood $\mathcal{N}_i^m$ of the i-th subproblem consists of the m subproblems defined with a weight vector belonging to the neighborhood of w^i. Figure 3.2 represents the neighborhood associated with a weight vector.

The core of the algorithm consists of iteratively optimizing the subproblems until a stopping criterion is satisfied. The stopping criterion is generally based on time, number of iterations, or number of evaluations. Usually, a random permutation of the subproblems is defined in the beginning so that subproblems are always solved in the same order. When the subproblem i is optimized, two solutions from the population are selected with the `Select` function for the `Crossover` step. The indexes of the chosen solutions belong to $\mathcal{N}_i^m$. After the crossover, in order not to change the population size, only one solution is kept. The solution can be chosen at random or according to its fitness. Note that the crossover is applied with probability p_x. When it does not occur, it is the current best solution to the problem in the population (i.e., x^i) that is kept. The remaining solution undergoes a `Mutation` step with probability p_m, attempting to improve the solution. Finally, the solution is added to another set P'. If the final solution obtained has a better fitness than x^i it is replaced in P with `UpdatePopulation`. Moreover, at most m_r solutions of P associated with neighboring subproblems may also be updated if the final solution obtained is better when evaluated with their weights than their current best. This situation is represented in Figure 3.3, where during the resolution of the subproblem associated with weight w^4, a better solution (x_f^4) for the subproblem associated with weight w^3 is found, thus replacing its current best solution.

Additionally, an external (unbounded) archive $\mathcal{A}^*$ is maintained throughout the execution to track the best non-dominated solutions. After seeing all the subproblems, the `UpdateArchive` procedure updates the archive with the solutions of P', which is emptied after that step. If the stopping criterion is reached $\mathcal{A}^*$ is returned, otherwise another iteration starts.

Algorithm 9: Reference MOEA/D Framework ($R_{MOEA/D}$).

```
Input: $M$ weight vectors $w^1, \dots, w^M$. $m$, the size of the neighborhood of a problem. $p_x$
       (resp. $p_m$) the probability of applying Crossover (resp. Mutation)
Output: The external archive $\mathcal{A}^*$
/* Initialization                                                    */
1  $(\mathcal{A}^*, P') \leftarrow (\varnothing, \varnothing)$
2  $P = \{x^1, \dots, x^M\} \leftarrow$ Initialization()
3  for $i \in \{1, \dots, M\}$ do
4      $\mathcal{N}_i^m \leftarrow$ indexes of the $m$ closest weight vectors to $w^i$
/* Core of the algorithm                                             */
5  while stopping criterion not satisfied do
6      for $i \in \{1, \dots, M\}$ do
7          $(i_1, i_2) \leftarrow$ Select($\mathcal{N}_i$)
8          $x \leftarrow$ Crossover($x^{i_1}, x^{i_2}$)
9          $x \leftarrow$ Mutation($x$)
10         $P' \leftarrow P' \cup \{x\}$
11         UpdatePopulation($P, \mathcal{N}_i^m, x$)
12     UpdateArchive($\mathcal{A}^*, P'$)
13     $P' \leftarrow \varnothing$
14 return $\mathcal{A}^*$
```

3.3 Improvement of MOEA/D Components

Since its creation, MOEA/D has received many interests over the years. A recent survey [Xu et al., 2020] classifies the existing variants of MOEA/D into two categories. On the one hand, the variants that improve the components of MOEA/D, and on the other hand, the variants that are applied to other research fields, like many-objective optimization, being a problem where $M \geq 4$ objective functions are simultaneously optimized.In this section, we present the different components of MOEA/D that can be improved, their related issues, and how they have been enhanced in the last few years. Section 3.3.1 is dedicated to the strategies used to decompose multi-objective problems. An overview of the existing methods to generate weight vectors is presented in Section 3.3.2. Finally, several evolution operators are presented in Section 3.3.3. Note that, although MOEA/D can be used to solve both combinatorial and continuous optimization problems, we mainly focus on the improvements suitable for combinatorial optimization.

3.3.1 Decomposition Methods

The notion of decomposition refers to the concepts introduced in Section 1.3.3. Originally, the decomposition methods used with MOEA/D were the weighted sum, the weighted Tchebycheff, and the penalty-based boundary intersection. The latter can be used when dealing with continuous optimization, and will not be further developed here. Note that, like the Tchebycheff approach, it requires a reference point, as close as possible to the ideal point of the front. Moreover, when the optimal Pareto front is not convex, all Pareto optimal solutions are not optimal solutions to a subproblem obtained with a weighted sum. An example is provided in Figure 3.4, where solution x_1^* is optimal to the subproblem using weight w^1, but the (red) solution in the middle is not optimal using weight vector w^2. Instead, it is the solution x_2^* that is the optimal solution associated with this weight vector. However, it is possible to find the red solution during the resolution, showing the importance of using an external archive that contains intermediate solutions of subproblems.

A strategy to improve a decomposition method concerns the volume reduction of the improvement region associated with a subproblem. According to Wang et al. [2015] the improvement region is determined by the current best solution, the optimal solution, and the contour of the subproblem. More formally, the improvement region of the solution x^i, being the current best solution of subproblem i, is the set of all the objective vectors of solutions better than x^i for the subproblem i. Figures 3.5 (resp. 3.6) show the improvement regions associated with a subproblem defined by a weighted sum (resp. a Tchebycheff approach) using the weight vector w^i. One can notice that the improvement region associated with the problem defined by the Tchebycheff approach is much smaller than the one associated with a subproblem defined by a weighted sum. With a large improvement region, a single solution with a better fitness for one subproblem can lead to the replacement of several old solutions for other subproblems, leading to a loss of diversity inside the population. This issue has been addressed with many different strategies. Wang et al. [2015] control the improvement region by constraining more the subproblems with a dedicated parameter. Cai et al. [2017] proposed a constrained decomposition with grids, where the objective space is divided into grids, and each subproblem is associated with one grid. Ma et al. [2017] propose to extend the Tchebycheff decomposition strategy with a L_p norm constraint on the direction vector. Two other strategies introduced by Jiang et al. [2017b] generalize the Tchebycheff approach. The multiplicative scalarizing function and the penalty-based scalarizing function methods, depend on a parameter, directly impacting the size of the improvement region of a subproblem. When the parameter value is set to 0, then both strategies turn back to the original Tchebycheff approach.

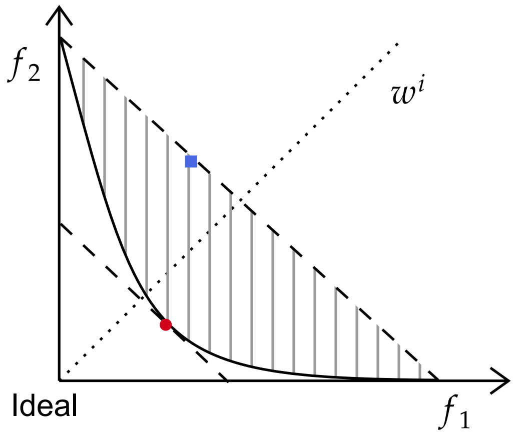


Figure 3.5: Improvement region associated with a subproblem defined with a weighted sum. The blue square is the current best solution. The red dot is the optimal solution.

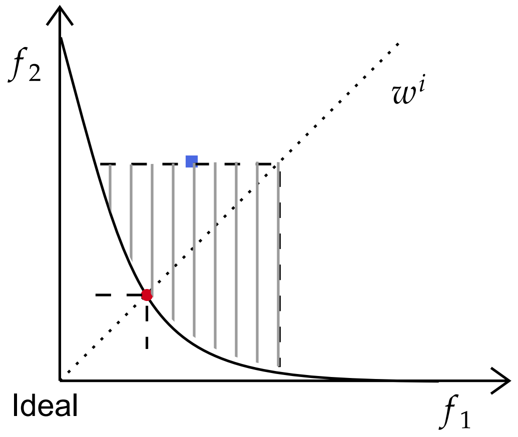


Figure 3.6: Improvement region associated with a subproblem defined with a Tchebycheff approach. The blue square is the current best solution. The red dot is the optimal solution.

It is known that depending on the shape of the final Pareto front, all decomposition strategies are not equivalent. Indeed, while the Tchebycheff approach is suited to convex and non-convex Pareto fronts, it converges slower than the weighted sum approach, which is not adapted to non-convex fronts. The weighted mixture-style decomposition of Zheng et al. [2018] combines the advantages of these two approaches, allowing MOEA/D to obtain more solutions. Ishibuchi et al. [2010] used weighted sum and Tchebycheff approaches for fitness assessment.

Finally, Liu et al. [2013] proposes to base the decomposition strategy on the decomposition of the objective space. The multi-objective problem is decomposed into several simpler multi-objective problems, with the same objective functions, but each one is restricted to a smaller part of the objective space. It can be achieved by constraining the subproblem or turning one objective function into a constraint. In particular, this strategy called *multiple-to-multiple*, is efficient when dealing with many-objective optimization.

3.3.2 Weight Vector Generation Methods

Most of the decomposition methods described above require the use of weight vectors to guide the search. In the first version of MOEA/D, the weight vectors were fixed during the execution. However, when the Pareto front is irregular or complex, it may not be a suitable strategy to use the same weight vectors all along the execution. Li et al. [2015] propose to generate new random weight vectors when no improvements are found after several iterations. A hierarchical structure between subproblems is developed by Xu et al. [2017]. The subproblems are divided into two layers. The higher-level layer contains the main search directions, while the lower-level layer can be dynamically adjusted according to the results obtained with the higher-level layer. Indeed, the objective of the lower-level layer is to refine the search between two subproblems from the higher-level layer.

When the structure of the Pareto front is not known, the best strategy is to consider a uniform distribution of weight vectors to maximize the diversity of the solutions obtained. With two dimensions, the generation of uniformly spread weight vectors is straightforward, however in higher dimensions mathematical strategies are required. The *good lattice point* strategy [Zaremba, 1966] can be applied to generate a set of uniformly distributed weight vectors [Tan et al., 2011]. Zhang et al. [2015] developed a uniform design method to initialize the population and generate the weight vectors. This is done by first generating a uniform matrix with the good lattice point strategy, and then transforming the matrix into a set of vectors over the unit simplex (i.e., vectors whose sum of coordinates is equal to 1).

When a decision maker provides any feedback during the search process, it can be used as an indicator to focus on a specific part of the objective space, called *region of interest* of the decision maker. Particularly, Pilat and Neruda [2015] incorporates the binary preferences of the decision maker to the individuals, to adjust the weight vectors during the execution. Ma et al. [2016] biased the weight vectors to focus on the region of interest of the decision maker. Indeed, when a subproblem returns solutions that are too far from the region of interest, the subproblem is discarded and another one is generated.

3.3.3 Evolution Operators

The evolution operators concern the mating selection (i.e., at l.7 of Algorithm 9, the selection of the solutions which undergo the crossover) of two solutions of the population, the replacement strategy of some solutions in the population when a new solution is discovered (l.11 of Algorithm 9), and the reproduction operators (i.e., crossover and mutation steps in general).

It is important that the mating selection does not always favor the same solutions to bring diversity to the next generation of solutions. Li and Zhang [2008] proposed to allow the mating of a solution with any other population member with some probability, instead of mating only with solutions from neighbor subproblems. Chiang and Lai [2011] decide to restrict the mating population to individuals associated with an *unresolved* subproblem. A subproblem is said *resolved* when its current best solution is not improved for a minimum number of generations. Note that, a new better solution may be found for a subproblem after many iterations, thus even a subproblem considered as resolved may become again unresolved during the execution.

The replacement strategy should not be too aggressive to avoid the loss of diversity in the population. In particular, we would like to avoid the same solution appearing many times in the populations (i.e., one solution that is the current best of several distinct subproblems). Wang et al. [2014] developed a global replacement strategy. In particular, when a new solution x^i_{new} is found during the resolution of the subproblem i, x^i_{new} is first associated with the best possible subproblem j. Then the replacement set considered is the neighborhood of the subproblem j, and a maximum number of replacements is performed like in the reference MOEA/D. One can see the replacement process as a mutual matching between solutions and subproblems. Indeed, each solution should be associated with the best subproblem, and inversely. However, the solution does not explicitly express a preference for subproblems in MOEA/D. Li et al. [2013b] proposed a simple and effective stable matching model to coordinate the association process. Each subproblem ranks all the solutions including the current best and offspring according to its aggregation function favoring the solutions with better function values. Additionally, each solution ranks all subproblems by considering the distance between its objective vector and the direction vectors of the subproblems and prefers the subproblems whose direction vectors are close to it. Finally, the model pairs each subproblem with one solution.

It is also possible to mix improvements of mating selection and replacement strategy by using distinct neighborhoods for each process. Ishibuchi et al. [2006] recommend using a large matching neighborhood (e.g., the entire population), and a small replacement neighborhood for better results.

Concerning the reproduction operators, any crossover or mutation step defined in the literature can be used, regarding the representation of the solution. Some possibilities are described in Section 1.2.2. We recall that the crossover step is similar to an exploration (or perturbation) step, where new (unseen) solutions are generated. The mutation often represents an exploitation step, where solutions are improved. Many nature-based algorithms, such as particle swarm optimization, simulated annealing, and ant colony optimization, can be combined with MOEA/D to further improve the algorithm's performance. For example, Ke et al. [2013] hybridized MOEA/D with an ant colony optimization framework. Here an ant is considered as an evolution unit, aiming to improve its best associated solution. All the ants (there are as many ants as subproblems) are divided into groups by clustering their corresponding weight vectors. Each group is designed to approximate a small part of the Pareto front. All the groups share a common solution pool, where solutions are updated like in the original MOEA/D. Each ant constructs a new solution with the pheromone matrix about itself and its neighbors (i.e., ants of the same group).

3.4 Knowledge Exploitation in MOEA/D

The components of MOEA/D described above can be further improved by using self-adaptive mechanisms. Indeed, during the search much information can be collected about the performance of the operators used, the shape of the Pareto front, and, more generally, about the objective space itself. This information is useful for adapting the strategy adopted for each component of MOEA/D, making it more suitable to solve complex problems. The remainder of this section is dedicated to the existing adaptive mechanisms related to each component of MOEA/D. To be more precise, Section 3.4.1 presents different approaches to make the decomposition strategies adaptive. Section 3.4.2 shows how weight vectors can be dynamically adjusted during the execution. Finally, a few adaptive evolution operators are described in Section 3.4.3.

3.4.1 Adaptive Decomposition Strategy

Using an adaptive decomposition strategy can help to accelerate the convergence of the algorithm towards the optimal Pareto front, and to obtain more widely spread solutions. An automatic selection strategy has been developed by Ishibuchi et al. [2009] to decide if a weighted sum or a Tchebycheff approach is better adapted to the current weight vector. Indeed, the Tchebycheff approach is more adapted to approximate non-convex regions of the Pareto front. Since it is hard to know if a region of the Pareto front is convex or not, the strategy used considers that the region is non-convex when the current solution is the best solution for many different neighboring subproblems (with its own weighted sum).

More generally, the searching ability of the family of the L_p scalarizing methods (when $p = 1$, the method is a weighted sum, when $p = \infty$, the method is a Tchebycheff approach), has been investigated in Wang et al. [2016]. The value of p impacts the convergence and the diversity of the solutions obtained for a specific problem. More precisely, when p increases, the probability of finding a better solution decreases, allowing the generation of more diverse solutions. Besides, the impact of p decreases with the proximity of a solution to the optimal Pareto front. The better p value is obtained when the optimal solution for that p is closer to the weight vector compared with other p values.

Finally, a *learning-to-decomposition* paradigm is developed by Wu et al. [2018]. The paradigm is composed of two modules, one is dedicated to learning, and the other to optimization. The objective of the learning module is to learn the characteristics of the estimated Pareto front, by using the non-dominated solutions obtained with a Gaussian process regression. The optimization module, based on MOEA/D, adapts the decomposition method according to the model returned by the learning module. Such a research topic shows the benefit of hybridizing MOEA/D with machine learning algorithms (and more generally the interest in hybridizing learning and optimization processes) to make better use of the information generated during the evolution.

3.4.2 Dynamic Adjustment of Weight Vectors

Using uniformly distributed weight vectors may not be enough to solve complex problems, or to approximate Pareto-fronts with complex shapes (e.g., discontinuous, non-convex). The goal of adjusting weight vectors during the search is to obtain more uniformly spread solutions along the optimal Pareto front.

Siwei et al. [2011] proposed a Pareto-adaptive weight vectors strategy taking into account the geometrical characteristics of the Pareto front. The generation of the weight vectors employs Mixture Uniform Design (MUD), to evenly spread vectors in a given region. These weight vectors intersect the optimal Pareto front in k points, and the weight vectors are adjusted to maximize the hypervolume of the k points obtained.

When the optimal Pareto front is irregular, fixed weight vectors may have difficulties to guide the algorithm to obtain distributed solutions along the Pareto front. In particular, in the discontinuous parts of the Pareto front, several subproblems have the same optimal solution. In the extreme regions of a Pareto front, many objective vectors are distributed in a narrow region of the objective space. Qi et al. [2014] developed an adaptive weight adjustment strategy, that identifies sparse regions (avoiding discontinuous parts), adds new subproblems in sparse regions, and removes redundant subproblems in crowded regions.

In real-world problems, different subproblems are likely to have different search complexity. Harada et al. [2017] developed a strategy for detecting hard problems, and then these problems are divided into several new subproblems to allocate more computational resources to the directions of the subproblems with search difficulty.

3.4.3 Adaptive Evolution Operators

Three evolutionary operators are used in MOEA/D. The mating selection, the replacement mechanism, and reproduction operators. The mating selection and the replacement mechanism use the neighborhood associated with the subproblem being solved. The size of the neighborhood is a parameter fixed at the beginning of the algorithm, however, it plays an important role in controlling the trade-off between intensification (small neighborhood) and diversification (large neighborhood). Zhao et al. [2012] proposed to adapt the neighborhood size of each subproblem (each subproblem may have a different neighborhood size) during the execution of the algorithm. The new size of the neighborhood is selected among a pool of sizes, and the probability of selecting one size depends on its efficacy in past generations (i.e., the proportion of improving solutions kept for the next generation). Probabilities are updated at the end of a learning period, reflecting a number of successive generations.

The reproduction operators themselves are, generally, specific to the problem solved. However, if several operators are available, one can use a high-level integration strategy (see Section 1.4.3) to select the best operators to apply. For example, Li et al. [2013a] proposed a bandit-based adaptive operator selection, using fitness improvement rates to track the dynamics of the search.

3.5 Discussion

This chapter aimed to present the components related to MOEA/D, and how these have been improved in the literature to overcome issues of the original design. We mainly focused on improvements that can be used in a combinatorial context, although there exists a lot of other works in the continuous field. Indeed, when solving continuous problems, it is often possible to exploit the properties of the decision space. For example, it is possible to exploit the regularity of continuous fronts, and their link with the decision space, to learn a distribution model of a current population. A Baldwinian learning operator can use this to construct a candidate descent direction based on the learned model and the history of individuals [Ma et al., 2014b]. On the other hand, it is also possible to explore the decision space by using opposition-based learning principles Ma et al. [2014a] (where a solution and its *opposite* are simultaneously explored to provide interesting knowledge).

However, such strategies can not be easily used in combinatorial optimization, since the decision space can be chaotic due to the constraints of the problems. Our intention through this chapter was to highlight the diversity of the works performed to improve MOEA/D. More importantly, the framework has been successfully used with learning mechanisms. Although we only use the original MOEA/D in the remainder of the thesis, the strategies developed there can lead to further improvements in our work. Overall, the popularity of the framework and the interest it received over the years, make MOEA/D a good candidate for our thesis work.

The next chapter presents the first version of the hybridization between MOEA/D and our knowledge discovery mechanism developed from PILS. In particular, it uses the single-objective decomposition principle to mimic the behavior of PILS in a multi-objective context.

Chapter 4

Knowledge Groups designed for MOEA/D

Contents

This chapter's contributions are linked to the following publications:

- Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, ME. 2022. Enhancing MOEA/D with learning: application to routing problems with time windows. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). https://doi.org/10.1145/3520304.3528909
- Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, ME. (2023). New Neighborhood Strategies for the Bi-objective Vehicle Routing Problem with Time Windows. In: Di Gaspero, L., Festa, P., Nakib, A., Pavone, M. (eds) Metaheuristics. MIC 2022. Lecture Notes in Computer Science, vol 13838. Springer, Cham. https://doi.org/10.1007/978-3-031-26504-4_4.
- Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, M. E. (2024, June). Investigation of the Benefit of Extracting Patterns from Local Optima to Solve a Bi-objective VRPTW. In Metaheuristics International Conference (pp. 62-77). Cham: Springer Nature Switzerland. https://www.doi.org/10.1007/978-3-031-62912-9_7

4.1 Introduction

This chapter presents the first proposition of hybridization between MOEA/D and our knowledge discovery mechanism. Different experiments are performed to highlight the performances of our hybridization (in terms of quality and speed-up).

For our knowledge discovery mechanism, the idea is to exploit the structure of close solutions in the objective space. However, it can only be interesting if close solutions in the objective space share common structures. Thus, we start by analyzing how the phenotypic proximity impacts the genotypic proximity for our problem in Section 4.2. In this section, we define metrics to measure the phenotypic distance and the genotypic distance, and we analyze the objective space of Solomon's instances. The results obtained show that, in most instances, solutions with close objective vectors share common structures, which can be exploited by the knowledge discovery mechanism.

Consequently, the knowledge discovery mechanism proposed exploits the structure of solutions by grouping them according to their location in the objective space. In Section 4.3, the concept of knowledge groups is introduced to store the different patterns extracted from solutions. As a first model, the knowledge groups are associated with the subproblems defined in MOEA/D, leading to a first promising hybridization (see Legrand et al. [2022a]).

In order to improve the results obtained, we decided to focus on the creation of a local search adapted to our bi-objective problem. With the use of a local search inside our algorithm, it is possible to produce solutions of better quality, which will, hopefully, lead to the learning of better patterns. For that local search, we use common neighborhood operators, but we define in Section 4.4, a new neighborhood exploration strategy, called *first-best*. Moreover, a granular search, which exploits both objectives optimized, is used additionally to reduce the size of the neighborhood explored. The results presented in Legrand et al. [2022b] show the effectiveness of our strategies.

Finally, Section 4.5 investigates the benefit of learning from local optima produced by the local search designed in the former section. Indeed, in single-objective optimization, local optima are known to contain more relevant structures and are often preferred when structures of solutions are learned. However, using local optima only may not be suited to a multi-objective context. The results presented in Legrand et al. [2024] show that learning from local optima contributes to obtaining solutions of better quality.

4.2 Relation between Genotype and Phenotype

In this section, we provide insight into the relation between phenotype and genotype of solutions of the bVRPTW. We recall that the phenotype of a solution refers to its objective vector and the genotype refers to its structure. Thus, our objective is to know if solutions with a close phenotype (i.e., solutions that are close in the objective space) also have a close genotype (i.e., share a similar structure). First, we present in Section 4.2.1 the metrics involved in measuring the phenotypic and genotypic distances. Our experimental protocol follows in Section 4.2.2, and the results are presented and discussed in Section 4.2.3.

4.2.1 Phenotypic and Genotypic Distances

For the phenotypic distance d_P it is possible to consider any metric defined on vectors of $\mathbb{R}^n$. We decided to consider the Euclidean distance, which is the most natural metric in the objective space. Thus, the phenotypic distance between two solutions x and y is measured with the Euclidean distance between their corresponding objective vectors $F(x)$ and $F(y)$.

The genotype of a solution to the bVRPTW is the list of the arcs involved in the routes. To measure the structural similarity between two solutions, we can consider any metric used to measure the similarity between two graphs. The Hamming distance, which counts the number of deletions and insertions of arcs required to transform one solution into another, is the most simple and intuitive metric to consider. More formally, to compute the Hamming distance between two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, we generate the graph of the intersection, having the same vertices V and the edges $E_1 \cap E_2$. Then, the Hamming distance between G_1 and G_2 is expressed as the following quantity:

$$d_H(G_1, G_2) = |E_1| + |E_2| - 2 \times |E_1 \cap E_2| \tag{4.1}$$

Indeed, to transform G_1 into G_2 it requires $|E_1| - |E_1 \cap E_2|$ deletions and $|E_2| - |E_1 \cap E_2|$ insertions. Consequently, the genotypic distance d_G between two solutions x and y is measured with the Hamming distance between their corresponding graph representation. In addition to the Hamming distance, we define a *similarity* value S between two graphs G_1 and G_2, with the same notations as above, which is expressed as:

$$S(G_1, G_2) = \frac{|E_1 \cap E_2|}{|E_1 \cap E_2| + d_H(G_1, G_2)} \tag{4.2}$$

Figure 4.1 provides an example of computation of the Hamming distance between two routing solutions and the similarity value is given.

4.2.2 Experimental Protocol

For this analysis, we only focus on Solomon's benchmark (see Section 2.3.1). The first issue is to generate a good approximation of the objective space of an instance. To that aim, we use a multi-objective local search, as described in Chapter 6, because such algorithms explore more largely the objective space. During the execution, we track every solution encountered, and at the end, the files containing the objectives and the structures of the solutions are written. Note that, because of the large number of solutions encountered during the execution (sometimes more than $5 \cdot 10^5$), we have to limit the number of solutions written by uniform sampling at most $5 \cdot 10^4$ solutions. We

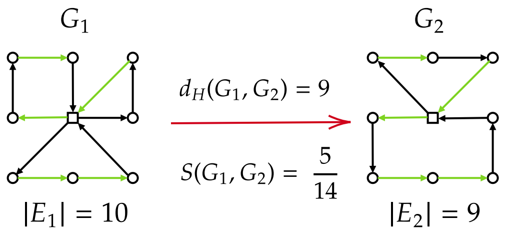


Figure 4.1: Graph representation of two solutions. The arcs in common are colored green. The Hamming distance between these solutions is 9, their similarity is about 36%.

execute five runs on each Solomon's instance, each run being executed with a different initial front. Finally, for each instance, at least $25 \cdot 10^3$ and at most $25 \cdot 10^4$ solutions are available to represent the associated objective space, which is enough to obtain a good representation of the solutions that can be found when solving an instance. Moreover, dealing with more solutions will result in computational issues for our study.

Our motivation is to highlight the presence of structural similarities between solutions with a close phenotype. Consequently, we consider a parameter R, which controls the size of the neighborhood of a solution in the objective space. Basically, the neighborhood of a solution contains its R nearest neighbors considering the phenotypic distance. Then, for different values of R, we can evaluate the average structural similarity between a solution and its neighborhood. We could do this for every solution of the objective space, however, the computational overhead would be too high, since it requires a quadratic number of operations in the number of solutions to generate all the neighborhoods. Hence, we decided to sample uniformly 2500 solutions of the objective space, which represents between 1% and 10% of the initial objective space.

It is possible to compute the neighborhood of each solution for different values of R (here $R \in \{1, 2, 5, 10, 25, 50, 75, 100\}$) and to evaluate the average genotypic similarity between a solution and its neighborhood, in a reasonable amount of time. Nevertheless, preliminary results suggested that the presence of genotypic similarities may be impacted by the position of the solution in the objective space. Indeed, converged solutions, close to the optimal Pareto front, may more likely share structural similarities than random solutions with high objective values. By considering all the solutions together, this phenomenon can not be highlighted. As a consequence, we decided to compute the Pareto ranks of all the 2500 remaining solutions and to homogeneously stratify the objective space according to the ranks of the solutions. To that aim, we define a threshold (here set to 20%) representing the proportion of solutions that the region should contain. Since a subfront of solutions is associated with each rank, we can simply aggregate them, by increasing rank value, until the desired proportion of solution is reached. Moreover, we allow a slight overlapping of four ranks between each region, i.e., if a region stops at rank k, then the next region starts at rank $k - 1$, but the proportion of solutions in the new region is considered only after rank k so that the objective space is always divided into exactly five regions. Finally, for each region and value of R, we can compute the average genotypic similarity for each solution and the mean over all the solutions of each region is returned, giving the average similarity of a solution in the region for the value of R considered. We perform that evaluation over 10 runs for each instance of Solomon's benchmark. The results obtained are presented and discussed in the next section.

4.2.3 Experimental Results

First, we present the average similarity of solutions according to the instance category. An overview of the results are available in Figure 4.2. Additional tables are provided in Appendix A with the corresponding values. One can remark that, for every category of instance, and no matter the region of the objective space, the average similarity tends to decrease when the size of the neighborhood increases. In particular, it indicates that considering one solution, the furthest solutions to it tend to have less structural similarity. Moreover, it shows that the maximum genotypic similarity is achieved for close solutions, no matter the region of the objective space. Another important point concerns the difference of similarity between solutions to instances of category 1 (with tight time windows) and instances of category 2 (with wide time windows). In instances of category 1 (C1, R1, RC1) the average similarity is about 40% between a solution and its closest neighbor (i.e., about 40 arcs are in common when the size of the neighborhood is 1), and the highest similarity is generally obtained for solutions that belong to the "r0" region (i.e., the region of the objective space that is the closest to the optimal Pareto front, and which contains the most interesting solutions for both objectives). In particular, it shows that the Pareto front tends to contain solutions that share a lot of characteristics. However, it is almost the opposite in instances of category 2 (C2, R2, RC2). For these instances, we generally have about 25% of average similarity between close solutions in the region "r0", while the similarity increases in region "r4" (i.e., the region of the objective space that contains the less interesting solutions, with many random solutions). In particular, it means that for these instances, the best solutions often share few characteristics, but many bad solutions share a lot of structural components.

Now, we have decided to focus on a subset of particular instances. The instances R101, R201, C106, and RC108 represent typical situations that can be encountered when dealing with multi-objective combinatorial optimization. An approximation of their associated objective space is provided in Figures 4.3 to 4.6. The objective space of R101 contains many non-dominated solutions (more than 1000), in R201 the objective space contains few non-dominated solutions (less than 250), the C106 contains only one non-dominated solution (minimizing both the waiting time and the transportation cost), and the RC108 can be considered as a single-objective instance since all solutions found in the objective space have the same value (0) for the total waiting time.

Figure 4.7 shows the average similarity between solutions for the four instances studied. Detailed values are available in Appendix A. The remarks made above still hold. More precisely, for R101, the maximum similarity is about 55% in the region "r0" with close solutions, and the similarity decreases with the size of the neighborhood and also with further regions, indicating that low-quality solutions share fewer structures in common. It is the contrary with R201, where high-quality solutions share around 30% of arcs, and low-quality solutions share almost 70% of arcs. Such information is useful to guide the algorithm towards more promising regions. In particular, the results obtained for R201 show that we should avoid solutions that share a lot of arcs, and consequently, classical neighborhoods should be discarded in favor of destroy and repair operators. In addition, it provides an example where learning from random solutions does not help the search process, and in this situation, it is more interesting to wait sometime until solutions with better quality are found. Their respective objective space may also explain the difference in the evolution of the similarity between R101 and R201. Indeed, R101 contains much more non-dominated solutions than R201 and the region of the objective space containing low-quality solutions is more sparse in R101 than R201. Concerning instance C106, it looks like an intermediate situation between R101 and R201. In this case, the average similarity remains almost stable through the different regions of the objective space but decreases with the number of closest solutions considered. Finally, with

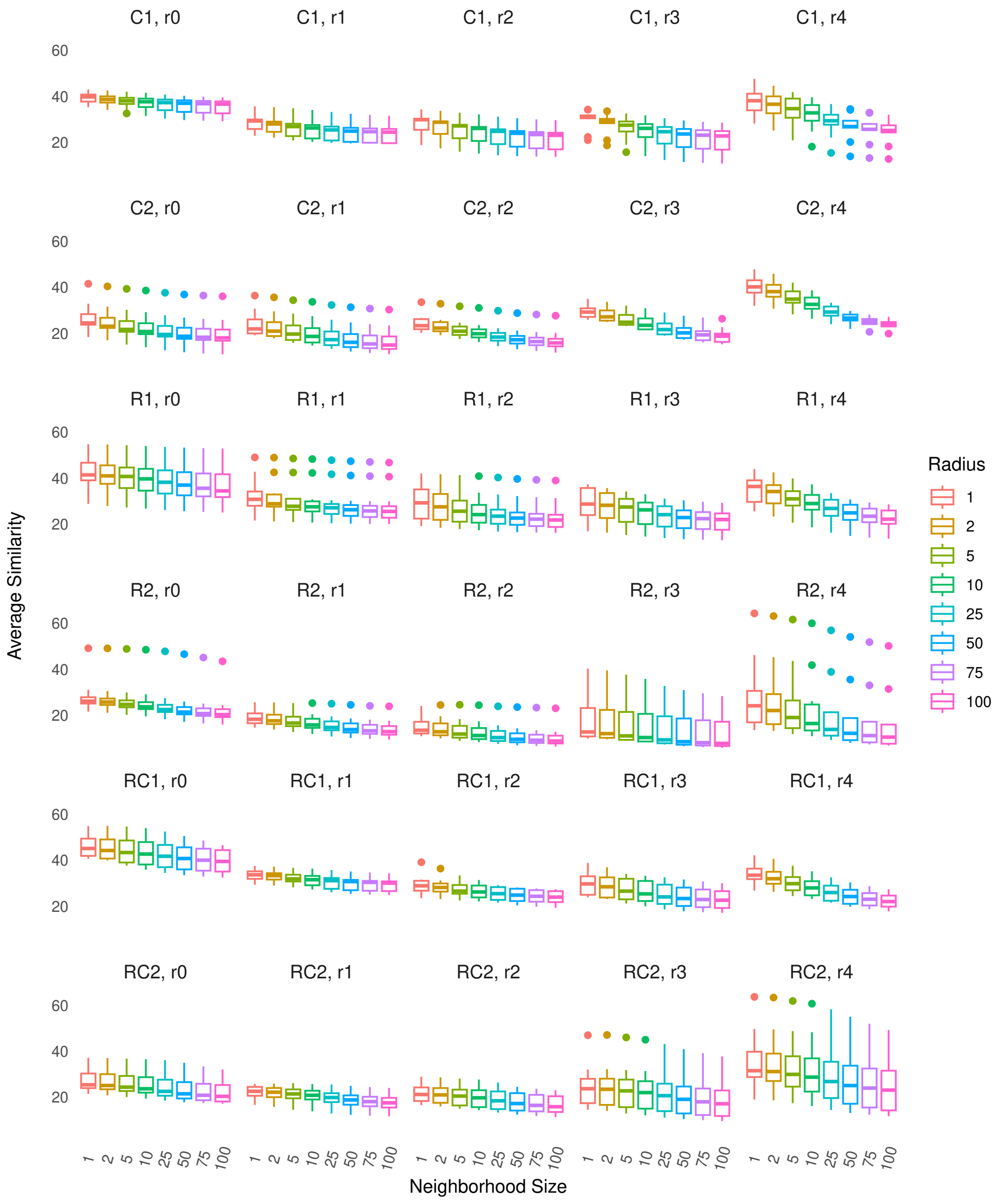


Figure 4.2: Boxplots representing the average similarity between a solution and its neighborhood for different neighborhood sizes (represented by the radius variable), and for different regions of the objective space. Each row represents a different category of instance. Each column represents a different region of the objective space.

instance RC108 we observe a high similarity between high-quality solutions and a low similarity between low-quality solutions. Moreover, this time the similarity highly drops between solutions of the region “r0” and solutions of the region “r1”. As a consequence, in a single-objective context, we find similar conclusions as those highlighted by Arnold et al. [2021], when the arcs in common for solutions with different quality levels are studied.

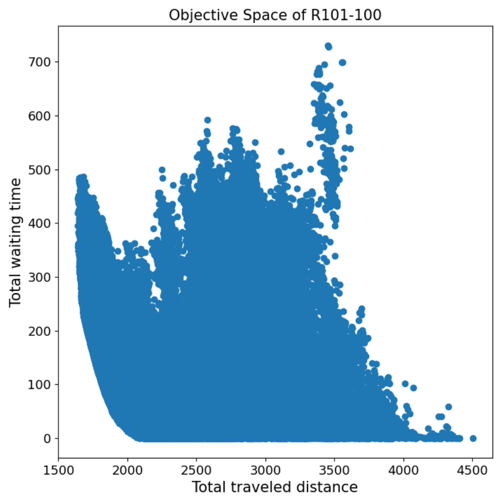


Figure 4.3: Approximation of the objective space of R101.

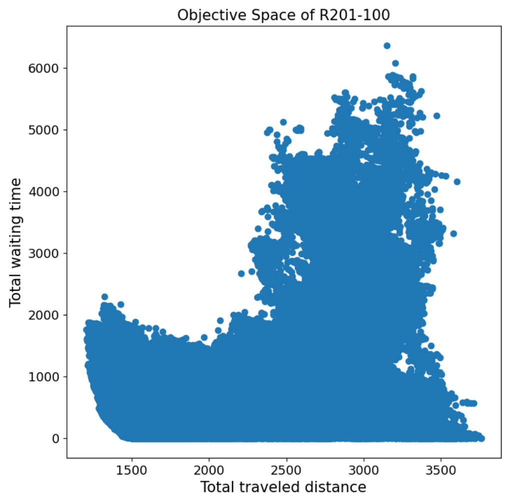


Figure 4.4: Approximation of the objective space of R201.

To conclude this section, our study has highlighted the presence of common structures between close solutions in the objective space, making relevant the notion of knowledge groups introduced in the following section. Moreover, even if the average similarity found in some instances is not very high, we recall that we considered only a subset of 2500 solutions in the objective space, over 10 different runs. Despite the size of the objective space, the variation over the different runs is not significant, meaning that our observations are not highly impacted by the sample considered, and thus that most solutions of the objective space share these properties. In practice, the similarity between solutions should be higher due to the density of the objective space. Moreover, this study encourages us to learn from structures of solutions in a multi-objective context.

4.3 Knowledge Groups

According to the results presented in the former section, we consider that the following assumption is valid: close solutions in the objective space share structural similarities. That is why, gathering the knowledge from neighboring solutions in the objective space is relevant. Hence, we introduce the notion of *knowledge groups*, which is the base component for learning from structures of solutions in a multi-objective context. Indeed, we recall that due to the optimization of conflicting objectives, the structure of a solution that is good for one objective, may not be relevant for another objective.

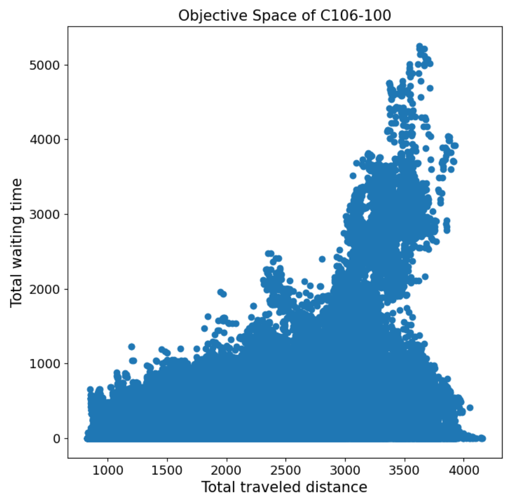


Figure 4.5: Approximation of the objective space of C106.

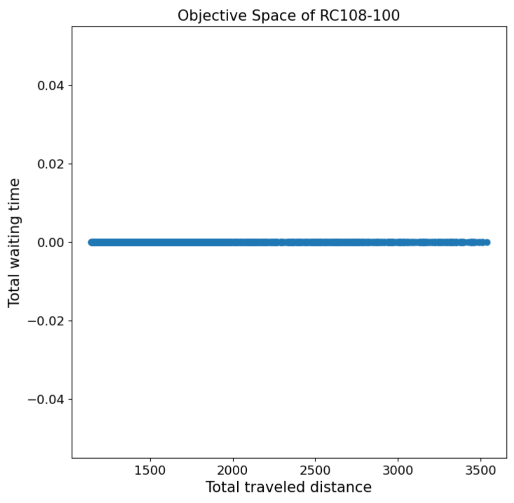


Figure 4.6: Approximation of the objective space of RC108.

For example, considering instance R201 from Solomon's benchmark, Figure 4.8 (resp. Figure 4.9) illustrates a solution obtained when minimizing the total transportation cost (resp. the total waiting time). These figures highlight the structural differences between two solutions optimizing conflicting objectives.

The remainder of this section is organized as follows. In Section 4.3.1 the concept of knowledge groups is introduced for MOEA/D, and the integration of the knowledge discovery mechanism is presented in Section 4.3.2. Then, the proposed algorithm is evaluated following the protocol described in Section 4.3.3. We show and discuss the results obtained in Section 4.3.5.

4.3.1 Concept

Since MOEA/D is itself based on mechanisms that decompose the objective space, it was quite natural to create knowledge groups that depend on the decomposition performed during MOEA/D. The interest in using a knowledge group is to gather in the same object the most frequent structures found when exploring a specific region of the objective space. A single knowledge group plays the same role as the structure defined with PILS to track the frequent patterns found (see Section 2.8).

Let us consider M subproblems that would be defined at the beginning of MOEA/D. In theory, each subproblem focuses on a specific region of the objective space, with minor overlapping. Moreover, we recall that $\mathcal{N}_i^m$ denotes the neighborhood of the subproblem i. Then, we decide to associate with the i-th subproblem a knowledge group $\mathcal{G}_i$, that gathers the structures of the solutions found when solving the subproblem i. Since the subproblem i may receive solutions from subproblems in $\mathcal{N}_i^m$, we decided to add also in $\mathcal{G}_i$ the solutions found when optimizing one neighbor problem of the i-th subproblem. Note that with this construction, there are as many groups as subproblems. Figure 4.10 shows an example of a knowledge group associated with its subproblem.

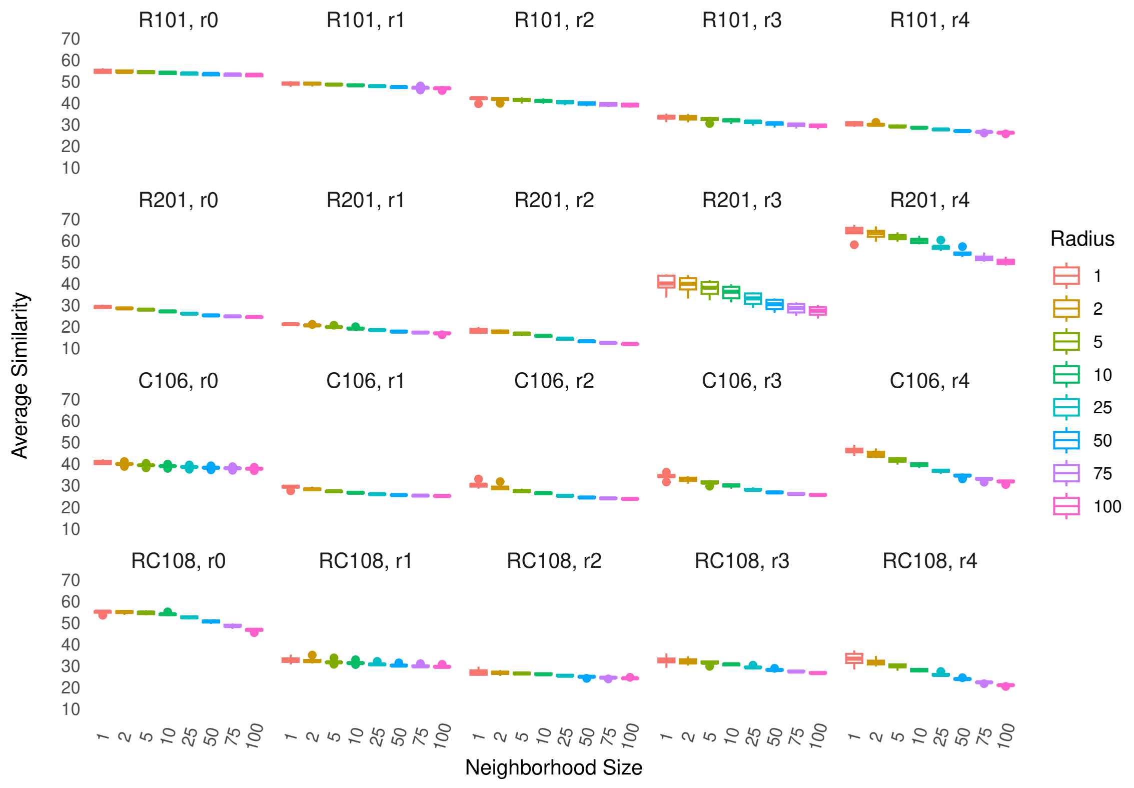


Figure 4.7: Boxplots representing the average similarity between a solution and its neighborhood for different neighborhood sizes, and for different regions of the objective space. Each row represents a different instance. Each column represents a different region of the objective space.

In this situation, the neighborhood of a subproblem is limited to $m = 3$. Concerning the update of the groups, $\mathcal{G}_4$ is updated with patterns from all solutions found when optimizing a subproblem associated with w_3, w_4, and w_5. Inversely, any solution found when optimizing the subproblem associated with w_4 contributes to knowledge groups $\mathcal{G}_3$, $\mathcal{G}_4$, and $\mathcal{G}_5$ as illustrated by Figure 4.11. Indeed, the subproblem associated with w_4 belongs to the neighborhood of subproblems associated with w_3, w_4, and w_5.

The knowledge extracted from solutions is the same as described in Section 2.8. We recall that a parameter s_p controls the maximum size of patterns extracted from a solution, and when a pattern is added to a knowledge group, its frequency is incremented by one.

The injection remains mainly similar to the one presented in Section 2.8, the major modification concerns the choice of the patterns injected. Suppose subproblem i is being solved and the current solution x undergoes the injection procedure. Since all knowledge groups contain patterns that can be injected in x, it is possible to choose the N_i patterns to inject from different groups. In fact, it does not seem relevant to consider a subset of patterns from one group, then continue with

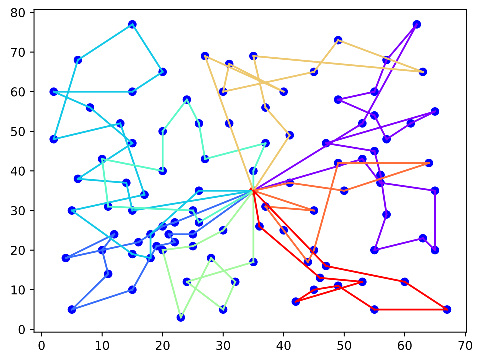


Figure 4.8: Example of solution minimizing the total transportation cost (instance R201).

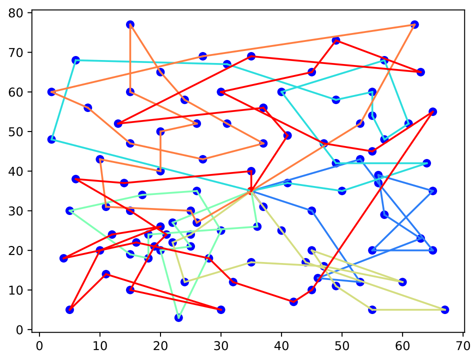


Figure 4.9: Example of solution minimizing the total waiting time (instance R201).

patterns from another group, and so forth. Indeed, each knowledge group is specialized in a specific region of the objective space, and by picking patterns from different groups we may unintentionally introduce conflicting patterns, resulting in poor results during the injection. Consequently, we decided to consider always the patterns to inject from the same group, however, the question of selecting which group remains. We investigate two different strategies. The first strategy selects the knowledge group associated with the current subproblem, i.e. $\mathcal{G}_i$, to favor intensification by exploring the same region of the objective space. This strategy is denoted s_I^i. The second strategy performs a selection uniformly at random among all the knowledge groups to favor structures from another region of the objective space, resulting in more diversified solutions explored. This strategy is denoted s_I^d. Once the group is selected, the size of the patterns injected is chosen uniformly at random, and N_i patterns of that size are selected among the N_f most frequent patterns of the same size in the knowledge group. We decide not to keep only the most frequent patterns, and not to bias the selection towards always the same structures. The injection of a pattern into a solution remains unchanged compared to the procedure in PILS. Figure 4.12 shows how the extraction and injection steps interact together with the knowledge groups.

4.3.2 Integration into MOEA/D

The reference MOEA/D considered is the one presented through Algorithm 9. The modifications related to the hybridization appear in red. MOEA/D is initialized, with a uniform set of weight vectors, and a population of randomly generated solutions (`Initialization`). It was important not to bias the knowledge discovery mechanism developed by considering additional information to solve routing problems in the different operators used (selection, crossover, mutation). In particular, the `Crossover` is the PMX (see Section 1.2.2) applied with probability p_x and it is a generic operator that does not exploit knowledge in routing problems. The selection provides two neighboring subproblems, giving two solutions for the crossover. Then, the best solution of the offspring undergoes a permutation swap mutation, with probability p_m. Only one solution of the offspring is considered to keep the size of the new population constant. The population is updated as usual, by replacing at most m_r solutions in the subproblem neighborhood. The external archive maintaining

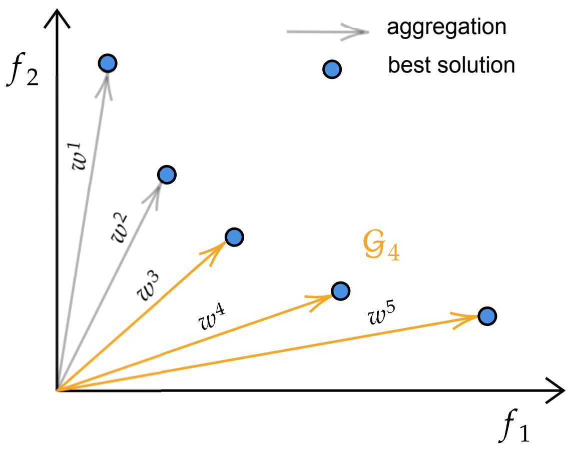


Figure 4.10: Example of knowledge group ($\mathcal{G}_4$) associated with its subproblem. The neighboring subproblems (in orange) contribute to $\mathcal{G}_4$.

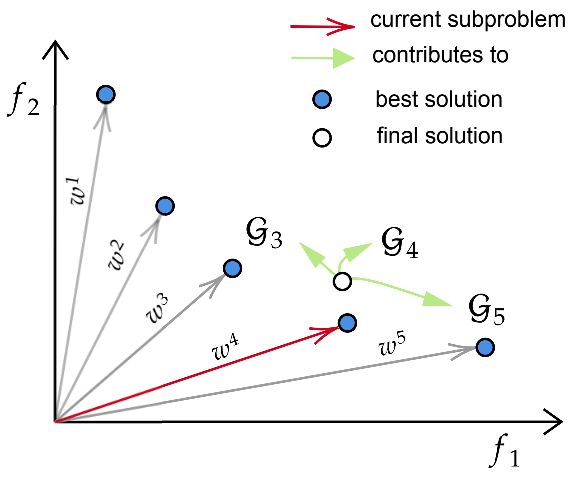


Figure 4.11: Update of knowledge groups with patterns extracted from the final solution obtained, when optimizing subproblem with w_4.

the best non-dominated solutions found during the execution is unbounded.

Our hybridized MOEA/D with the concept of knowledge groups is presented in Algorithm 10. Figure 4.13 shows the main components of the algorithm. The red blocs contain mechanisms related to knowledge discovery. During the initialization step, each group is initialized as empty. The injection is applied with probability p_{inj} on the solution obtained after the crossover. The extraction step occurs after the mutation with probability p_{ext}. Finally, the groups are updated following the strategy described in the former section. The two possible injection strategies lead to two hybridization models. The one using the intensification strategy s_I^i (resp. diversification strategy s_I^d) called $\text{KD1}^i_{\text{MOEA/D}}$ (resp. $\text{KD1}^d_{\text{MOEA/D}}$).

4.3.3 Experimental Protocol

We compare the three algorithms presented in Section 4.3.2, a reference MOEA/D indicated with $\text{R}_{\text{MOEA/D}}$ and two variants hybridized with the knowledge groups $\text{KD1}^i_{\text{MOEA/D}}$ and $\text{KD1}^d_{\text{MOEA/D}}$.

In order to be fair, we tune the three algorithms with an automatic configurator tool, that is **irace** [López-Ibáñez et al., 2016], to evaluate and compare the performances of their best version. **irace**, Iterated Racing for Automatic Configuration, uses racing competitions to discover the best configurations (i.e., values of parameters) to efficiently use an algorithm on a specific problem. A user can define all the parameters wanted for tuning. Each parameter tuned must belong to one of the following categories: *integer*, *real*, *ordinal*, and *categorical*. In practice, an ordinal parameter often takes a finite number of values, and it is meaningful to consider a relation order between the different values. A categorical parameter generally refers to the choice of an operator or the strategy used in an operator (e.g., the exploration strategy in a local search). The values of a categorical parameter are not ordered. Each parameter is associated with a domain, defining the possible values that can be taken by the parameter. The first iteration of **irace** generates a set of random configurations (ensuring diversity in the configuration space) when not enough pre-

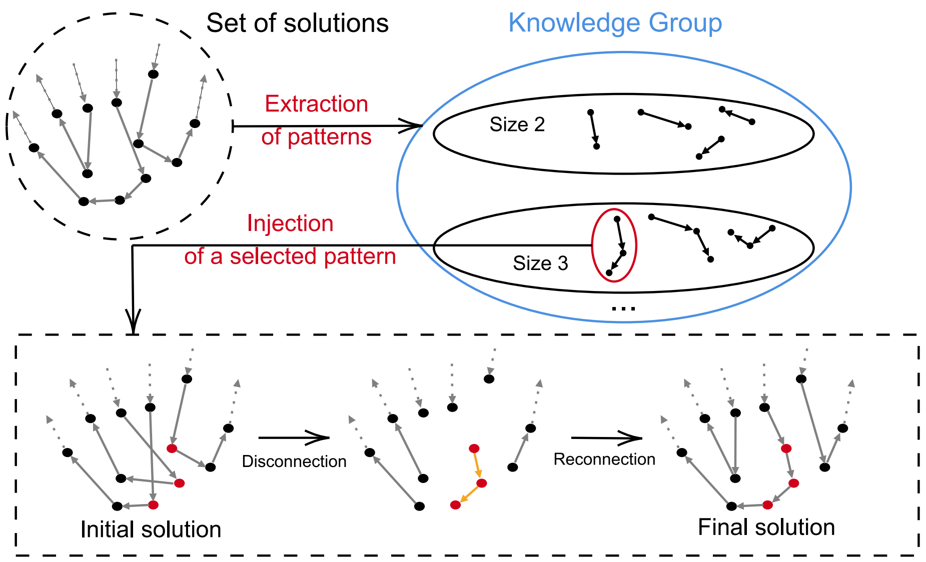


Figure 4.12: Patterns are extracted from solutions, and added to the corresponding knowledge groups. The injection of a frequent pattern of size 3 into a solution is presented.

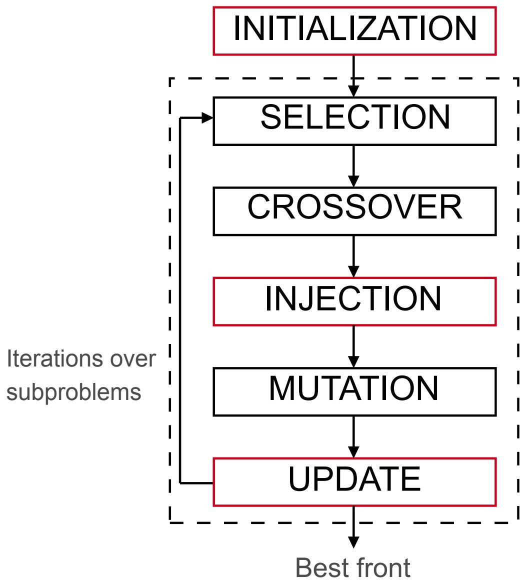


Figure 4.13: Overview of the main components of Algorithm 10. The red blocs contain elements related to the knowledge discovery mechanism.

Algorithm 10: First proposition of hybridization between MOEA/D and knowledge discovery (KD1$_{\text{MOEA/D}}$).

```
Input: $M$ weight vectors $w^1, \dots, w^M$. $m$, the size of the neighborhood of a problem. $p_x$
       (resp. $p_m$) the probability of applying Crossover (resp. Mutation). $s_I$ denotes the
       strategy followed to select patterns for the injection.
Output: The external archive $\mathcal{A}^*$
/* Initialization                                                    */
1  $(\mathcal{A}^*, P') \leftarrow (\varnothing, \varnothing)$
2  $P = \{x^1, \dots, x^M\} \leftarrow$ Initialization()
3  $\mathcal{G}_1, \dots, \mathcal{G}_M \leftarrow \varnothing, \dots, \varnothing$
4  for $i \in \{1, \dots, M\}$ do
5      $\mathcal{N}_i^m \leftarrow$ indexes of the $m$ closest weight vectors to $w^i$
/* Core of the algorithm                                             */
6  while stopping criterion not satisfied do
7      for $i \in \{1, \dots, M\}$ do
8          $(i_1, i_2) \leftarrow$ Select($\mathcal{N}_i^m$)
9          $x \leftarrow$ Crossover($x^{i_1}, x^{i_2}$)
10         $x \leftarrow$ Injection ($\mathcal{G}, s_I, x$)
11         $x \leftarrow$ Mutation($x$)
12         $\mathcal{K} \leftarrow$ Extraction($x$)
13         UpdateGroup($i, \mathcal{K}$)
14         $P' \leftarrow P' \cup \{x\}$
15         UpdatePopulation($P, \mathcal{N}_i^m, x$)
16     UpdateArchive($\mathcal{A}^*, P'$)
17     $P' \leftarrow \varnothing$
18 return $\mathcal{A}^*$
```

defined configurations are provided. The configurations are evaluated by executing the algorithm on randomly selected instances, provided by the user as a training set. Once enough executions are performed, the worst-performing configurations are eliminated by using statistical tests (the Friedman test by default). A new iteration is then started, where new configurations are generated. More precisely, the best-surviving configurations contribute to the refinement of the distribution model of each parameter, allowing the generation of new interesting configurations to evaluate. Figure 4.14 shows an overview of the tuning of an algorithm by using the irace tool.

The algorithms are tuned with the set of instances we generated (see Section 2.3.4). We recall that the hybrid-MOEA/Ds contain 9 parameters: 4 are related to MOEA/D (the number of sub-problems M, the size of the neighborhood m of each subproblem, the probabilities of crossover p_x and mutation p_m), and 5 are related to the learning mechanism (the maximum size of the patterns extracted s_p, the number of patterns injected N_i chosen among the N_f most frequent patterns, and the probabilities of extraction p_{ext} and injection p_{inj}). For the tuning phase, we need to define a range of values for each parameter. All probabilities belong to $[0, 1]$. Note that, if the tuning selects a probability of 0 the associated mechanism is disabled. For the other parameters, we defined sets of integers, where N is the number of customers: $M \in \{1, \dots, 2 \times N\}$, $m \in \{1, \dots, M\}$,

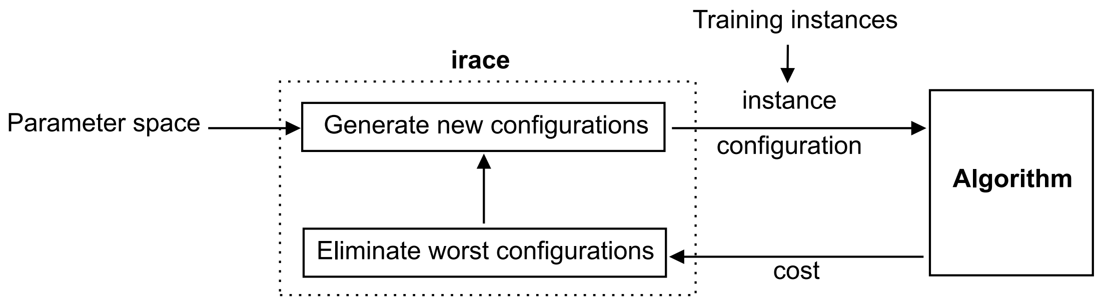


Figure 4.14: Synthesis of `irace`.

$s_p \in \{2, \ldots, 10\}$, $N_f \in \{1, \ldots, 2 \times N\}$, $N_i \in \{1, \ldots, N_f\}$.

We allocated a budget of 2000 configurations (a configuration associates each parameter with a value) to irace. The configurations are tested over 8 iterations. The best configurations returned by irace are presented and discussed in Section 4.3.4.

To evaluate the performances of the three algorithms, we use Solomon's instances. Note that instances of size 25 are discarded due to their small size. All the algorithms share the same maximum running time allocated, which is set to $6 \times N$ seconds, allowing hypervolume convergence for all. The algorithms are executed 30 times on each instance. The k-th run of an instance is executed with the seed $10 \times (k-1)$, so that, all algorithms are compared with the same seeds. To compute the hypervolume, objectives are normalized by using the best and worst objectives found in the final Pareto fronts returned by all the algorithms. We run the experiments on two computers "Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz", with 24 cores each, in parallel by using slurm. The implementation of the algorithms has been realized with the jMetalPy framework [Benitez-Hidalgo et al., 2019]. Once all the results are obtained, the objective vectors of the solutions of the Pareto fronts are normalized according to the best and worst objectives obtained (among all runs and algorithms). The average hypervolume obtained is statistically compared for each instance. A Friedman test is performed to see if all the algorithms are statistically equivalent, when it is not the case, a pairwise Wilcoxon test with Bonferroni correction is performed. The results are reported in Section 4.3.5.

4.3.4 Tuning Results

The best configurations obtained for each algorithm and sizes of instances are reported in Table 4.1.

First, more subproblems are considered in $\text{KD1}^{i}_{\text{MOEA/D}}$ and $\text{KD1}^{d}_{\text{MOEA/D}}$ than in $\text{R}_{\text{MOEA/D}}$. It means that both hybrid MOEA/Ds converge faster towards good solutions and thus require more subproblems to diversify the Pareto front. In all algorithms the crossover occurs with a very low probability, thus the mechanism is almost disabled. Then, concerning the other parameters related to the reference MOEA/D, the three algorithms use similar values. It is worth noting that, the size of the neighborhood is small even with the hybrid MOEA/Ds. It means that the knowledge groups are efficient since a larger neighborhood implies less diverse patterns per group. Indeed, we recall that in a knowledge group $\mathcal{G}_i$, the patterns obtained from solutions generated when optimizing neighbor subproblems of subproblem i are also added. Concerning the parameters dedicated to the learning phase, the values are quite similar between the two hybrid MOEA/Ds. Only the probability of extraction is significantly lower for $\text{KD1}^{d}_{\text{MOEA/D}}$ on instances of size 100,

Table 4.1: The configurations returned by irace for each variant and for both sizes of instance.

Parameters	$R_{\text{MOEA/D}}$		$\text{KD1}^i_{\text{MOEA/D}}$		$\text{KD1}^d_{\text{MOEA/D}}$	
	50	100	50	100	50	100
M	9	16	27	113	31	132
m	8	11	8	13	7	8
p_x	0.11	0.04	0.18	0.05	0.18	0.14
p_m	0.58	0.71	0.65	0.46	0.70	0.27
s_p	-	-	3	6	4	4
p_{ext}	-	-	0.89	0.73	0.82	0.24
N_f	-	-	73	187	72	190
N_i	-	-	8	65	12	51
p_{inj}	-	-	0.50	0.91	0.76	0.24

probably because the patterns injected can come from any knowledge group, and consequently, it is not necessary to update the groups with new patterns very often. One can also remark that more patterns are injected in instances of size 100 than in instances of size 50. It seems, coherent since larger instances are much more difficult to solve than smaller instances.

4.3.5 Experimental Results

The mean hypervolume obtained by each variant on each category of instance is reported in Table 4.2. The detailed hypervolumes for each instance are given in Appendix B. For each category of instance, the values in bold are statistically better. If two values are in bold for the same category, then the average performance values are statistically identical. The biggest improvements are obtained on clustered instances. The hybrid variants $\text{KD1}^i_{\text{MOEA/D}}$ and $\text{KD1}^d_{\text{MOEA/D}}$ always return better hypervolumes than $R_{\text{MOEA/D}}$, showing the effectiveness of the knowledge discovery mechanism. Moreover, both hybrid variants reach similar performances on most instances, indicating that there is not a big interest in considering all groups during the injection step.

The results obtained highlight that the hybridizations between the knowledge discovery mechanism proposed and MOEA/D are efficient in solving a bi-objective VRPTW. The knowledge discovery mechanism is mainly based on the creation of knowledge groups structuring the objective space. Each knowledge group gathers the knowledge of quality-close solutions. With this first approach, we proved that learning from solutions in multi-objective combinatorial optimization is successful and that it is worth spending time learning, instead of optimizing only.

Furthermore, the hybridization proposed may be improved. Indeed, many knowledge groups are created for big instances, which may lead to the learning of redundant knowledge in close groups. To remedy this issue, we present in Chapter 5 an improved strategy to control the number of knowledge groups created independently from the number of subproblems. Another possible improvement concerns the replacement of the mutation operator by a local search, to learn from local optima, which would provide more interesting information. This aspect is investigated in the following sections.

Table 4.2: Average hypervolume obtained on all categories of instances. Bold results are statistically better.

Category	Size	$\text{R}_{\text{MOEA/D}}$	$\text{KD1}^{i}_{\text{MOEA/D}}$	$\text{KD1}^{d}_{\text{MOEA/D}}$
R1	50	0.755	0.968	**0.974**
R2	50	0.727	0.961	**0.973**
R1	100	0.685	**0.982**	**0.987**
R2	100	0.574	0.968	**0.983**
RC1	50	0.695	0.968	**0.980**
RC2	50	0.675	0.961	**0.969**
RC1	100	0.696	**0.987**	**0.989**
RC2	100	0.567	0.974	**0.985**
C1	50	0.673	**0.989**	**0.995**
C2	50	0.553	**0.990**	**0.995**
C1	100	0.586	**0.997**	**0.998**
C2	100	0.476	**0.998**	**0.999**

4.4 Local Search for the bVRPTW

In the former section, we hybridized a version of MOEA/D without a dedicated optimization procedure. This led to poor results for the reference algorithm. In order to be more competitive on the problems solved, we decided to replace the mutation by a local search and we proposed neighborhood strategies that are better adapted to the bVRPTW. Indeed, the mutation is commonly replaced by a local search [Bossek et al., 2018, Knowles, 2002, Land, 1998, Ishibuchi and Murata, 1998], to intensify the search in the regions identified with the crossover. First, we present a new strategy to explore the neighborhood of bVRPTW solutions inspired by the state-of-the-art of permutation flowshop in Section 4.4.1. Second, we propose in Section 4.4.2 a pruning technique that considers not only the distance between the clients but also their respective time windows. We integrated the resulting local search in the hybrid variant $\text{KD1}^{d}_{\text{MOEA/D}}$, which has been presented in the former section. We compared the new algorithm obtained with different algorithms of the literature to solve the original VRPTW (i.e., which minimizes the number of vehicles instead of the total waiting time). Our protocol is described in Section 4.4.4. The results are presented and discussed in Section 4.4.5.

4.4.1 Exploration Strategy

We refer to Section 1.2.1 for the concept of neighborhood exploration and for the presentation of the *best* and *first* strategies. In routing problems, the most commonly used neighborhood exploration strategy is the classical *best* strategy [Subramanian et al., 2013, Schneider et al., 2017, Accorsi and Vigo, 2021], where the best neighboring solution found by the application of one operator on the current solution is selected. The operators considered for the problem are the Relocate, Swap, and 2-opt* and provide a number of possible neighbors which quadratically increases with the size of the problem (see Section 2.6 for more details). When considering large problems, it may be necessary to speed up each neighborhood exploration. However, the *first* strategy does not consider the best move for a considered customer, but only the first neighbor found with a better fitness than the current solution. A compromise between these two classical strategies called *first-best* has been proposed in the literature for scheduling problems, particularly for permutation flow-shop by Ruiz and Stützle [2007]. Algorithm 11 gives the pseudo-code of the *first-best* procedure. The procedure requires a neighborhood operator (e.g., Swap, Relocate, or 2-opt*), and the solution x which undergoes the local search. For the given operator each customer (l.5) is tentatively moved to its best location until an improving move is found (l.8). When it is the case, the customer is moved, and a new iteration starts. The authorized moves are generated by the function generate_moves (l.6).

Algorithm 11: The *first-best* procedure.

```
Input: A solution $x$ and a neighborhood operator $\mathcal{N}$
Output: A better solution (if any)
1 $customers \leftarrow$ shuffle($[1 \dots N]$)
2 for $customer \in customers$ do
3     $moves \leftarrow$ generate_moves($customer$, $\mathcal{N}$)
4     $x' \leftarrow$ best solution obtained by applying a move from $moves$
5     if $f(x') < f(x)$ then
6         $x \leftarrow x'$
7         break
8 return $x$
```

4.4.2 Granular Search

In routing problems, several moves may have a small chance to improve the current solution and thus can be discarded from the exploration without high deterioration to the quality of the solution. Most of the time these moves consider customers that are far, i.e. distant from each other and thus have a small probability of being part of the same route. Having a method that restricts the neighborhood to relevant moves is interesting to spare time and resources during the local search. However, such a method requires a way to quantify the closeness between customers. In Toth and Vigo [2003], the closeness between two customers is evaluated according to the distance between them. If it is enough for single-objective problems, where the total transportation cost is minimized, it might not be adapted for multi-objective problems. In particular, for our bi-objective VRPTW, close customers can incur a long waiting time if they are visited on the same route.

For our study, we compare two different metrics. The first metric called d_1, is the classical metric used in single-objective routing problems: the distance between two customers is simply the Euclidean distance between them. The second metric, d_2, is an aggregation of two quantities, each one linked with one of the objectives optimized. In addition, the weights used to aggregate the quantities depend on the weight vectors used to define the subproblems in MOEA/D. Consequently, the metric associated with a subproblem defined with a weight vector $w = (w_1, w_2)$, is defined as: $d_2^w(i,j) = w_1 \cdot d_E^*(i,j) + w_2 \cdot WT^*(i,j)$, where d_E^* (resp. WT^*) is the normalized value (between 0 and 1) of d_E (resp. WT). Since each value is only instance-dependent, it is possible to compute the maximum and minimum of each value in a pre-processing step at the beginning of the execution. We recall that d_E is the Euclidean distance. $WT(i,j)$ represents the waiting time incurred by going to j from i. If $[a_i, b_j]$ (resp. $[a_j, b_j]$) is the time window of customer i (resp. j), t_i^s the service time of customer i and $t_{i,j}$ the traveling time from i to j, then WT is expressed as follows:

$$WT(i,j) = max(0,\ a_j - (a_i + t_i^s + t_{i,j})). \tag{4.3}$$

Note that a similar formula was defined by Vidal et al. [2013] to focus on a subset of promising arcs. In particular, an arc is considered promising, when it induces a low cost and a low waiting time.

Once the metric between customers is selected, a natural way to prune the neighborhood associated with a customer (i.e., the moves involving that particular customer) is to consider the moves including only the δ nearest customers according to the metric defined. The parameter δ controls the size of the neighborhood explored.

4.4.3 Local Search and Algorithms Considered

The local search considered is a Randomized Variable Neighborhood Descent [Subramanian et al., 2013], where the order of the neighborhood operators is kept during descent but shuffled each time the local search is called. During the local search, we consider two possible exploration strategies, the classical *best*, and our proposition *first-best*. To reduce the size of the neighborhood, we use a granularity parameter δ, where the proximity between customers is defined with metric d_1 (which is the same for all subproblems) or metric d_2 (which depends on the weight vectors associated with the subproblems).

This local search replaces the `Mutation` step of Algorithm 10. It leads to four variations of the algorithm $\text{KD1}^{d}_{\text{MOEA/D}}$: $\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$, $\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$, $\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$, $\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$, where b (resp. fb) denotes the *best* (resp. *first-best*) exploration strategy and d_1 (resp. d_2) denotes the metric used to measure the distance between customers.

4.4.4 Experimental Protocol

We start by tuning the four algorithms $\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$, $\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$, $\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$, $\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$. The protocol for the tuning is the same as the one described in Section 4.3.3. Only the parameter related to the granularity δ is added. We set $\delta \in [0, N]$. The best configurations obtained are displayed in Table 4.3. Compared to Table 4.1, the number of subproblems generated is lower for instances of size 100. The probability of crossover is higher (now that the mutation is replaced, the crossover is more important to generate new different solutions), and the probability of mutation is lower (indeed, the local search requires more computational resources than a random mutation). The granularity is sometimes high (value of 75 for $\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$), but generally, it is set to values that are coherent with the literature. Concerning the learning mechanism, the extraction and injection steps are performed more frequently, indicating that using a local search procedure is more beneficial for the mechanism since better quality solutions are generated.

To compare the four variants, we use each one with its best configuration returned by irace. All the variants use the same termination criteria, being the maximum running time allowed. It is fixed to $N \times 6$ seconds, where N is the size of the instance. Each variant is executed 30 times on each instance of Solomon's benchmark (56 instances of size 50, and 56 instances of size 100). For each algorithm, the k-th run of an instance is executed with the same seed being $10 \times (k-1)$. To compare the results obtained, we use the hypervolume metric. Note that, for the experiments we use the same values to normalize the objectives of the solutions returned by all variants. These

Table 4.3: The configurations returned by irace for each variant and for both sizes of instance.

	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
Parameters	50	100	50	100	50	100	50	100
M	13	68	31	50	42	15	29	15
m	4	26	8	15	6	4	11	4
p_x	0.94	0.30	0.88	0.86	0.93	0.35	0.94	0.67
δ	21	51	25	75	16	19	36	31
p_m	0.06	0.05	0.42	0.55	0.05	0.06	0.11	0.21
s_p	2	3	5	5	2	4	2	5
p_{ext}	0.50	0.96	0.48	0.60	0.55	0.90	0.86	0.83
N_f	73	165	74	135	52	175	66	115
N_i	33	80	17	74	10	63	18	31
p_{inj}	0.70	0.88	0.83	0.93	0.89	0.59	0.86	0.70

values are simply the best and worst values for each objective, obtained among all the (current and past) executions for each instance. Since the extreme values used for normalization may have changed between the former section and the current one, the hypervolume can be lower than those obtained in our former study, even if the front returned is better.

To complete the results obtained, the gap between the best-known and the best solution found by each algorithm is given, as well as the average gap over the 30 runs. The optimal solutions are available on CVRPlib.

Finally, we compare our best variant, considering the results obtained, to state-of-the-art algorithms for the VRPTW: the TS_{tw} from Schneider et al. [2017] and NBD from Nagata et al. [2010a], but also to competitive multi-objective algorithms to solve the VRPTW: the M-MOEAD from Qi et al. [2015] and the MODLEM from Moradi [2020].

The experiments are performed on two computers "Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz", with 24 cores each, in parallel (with slurm). The variants have been implemented using the jMetalPy framework [Benitez-Hidalgo et al., 2019].

4.4.5 Experimental Results

Table 4.4 regroups the average hypervolume obtained on all classes of instances for all the variants. Detailed results per instance are given in Appendix C. One can see that two variants stand out from the others: $\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$ and $\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$. Meaning that the exploration strategy has a positive impact on the performance of the algorithm, and thus it is better than the strategy *best*. However, the variant $\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$ returns slightly higher hypervolumes than $\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$ in most instances, and clearly outperforms $\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$ in few instances (e.g. RC1 of size 50 and C1 of size 100).

Now we analyze the gaps between the best solutions returned for the total cost objective and the best-knowns. The average gaps obtained are reported in Table 4.5. Detailed results are provided in Appendix C. In these tables, for each variant, the first column is the gap between the best-known and the best solution returned, while the second column is the average gap considering the solutions returned on all 30 runs. The best solution is achieved in almost all instances of category C, but the gap to the optimal remains important in other categories of instance. In particular,

Table 4.4: Average hypervolume obtained with the four variants on all classes of instance of both sizes.

Class	Size	$\mathrm{KD1}^{d,b,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{d,b,d_2}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{d,fb,d_2}_{\mathrm{MOEA/D}}$
R1	50	0.716	0.768	0.729	**0.783**
R2	50	0.666	**0.747**	0.690	0.743
R1	100	0.773	**0.842**	0.720	0.833
R2	100	0.626	**0.760**	0.615	0.747
RC1	50	0.604	**0.760**	0.611	0.689
RC2	50	0.637	0.682	0.647	**0.692**
RC1	100	0.682	**0.758**	0.631	0.739
RC2	100	0.658	0.766	0.662	**0.769**
C1	50	0.519	**0.574**	0.523	0.550
C2	50	0.404	0.408	**0.414**	0.403
C1	100	0.881	**0.945**	0.846	0.882
C2	100	0.899	**0.967**	0.858	0.954

it indicates that the algorithm may be further improved. In addition, it is harder to reach the optimal transportation cost, since we optimize a second objective, and the time allocated may not be enough to reach that optimal. On the other hand, the proposed algorithms remain quite simple with few operators, and they are able to reach good results overall. One can notice that the variants $\mathrm{KD1}^{d,fb,d_1}_{\mathrm{MOEA/D}}$ and $\mathrm{KD1}^{d,fb,d_2}_{\mathrm{MOEA/D}}$ still outperform the two other variants. However, this is the variant $\mathrm{KD1}^{d,fb,d_1}_{\mathrm{MOEA/D}}$ which returns the best results in most instances.

Considering the results obtained above, we decided to compare the variant $\mathrm{KD1}^{d,fb,d_2}_{\mathrm{MOEA/D}}$ to other state-of-the-art algorithms. Table 4.6 compares the average value of the best transportation cost obtained by different algorithms on each class of instance of size 100. We recall that, there are two single-objective algorithms: TS_{tw} [Schneider et al., 2017] and NBD [Nagata et al., 2010a], and two multi-objective algorithms: M-MOEA/D [Qi et al., 2015] and MODLEM [Moradi, 2020]. Note that MODLEM integrates a learning mechanism, which is a learnable evolution model based on decision trees. Moreover, the algorithms that solve the VRPTW, first minimize the number of vehicles and then the traveled distance. To be fair, we add in brackets the average number of vehicles contained in the solutions returned by our algorithm. Since our algorithm did not focus on the number of vehicles, it seems normal that the average number of vehicles used in the solutions returned is higher than the one found by other algorithms. However, our algorithm is able to reach competitive results on C instances, and good results on the other instances.

Through our experiments, we showed that the *first-best* exploration strategy performs better than the *best* strategy in MOEA/D to solve many instances of the bVRPTW. Moreover, the granular search based on the proposed metric taking into account the waiting times is able to reach similar results to the one based on the Euclidean distance, but with smaller neighborhoods. Thus reducing the running time of the associated local search. Finally, the performance of our algorithm compared to state-of-the-art algorithms for the VRPTW shows the interest of our new neighborhood strategies. In the next section, we investigate more precisely the impact of the solutions returned by the local search on the learning mechanism.

Table 4.5: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-knowns for each category of instance.

Instance	Size	$\mathrm{KD1}^{d,b,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{d,b,d_2}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{d,fb,d_2}_{\mathrm{MOEA/D}}$
R1	50	1.9	**1.4**	1.8	**1.3**
R2	50	3.3	**2.7**	3.1	**2.6**
R1	100	5.5	**4.4**	6.1	**4.4**
R2	100	8.8	**5.7**	8.2	**5.7**
RC1	50	4.4	**2.6**	4.4	3.4
RC2	50	2.3	**1.9**	2.0	**1.4**
RC1	100	8.1	**7.0**	8.8	**7.3**
RC2	100	8.7	**5.0**	7.6	**4.6**
C1	50	1.3	**0.4**	1.1	**0.8**
C2	50	**0.6**	**0.6**	**0.6**	**0.5**
C1	100	3.4	**2.2**	5.1	3.8
C2	100	1.0	**0.4**	1.4	**0.5**

Table 4.6: Comparison of the average of the best total transportation cost obtained on instances of size 100 between four state-of-the-art algorithms and our algorithm $\mathrm{KD1}^{d,fb,d_2}_{\mathrm{MOEA/D}}$. The corresponding average number of vehicles used in the solutions is given in brackets.

Class	NBD	TS_{tw}	M-MOEA/D	MODLEM	$\mathrm{KD1}^{d,fb,d_2}_{\mathrm{MOEA/D}}$
R1	1210.34 (11.9)	1220.83 (11.9)	1216.73 (12.4)	1210.40 (11.9)	1196.22 (13.8)
R2	951.03 (2.7)	959.86 (2.7)	924.18 (3.1)	916.95 (4.6)	892.85 (5.0)
RC1	1384.16 (11.5)	1392.54 (11.5)	1390.35 (11.9)	1384.17 (11.5)	1387.11 (13.8)
RC2	1119.24 (3.3)	1140.13 (3.3)	1119.93 (3.4)	1074.67 (4.0)	1015.76 (5.8)
C1	828.38 (10.0)	828.38 (10.0)	828.38 (10.0)	828.38 (10.0)	**827.02** (10.0)
C2	589.86 (3.0)	589.86 (3.0)	589.86 (3.0)	589.86 (3.0)	**587.38** (3.0)

4.5 Impact of Solutions for Learning

In single-objective problems, the learning tasks are often performed on local optima [Khalil et al., 2017, Arnold and Sörensen, 2019, Arnold et al., 2021], which tend to contain more relevant structural information. In particular, it highlights the importance of the structure of the solutions themselves during the learning. In the following, we investigate the benefit of extracting knowledge from local optima in a multi-objective context and more precisely to solve a bVRPTW. To that aim, we considered the MOEA/D hybridized with the knowledge groups, described in Algorithm 10. Note that, in the remainder of this section, the mutation is replaced by the local search described in the former section, with the *first-best* exploration strategy and the d_1 metric to measure the proximity between customers. In Section 4.5.1, we present four related variants, each one with a different strategy for extraction (i.e., learning from all solutions or local optima only) and injection (i.e., the s_I^i and s_I^d strategies already described in Section 4.3.1). The experimental protocol is presented in Section 4.5.2 and the results in Section 4.5.3.

4.5.1 Extraction Strategies and Compared Algorithms

The reference MOEA/D considered is the one described in Algorithm 9, where the mutation is replaced by the local search, with the *first-best* exploration strategy and the metric d_1, being the Euclidean distance, to measure the proximity between customers. This algorithm is called $\mathrm{R}_{\mathrm{MOEA/D}}^{fb,d_1}$ in the following.

From now we consider the hybridized version of MOEA/D described in Algorithm 10. We recall that the injection step is applied with probability p_{inj} and can follow two different strategies s_I^i and s_I^d. The first one is an intensification strategy, where the search is focused on a specific part of the exploration space. In this case, only the knowledge group associated with the current subproblem can provide patterns for the injection. The other strategy concerns diversification, where the knowledge of all the groups can be used, to favor a larger exploration of the space. In this case, the patterns to inject come from a knowledge group randomly chosen among all the existing groups.

The extraction step is performed after the local search, with a probability p_{ext}. We now consider two strategies to investigate the benefit of extracting knowledge from local optima. The first strategy, called s_E^{all}, allows the extraction on the current solution at the moment where the extraction step is applied (i.e., it is the same as before). With this strategy, any solution found can undergo the extraction step. The second strategy, called s_E^{lo}, allows the extraction on local optima solutions only. More precisely, suppose that the current solution before the local search step is x. If x undergoes the local search, and an improving solution x' is returned after the local search, then x' may undergo the extraction step (with the associated probability). If the local search is not applied, x can not undergo the extraction step. In other words, the extraction can be applied only if the local search has been applied just before.

Finally, our design leads to the design of the four following variants: $\mathrm{KD1}_{\mathrm{MOEA/D}}^{lo,i,fb,d_1}$, $\mathrm{KD1}_{\mathrm{MOEA/D}}^{lo,d,fb,d_1}$, $\mathrm{KD1}_{\mathrm{MOEA/D}}^{all,i,fb,d_1}$, and $\mathrm{KD1}_{\mathrm{MOEA/D}}^{all,d,fb,d_1}$. The term *lo* (resp. *all*) indicates that the extraction is performed on local optima only (resp. on all solutions), and i (resp. d) indicates that the strategy s_I^i (resp. s_I^d) is followed by the injection.

Table 4.7: The configurations returned by irace for each variant and for both sizes of instance.

Parameters	$\text{R}^{fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$	
	50	100	50	100	50	100	50	100	50	100
M	94	68	50	52	61	60	23	45	31	50
m	73	55	15	6	15	5	7	20	8	15
p_x	0.94	0.99	0.85	0.61	0.76	0.25	0.97	0.78	0.88	0.86
δ	16	67	24	66	28	60	21	73	25	75
p_m	0.71	0.86	0.49	0.48	0.54	0.30	0.49	0.51	0.42	0.55
s_p	-	-	4	6	4	5	5	6	5	5
p_{ext}	-	-	0.47	0.68	0.46	0.80	0.45	0.68	0.48	0.60
N_f	-	-	132	180	124	150	86	171	74	135
N_i	-	-	48	70	32	60	25	44	17	34
p_{inj}	-	-	0.85	0.91	0.81	0.97	0.86	0.97	0.83	0.93

4.5.2 Experimental Protocol

As usual, the five algorithms are tuned following the same protocol as the one presented in Section 4.4.4. The configurations obtained are available in Table 4.7. Compared to Table 4.1, the algorithm $\text{R}^{fb,d_1}_{\text{MOEA/D}}$ uses more subproblems than $\text{R}_{\text{MOEA/D}}$ showing that the local search is beneficial to solve more subproblems. The probability of crossover is also increased since it is the only operator that brings diversity to the solutions explored. Concerning the hybridized variants, we find configurations that are similar to those found in previous sections (see Table 4.1 and Table 4.3). Moreover, we can remark that in variants learning on local optima only (i.e., $\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$ and $\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$), the number of patterns injected N_i is lower than in variants learning on all solutions (i.e., $\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$ and $\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$). Indeed, learning from local optima may provide better quality patterns thus it is not necessary to try injecting as many patterns to achieve good results.

In the following, three batches of experiments are considered. The first batch evaluates the impact of the knowledge discovery mechanism, according to the different strategies followed by the injection and the extraction steps, by comparing the hybridized variants and the reference algorithm with the same parameters. The second batch evaluates the performance (in terms of quality) of all the algorithms with their best parameters. The last batch analyses the speed-up of the variants compared with $\text{R}^{fb,d_1}_{\text{MOEA/D}}$ to reach a target hypervolume. To compute the hypervolume, objectives are normalized by using the best and worst objectives found in the final Pareto fronts returned by all the algorithms (from previous and current experiments).

For the first batch of experiments, we define a new configuration by considering the configurations obtained for all variants with irace. We take the 5 parameters of $\text{R}^{fb,d_1}_{\text{MOEA/D}}$ and for the parameters associated with the knowledge discovery mechanism, we compute the mean of the corresponding parameters of the hybridized variants, and we round the result, to conserve the same precision. Each variant is executed 30 times on each instance of Solomon's benchmark (56 instances of size 50, and 56 instances of size 100). For each algorithm, the k-th run of an instance is executed with the seed $10 \times (k-1)$, so that, all algorithms are compared with the same seeds. The termination criterion is fixed to $N \times 6$ seconds, where N is the size of the instance. The results are

Table 4.8: Average hypervolume obtained with the algorithms when they all have the same parameters.

Class	Size	$\text{R}^{fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$
$C1$	50	0.558	**0.827**	**0.825**	**0.825**	**0.825**
$R1$	50	0.657	0.**852**	**0.855**	0.823	0.801
$RC1$	50	0.574	**0.802**	**0.804**	**0.808**	**0.803**
$C2$	50	0.514	**0.574**	**0.571**	**0.577**	**0.576**
$R2$	50	0.687	**0.780**	**0.780**	0.752	0.751
$RC2$	50	0.618	**0.715**	**0.717**	**0.718**	**0.718**
$C1$	100	0.337	**0.935**	**0.941**	**0.940**	**0.940**
$R1$	100	0.550	**0.737**	**0.742**	**0.736**	**0.740**
$RC1$	100	0.374	**0.699**	**0.698**	**0.693**	**0.698**
$C2$	100	0.637	**0.981**	**0.979**	**0.976**	**0.978**
$R2$	100	0.513	**0.617**	**0.613**	**0.620**	**0.615**
$RC2$	100	0.556	0.676	**0.685**	**0.681**	**0.681**

compared using the hypervolume.

For the second batch of experiments, we proceed similarly to the first batch, except that we use the configurations returned by irace for each variant.

Finally, for the third batch of experiments, we evaluate the speed of the hybridized variants to reach 95% of the mean hypervolume returned by $\text{R}^{fb,d_1}_{\text{MOEA/D}}$. The termination criterion is slightly different from the other batches since we consider the value of the hypervolume to be reached. Except for that change, the remaining of the experiment is similar to the previous one (i.e., considering the number of executions and the configurations). Note that, for all the experiments, we use the same values to normalize the objectives of all variants. These values are obtained with the first experiment and are simply the best and worst values obtained among all the executions. It allows an easy computation of the hypervolume during the execution of the algorithm.

The following experiments are performed on two computers "Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz", with 24 cores each, in parallel (with slurm). The variants have been implemented using the jMetalPy framework [Benitez-Hidalgo et al., 2019].

4.5.3 Experimental Results

For all the results presented below, supplementary tables with results per instance are provided in Appendix D. Table 4.8 shows the average hypervolume obtained with all algorithms on the different categories of instance. First one can see that all hypervolumes returned by the KD variants are strictly higher than the hypervolumes returned by $\text{R}^{fb,d_1}_{\text{MOEA/D}}$. Bold results represent significantly better performances. The improvement (regarding $\text{R}^{fb,d_1}_{\text{MOEA/D}}$) obtained with the KD variants on instances of class 1 is significantly better compared to the improvement obtained on instances of class 2. We recall that instances of class 2 contain wider time windows and thus are less constrained.

Table 4.9 shows the mean hypervolume obtained when the parameters are chosen by irace for each variant. It leads to an overall improvement of the results of Table 4.8 (except for the algorithms

Table 4.9: Average hypervolume obtained with the algorithms when they all use their own elite configuration returned by irace.

Class	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
$C1$	50	0.558	**0.839**	**0.839**	**0.829**	**0.830**
$R1$	50	0.657	**0.852**	**0.855**	0.823	0.801
$RC1$	50	0.574	**0.884**	**0.891**	0.861	0.846
$C2$	50	0.514	**0.590**	**0.592**	0.565	0.556
$R2$	50	0.687	**0.780**	**0.780**	0.752	0.751
$RC2$	50	0.618	**0.761**	**0.763**	0.746	0.739
$C1$	100	0.337	0.919	0.941	0.939	**0.969**
$R1$	100	0.550	0.824	**0.865**	**0.862**	**0.868**
$RC1$	100	0.374	0.730	0.821	0.834	**0.859**
$C2$	100	0.637	0.828	0.934	**0.972**	**0.984**
$R2$	100	0.513	0.673	0.728	**0.741**	**0.753**
$RC2$	100	0.556	0.669	0.755	0.790	**0.815**

learning on local optima on some C2 and R1 instances). More importantly, we see that learning on local optima is mainly beneficial for instances of size 100 (on average the variant $\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$ returns the best results on these instances). However, it is outperformed by the other KD variants on instances of size 50. In instances of size 50, it seems more interesting to learn from any solutions found. Indeed the algorithm may get stuck easily on the same local optima, since the instance is easier to solve, which does not add useful information to the mechanism.

Table 4.10 gives the gaps with the best-known solutions for the total cost objective. The results highlight that our KD variants return much better results than $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$. However, in some instances (e.g., RC1 of size 100), the gaps obtained are still high ($> 2\%$), meaning that in a multi-objective context, it is harder to find the optimal value of each objective. Moreover, larger gaps are obtained on instances of size 100, showing scalability issues. One can also notice that the gaps obtained are slightly lower than those obtained in Table 4.5 for the instances of size 50. On average, $\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$ returns the smallest gaps on instances of size 50, while $\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$ returns the smallest gaps on instances of size 100.

Table 4.11 shows the speed-up of the KD variants to reach 95% of the mean hypervolume returned by $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$. The table shows that we reach an average speed-up of 73.5% (resp. 64.1%) on instances of size 50 (resp. 100) compared to $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$, leading to a significant improvement. By considering the results in Table 4.9, we observe that $\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$ reaches good results faster than $\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$ on instances of size 100, but then slows down and is outperformed by $\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$. On instances of size 50, it is the intensification variant $\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$, which is the fastest variant on average.

The question raised and discussed in this section concerns the impact of extracting knowledge from local optima solutions only. To that aim, four variants of MOEA/D are designed, depending on the strategy followed during the injection step and depending on the solutions used during

Table 4.10: Mean gap (%) obtained regarding the total cost objective. The gaps are computed with the optimal value known for each instance.

Class	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
$C1$	50	10.86	**0.07**	**0.08**	0.40	0.40
$R1$	50	3.09	**0.90**	**0.92**	1.23	1.43
$RC1$	50	7.41	**1.72**	**1.56**	2.17	2.54
$C2$	50	2.15	**0.35**	**0.35**	**0.58**	**0.55**
$R2$	50	4.56	**2.37**	**2.40**	2.66	2.65
$RC2$	50	7.31	1.52	**1.22**	1.97	1.95
$C1$	100	35.03	5.13	3.79	4.02	**2.22**
$R1$	100	10.44	5.15	**4.43**	**4.51**	**4.37**
$RC1$	100	18.84	10.31	8.11	7.67	**7.03**
$C2$	100	5.67	2.64	1.24	**0.54**	**0.38**
$R2$	100	9.58	7.81	6.58	6.15	**5.71**
$RC2$	100	12.02	11.00	7.61	6.24	**5.04**

Table 4.11: Mean gain (%) obtained, in terms of speed-up, with respect to $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$, to reach 95% of the mean hypervolume of $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$.

Class	Size	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
$C1$	50	**85.0**	79.9	84.4	80.8
$R1$	50	73.7	70.4	**76.7**	74.3
$RC1$	50	**84.7**	81.7	83.5	83.9
$C2$	50	64.5	**64.7**	59.2	55.5
$R2$	50	**65.8**	58.9	65.6	63.6
$RC2$	50	78.6	72.6	**78.9**	77.9
$C1$	100	73.4	**81.2**	76.2	69.9
$R1$	100	55.7	**65.9**	62.6	54.4
$RC1$	100	68.1	**75.6**	73.5	68.6
$C2$	100	59.9	**70.0**	68.0	62.5
$R2$	100	53.4	**58.2**	56.4	46.6
$RC2$	100	57.0	61.9	**62.7**	57.0

extraction. In particular, the variant $\text{KD1}_{\text{MOEA/D}}^{all,d,fb,d_1}$ (resp. $\text{KD1}_{\text{MOEA/D}}^{lo,d,fb,d_1}$) extracts the knowledge from all potential (resp. local optima) solutions found, and follows a diversification strategy during the injection, meaning that we use knowledge from all groups to favor larger exploration of space. Some experiments were conducted on Solomon's instances of the bi-objective VRPTW and showed the benefit of exploiting knowledge to optimize better solutions. Additionally, extracting patterns from local optima ($\text{KD1}_{\text{MOEA/D}}^{lo,d,fb,d_1}$), especially for larger instances, is preferable to obtain better solutions. The investigation of the speed-up reveals that it is possible to converge faster towards good solutions when the learning exploits the knowledge from all solutions found instead of focusing on local optima only. More precisely, the variant $\text{KD1}_{\text{MOEA/D}}^{all,d,fb,d_1}$ converges faster towards good solutions. In practice, it means that focusing on local optima solutions only is not a necessity to achieve quickly good performances, however exploiting local optima solutions allows convergence to better solutions on bigger instances.

4.6 Conclusion

Through this chapter, we proposed a knowledge discovery mechanism based on knowledge groups associated with the subproblems defined in MOEA/D. More precisely, each subproblem is associated with a knowledge group carrying the patterns extracted from solutions discovered when optimizing that subproblem or its neighbors. Inside each knowledge group, each pattern is linked with its current frequency of appearance, and the most frequent patterns can be exploited during an injection step to improve another solution. Since each knowledge group learns the most frequent structures for each subproblem (and for the associated region of the objective space), the injection integrates the knowledge from a region of the objective space into a solution belonging to the same region or a different one.

The experiments performed show an interest in using such a mechanism (i.e., it increases the performances of MOEA/D in terms of quality and speed-up). However, the mechanism remains too dependent on MOEA/D. For example, the number of groups can not be controlled and depends on the number of subproblems generated in MOEA/D. Suppose we decide to use a strategy that modifies the subproblems (e.g., by modifying the associated weight vectors). In that case, the associated knowledge groups and the patterns learned may not be exploitable anymore. Moreover, with the current mechanism, it is not possible to consider its integration into another kind of multi-objective algorithm, which would not use any decomposition strategy.

The next chapter presents a different construction for the knowledge groups, in order to control the number of groups created. With this new construction, the strategies for the extraction and the injection are modified accordingly.

Chapter 5

Improvement of Knowledge Groups and Parameters Analysis

Contents

This chapter's contributions are linked to the following publications:

- Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, Marie-Eléonore Kessaci. Improving MOEA/D with Knowledge Discovery. Application to a Bi-objective Routing Problem. EMO 2023-Evolutionary Multi-Criterion Optimization, Mar 2023, Leiden, Netherlands. pp.462-475, `https://hal.science/hal-04040436/`.

- Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, M. E. (2023). Improving neighborhood exploration into MOEA/D framework to solve a bi-objective routing problem. International Transactions in Operational Research. `https://onlinelibrary.wiley.com/doi/pdfdirect/10.1111/itor.13373`.

5.1 Introduction

The main issue with the concept of knowledge groups presented in the former chapter concerns its dependency on MOEA/D. Indeed, the knowledge groups are intrinsically connected to the weight vectors generated by MOEA/D. In particular, if the weight vectors are modified during the execution of MOEA/D, there is no guarantee to preserve the coherence of the associated knowledge groups. More importantly, there is no control over the number of groups generated and the regions covered by the knowledge groups (since they depend on the solutions found while optimizing the subproblems).

In this chapter, we present an enhancement of the construction of the knowledge groups proposed in the former chapter to remedy the issue mentioned above. Section 5.2 describes the new construction, with a particular focus on the terminology adopted concerning the definition of the regions of the knowledge groups, and strategies associated with the extraction and injection steps. In addition to this improvement, a new hybridization with MOEA/D is proposed, which modifies where (and when) the extraction step is performed. The experiments compare variants of the hybridization with different values for the number of knowledge groups constructed and different strategies for the injection step. The results, presented in Legrand et al. [2023b], highlight the interest in using several knowledge groups with an intensification strategy for both extraction and injection steps.

We push further the analysis of the hybridization in Section 5.3, where most parameter values are left to the tuning and the influence of each (tuned) parameter is analyzed. Concerning the strategies for extraction and injection, we only consider intensification, which is more transparent to know where each knowledge group contributes. Moreover, using different strategies for the extraction and the injection could bias the study of the number of knowledge groups. With this analysis, we hope to discover that the parameters related to the knowledge discovery mechanism positively impact the performances of the algorithms, and which values should be used to obtain the best performances. Thanks to this work, we were able to propose more interesting domains for each parameter, leading to more robust configurations found by irace. Finally, the results, presented in Legrand et al. [2023a], show the efficiency of our proposed strategies.

5.2 Improvement of Knowledge Groups

In this section, we propose to improve the knowledge discovery mechanism introduced in the former chapter. Although it is effective with MOEA/D, the mechanism is not flexible enough to be integrated into other multi-objective algorithms. In particular, it is too dependent on the subproblems defined. In practice, we would like to have a mechanism where the number of groups and their construction can be adapted to the instance. Indeed, according to the shape of the optimal Pareto front (when it is known) and of the objective space, we may prefer one construction to another, so that the division of the objective space is adapted to the instance to be solved. Since in MOEA/D there already exists plenty of methods to decompose the the objective space, we would like to make these strategies available for the construction of the knowledge groups.

We introduce the new construction framework in Section 5.2.1 and we provide an instantiation in the case where the objective space is divided with weight vectors. In Section 5.2.2, we discuss the possible strategies that can be used for the extraction and injection steps. These strategies generalize the ones used in the former chapter for the injection. Then, we present the modifications brought to the former hybridizations and the different variants compared in Section 5.2.3. Our setup is described in Section 5.2.4, our protocol in Section 5.2.5, and the results obtained are presented in Section 5.2.6.

5.2.1 Decomposition-based Knowledge Groups

To bring more flexibility to the knowledge groups, we need to associate them with a region of the objective space. We explain how to define such regions in Section 5.2.1.1. Moreover, an example is provided in Section 5.2.1.2 when weight vectors are used to decompose the objective space. Indeed, using weight vectors is undoubtedly the most common strategy to decompose the objective space.

5.2.1.1 Generalized Knowledge Groups

Suppose that we want to define k_G knowledge groups to store the knowledge extracted during the execution. We propose to divide the objective space into k_G regions each one representing a knowledge group. The region can be defined explicitly or implicitly. The set of knowledge groups is denoted as $\mathcal{G}$.

We consider that a region is explicitly defined when it is defined with equations (e.g., with curves or hyperplanes). In that case, if all regions cover entirely the objective space, it is easy to know if a solution belongs to a specific region, however, distances to other regions may be harder to compute. Moreover, we would recommend using simple equations to define the regions. Indeed, it will simplify the computations required, and make easier the interpretation of the regions. Explicit regions might be more adapted in a continuous context.

Most of the time, it is easier to consider implicitly defined regions. In that case, each region is associated with a unique representative. The representative can be a point in the objective space, or something else while it is possible to define a distance between the representative and the objective vector of a solution. Given the representative g^i of the region defining the knowledge group $\mathcal{G}_i$, the region of the group is defined as the set of all solutions whose closest representatives is g^i. In two dimensions, given a set of points, the region associated with one point corresponds to a Voronoï cell. With implicit representations, the regions depend on the representatives, and modifying one representative may impact the other regions. It was not the case with explicit representations, changing one equation does not change the others. Moreover, in implicit representations, a solution is always associated with (at least) one knowledge group, being the one associated with the closest representative.

In the next section, we provide an example of knowledge groups where regions are implicitly defined, and where the representatives of groups are weight vectors.

5.2.1.2 Weight Vectors based Knowledge Groups

We describe how to use a weight vector decomposition to create the knowledge groups. With this construction, the regions associated with the knowledge groups are implicitly defined. Let $(g^1, \ldots, g^{k_G})$ be a set of k_G weight vectors to define the corresponding k_G knowledge groups. The vectors can be uniformly spread or not, but we consider that they do not change during the execution.

Let x be a feasible solution found during the execution. To evaluate the proximity between x and a representative g^i, we compute the quantity $g^{WS}_{fit}(x, g^i)$ which is exactly the fitness of the solution obtained when g^i generates a subproblem with the weighted sum approach (see Equation 1.7). The solution is then associated with the representative g^{i^*}, where i^* verifies:

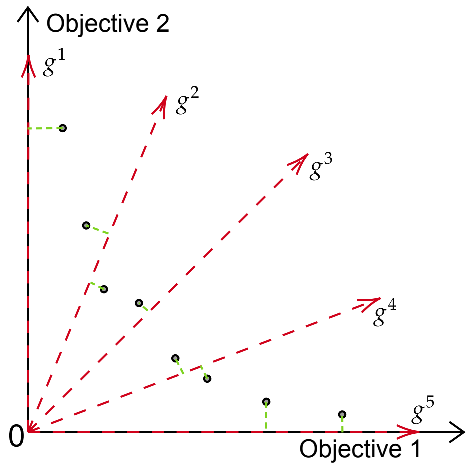


Figure 5.1: Construction of five knowledge groups based on weight vectors. Some solutions are represented and associated with their closest knowledge groups.

$$i^* = \underset{1 \leq i \leq k_G}{\operatorname{argmin}} \, g_{fit}^{WS}(x, g^i)$$

If the minimum fitness is attained for multiple representatives, the representative is randomly chosen among them. In other words, the group $\mathcal{G}_{i^*}$ is the most adapted to store the knowledge extracted from the solution x.

Figure 5.1 shows how solutions are associated with knowledge groups defined with weight vectors. Geometrically, the distance between one solution and a knowledge group $\mathcal{G}_i$ of weight vector g^i is the distance between the solution and its orthogonal projection to the line passing through 0 with direction g^i.

5.2.2 Extraction and Injection Strategies

Evolutionary algorithms use intensification and diversification mechanisms to explore the search space more in-depth or more largely. We propose to transpose these mechanisms of intensification and diversification to the knowledge discovery for the extraction (in Section 5.2.2.1) and injection (in Section 5.2.2.2) mechanisms. On the one hand, we propose an intensification strategy, where the procedure has access to a small number of groups. With this strategy, the objective is to focus on the same region of the objective space, by exploring close regions to the current solution. In that case, the knowledge is not widely shared between the groups. On the other hand, with a diversification strategy, the procedure has access to a large number of groups. The diversification aims to explore different regions of the objective space, by bringing diversity to the solutions. In that case, the knowledge can be easily shared through all the groups.

5.2.2.1 Strategies Related to the Extraction

The strategy of the extraction defines the number of knowledge groups that are updated with the knowledge extracted from one solution. When a few groups are updated, we consider that the

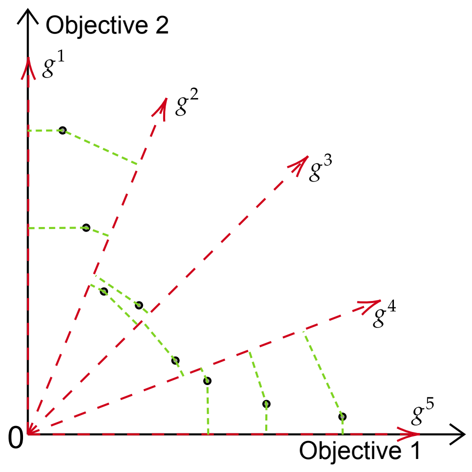


Figure 5.2: Representation of the extraction strategy in the case $d_e = 2$ (i.e., the patterns of each solution are sent to the two closest groups).

extraction follows an intensification strategy. Increasing the number of groups updated brings more diversity to the extraction step.

The parameter $d_e \in \{1, \dots, k_G\}$ controls the number of groups updated. With $d_e = 1$, only the closest group to the solution is updated (this situation is represented in Figure 5.1). With $d_e = k_G$ all the groups are updated. Note that it is not interesting to update all groups since it is equivalent to having only one group.

In the following, we decided to focus on the most intensive strategy only (i.e., $d_e = 1$). In fact, in the construction of groups proposed in Chapter 4, the number of groups updated during the extraction step was dependent on the neighborhood size of the subproblems, allowing us to already test higher values of diversities. We remarked after the tuning steps that a small neighborhood size is generally preferred, thus we decided to consider a low value for the diversity of the extraction, and fixing $d_e = 1$ allows better comprehension of the knowledge learned.

5.2.2.2 Strategies Related to the Injection

The strategy of the injection defines the number of knowledge groups that can provide patterns to inject into the current solution. Similarly to the extraction, the more groups are considered, the higher the diversity of the mechanism.

We introduce a parameter $d_i \in \{1, \dots, k_G\}$ to control the strategy followed by the injection. With $d_i = 1$, only the closest group to the solution can provide patterns to be injected. With $d_i = k_G$ any existing group can be selected to provide the patterns.

In the following, we decided to consider only the two extreme strategies (i.e., $d_i = 1$ and $d_i = k_G$), which were the strategies adopted in Chapter 4. Using these two strategies allow us to compare the new construction of groups to the results obtained previously.

Algorithm 12: Second proposition of hybridization between MOEA/D and knowledge discovery ($\text{KD2}_{\text{MOEA/D}}$).

```
Input: $M$ weight vectors $w^1, \dots, w^M$. $m$, the size of the neighborhood of a problem. $p_x$
       (resp. $p_m$) the probability of applying Crossover (resp. Mutation). $s_I$ (resp. $s_E$)
       denotes the strategy followed to select patterns for the injection (resp. to select
       groups updated during the extraction).
Output: The external archive $\mathcal{A}^*$
/* Initialization                                                                    */
1  $(\mathcal{A}^*, P') \leftarrow (\emptyset, \emptyset)$
2  $P = \{x^1, \dots, x^M\} \leftarrow$ Initialization()
3  $\mathcal{G} \leftarrow$ CreateGroups($k_G$)
4  for $i \in \{1, \dots, M\}$ do
5      $\mathcal{N}_i^m \leftarrow$ indexes of the $m$ closest weight vectors to $w^i$
/* Core of the algorithm                                                             */
6  while stopping criterion not satisfied do
7      for $i \in \{1, \dots, M\}$ do
8          $(i_1, i_2) \leftarrow$ Select($\mathcal{N}_i^m$)
9          $x \leftarrow$ Crossover($x^{i_1}, x^{i_2}$)
10         $x \leftarrow$ Injection ($\mathcal{G}, s_I, x$)
11         $x \leftarrow$ Mutation($x$)
12         $P' \leftarrow P' \cup \{x\}$
13         UpdatePopulation($P, \mathcal{N}_i^m, x$)
14     UpdateArchive($\mathcal{A}^*, P'$)
15     UpdateGroup($\mathcal{G}, s_E, P'$)
16     $P' \leftarrow \emptyset$
17 return $\mathcal{A}^*$
```

5.2.3 Integration into MOEA/D

The hybridization is the one presented in Algorithm 12, where the mutation is the local search already presented in Section 4.4, with the *first-best* exploration strategy and the metric d_2, which considers the Euclidean distance and the waiting time between customers to evaluate their proximity. The algorithm slightly differs from Algorithm 10 (the modifications appear in red). At l.3, a new function `CreateGroups` is used to initialize the k_G groups, with their own representative, or their regions. Each knowledge group object is defined during the execution of this function, and in particular, the function evaluating the distance between a group and a solution must be defined. Here, the construction adopted is the one described in Section 5.2.1.2. Moreover, to avoid the same solution being added several times to the same group, each group maintains a set of already-seen objective vectors (assuming that, there is a unique association between solutions and objective vectors). This strategy is considered to avoid the bias from local optima easy-to-find (i.e., attractive solutions).

The injection step does not change and is still performed between the crossover and the mutation. The injection can be performed with a strategy s_I^i (resp. s_I^d) where $d_i = 1$ (resp. $d_i = k_G$). However, we moved the extraction step outside the loop and inside the `UpdateGroup` procedure. Now, the

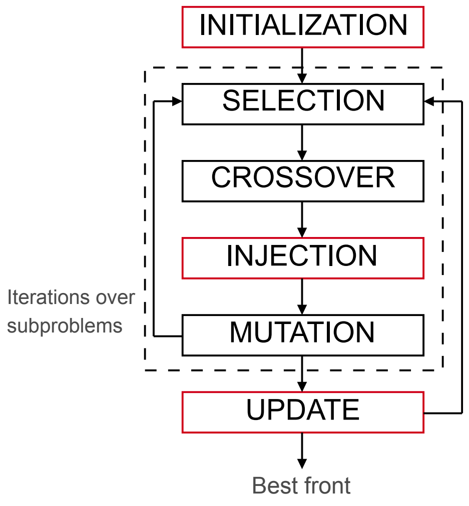


Figure 5.3: Overview of the main components of Algorithm 12. The red blocs contain elements related to the knowledge discovery mechanism.

extraction step, when applied, is performed on all solutions found during the last iteration (P'), which seems more coherent inside iterations. Indeed, now the knowledge groups are updated at the end of an iteration, and considering this possibility enriches our model since we can decide to extract the knowledge from a subset of the solutions obtained (e.g., extract from non-dominated solutions only). This difference is highlighted in Figure 5.3, showing that the update (of the groups and of the archive) is performed when the iteration over all subproblems has ended. The strategy for the extraction differs from our former hybridizations since the extraction is performed in an intensive manner ($d_e = 1$).

To evaluate the impact of the parameter k_G controlling the number of knowledge groups we decided to compare the hybridizations with different values of k_G. First of all, we consider the simplest case, where there is only one group ($k_G = 1$). In that case, the intensification is equivalent to the diversification, leading to only one variant, the so-called $\text{KD2}^1_{\text{MOEA/D}}$ algorithm. Using only one group is equivalent to considering that the problem solved is a single-objective problem. In this case, we do not consider that optimizing different objectives will give highly structurally different solutions. However, this case remains important to show that it is not possible to exactly transpose the learning strategies, that are effective in a single-objective context to a multi-objective context.

Naturally, when a bi-objective problem is optimized, the objective space contains three important regions. One region is dedicated to solutions optimizing the first objective, another region to solutions optimizing the second objective, and a third region to solutions that have an equivalent trade-off between the objectives. In this third region, the solutions can not generally be considered of good quality for either objective. In general, these solutions, which compose the middle of the optimal Pareto front are the most difficult to obtain. Therefore we propose to create $k_G = 3$ knowledge groups, each focusing on a region described above, allowing the learning of relevant patterns for each region. Here, we limit the investigation to the case where the groups are uniformly spread along the front (i.e., whose weight vectors are uniformly spread). This design leads to two variants using three groups: $\text{KD2}^{3,s_I^i}_{\text{MOEA/D}}$ (resp. $\text{KD2}^{3,s_I^d}_{\text{MOEA/D}}$), which uses the s_I^i (resp. s_I^d) strategy for the injection.

The decomposition can be refined by creating $k_G = 5$ (uniformly spread) groups in the objective space. The two additional groups focus on intermediary solutions between those optimizing one objective and those having an equivalent trade-off. Similarly, Ii leads to two other variants: $\text{KD2}^{5,s_I^i}_{\text{MOEA/D}}$ and $\text{KD2}^{5,s_I^d}_{\text{MOEA/D}}$.

Finally, we consider the extreme case where $k_G = M$, creating as many groups as subproblems like it was done in Chapter 4. In this case, each group is dedicated to one specific aggregation. More precisely, for $i \in \{1, \ldots, k_G\}$, $g^i = w^i$, where w^i is the weight vector associated with the i-th subproblem. Hence, the last two variants are $\text{KD2}^{M,s_I^i}_{\text{MOEA/D}}$ and $\text{KD2}^{M,s_I^d}_{\text{MOEA/D}}$.

5.2.4 Tuning of Parameters

Each algorithm is tuned with irace to find a good setting of the parameters. To perform the tuning, we use our set of generated instances of size 100 (see Section 2.3.4). The hybridizations contain the following parameters. M is the number of subproblems considered and m is the size of the neighborhood of each subproblem. The probabilities associated with each mechanism are p_x for the crossover, p_{inj} for the injection, and p_m for the local search. The granularity parameter δ is used to reduce the neighborhood during local search. The maximal size s_p of the patterns extracted, and the number N_i of patterns injected, chosen among the N_f most frequent patterns.

We decided to set some parameter values in order to reduce the size of the configuration space explored by irace. In the following, $m = 1/4 \times M$, $N_f = 100$, and $p_{ext} = 1.00$. The values were chosen according to our previous experiment. Furthermore, in our studies, the size of the subproblem's neighborhood and the number of most frequent patterns considered did not highly impact the performance of the algorithms. Concerning the value of the probability of the extraction, it is more a choice of design. Indeed, it seems more coherent and reliable to control the solutions from which patterns are extracted instead of applying the extraction with a probability. We do not consider the number of groups k_G in the tuning, because we want to highlight its influence on the algorithm. The range of values of the remaining parameters are presented in Table 5.1). The domain of each parameter is chosen to explore various parts of the space. Furthermore, we discretize each domain to reduce the size of the configuration space given to irace, hoping that it will produce more robust configurations. We granted a budget of 2000 runs over 8 iterations to irace. The best configurations returned for each variant are presented in Table 5.2.

We can remark that the number of subproblems is always below 60, which makes sense since small populations are often preferred in genetic algorithms. The granularity is almost always set to 25, which is coherent with existing studies in the literature on routing problems. The maximal size of patterns alternates between 5 and 7, which is close to the value recommended by Arnold et al. [2021]. Moreover, the probability of applying the local search seems low, but the local search is the most time-consuming step of the algorithm, mainly in the beginning when solutions are not optimized. With $p_m = 0.10$, the local search represents already 50% of the total running time. However, it represents only 60% when $p_m = 0.25$. The second most time-consuming step is the injection mechanism. When $p_{inj} = 1.00$, it represents around 25% of the total running time, but this mechanism requires a constant cost during the execution contrarily to the local search. Consequently, the probabilities chosen by irace try to balance the time allocated to each mechanism.

Table 5.1: Configuration space given to irace. The space contains 77175 configurations.

Parameter	Domain
Number of subproblems: M	(20, 40, 60, 80, 100)
Granularity: δ	(10, 25, 50, 75, 100)
Probability of PMX: p_x	(0.00, 0.10, 0.25, 0.50, 0.75, 0.90, 1.00)
Probability of LS: p_m	(0.00, 0.10, 0.25, 0.50, 0.75, 0.90, 1.00)
Maximum size of pattern: s_p	(5, 7, 10)
Number of patterns injected: N_i	(20, 40, 60)
Probability of injection: p_{inj}	(0.00, 0.10, 0.25, 0.50, 0.75, 0.90, 1.00)

Table 5.2: Best elite configurations returned by irace for each variant, each represented by a pair (number of groups, injection strategy).

Parameters	$(1, -)$	$(3, s_I^i)$	$(3, s_I^d)$	$(5, s_I^i)$	$(5, s_I^d)$	(M, s_I^i)	(M, s_I^d)
M	60	60	40	40	20	40	20
δ	50	25	25	25	25	25	25
p_x	0.50	0.50	0.90	0.50	0.90	0.50	0.75
p_m	0.10	0.10	0.10	0.25	0.10	0.10	0.25
s_p	5	5	7	7	5	5	5
N_i	60	20	40	60	60	40	40
p_{inj}	0.75	0.75	1.00	1.00	0.90	1.00	0.90

5.2.5 Experimental Protocol

In our experiments, we investigate how the number of groups and the strategy followed by the injection impact the quality of the solutions returned by the different variants. Each variant is executed 30 times on the 56 instances of size 100 of Solomon's benchmark. For each algorithm, the k-th run of an instance is executed with the seed $10(k-1)$, to compare the algorithms with the same seeds. The termination criterion is set to 720 seconds for all variants.

Then, for each category of instance (R, RC, and C), we compute the average uHV obtained over the 30 runs. We recall that, for computing the hypervolume the objectives are normalized by using the ideal and nadir points of the best approximation fronts obtained. These reference fronts were obtained by aggregating all the results obtained in the former chapter and the results obtained in this study. This update of the fronts may induce a variation in the hypervolumes obtained in former experiments. We rank each variant on each instance and we compute the average rank on all the categories. We perform a Friedman test on the average uHV, to know if all algorithms are equivalent, and if this is not the case, we apply a pairwise Wilcoxon test with the Bonferroni correction to know which algorithms are statistically better. We repeat the same procedure when all results are gathered in a category *All*, containing all the instances.

5.2.6 Experimental Results

Table 5.3 (resp. Table 5.4) shows the average rank (resp. uHV) of each variant on each category of instance. The detailed results for each instance are given in Appendix E. First, the hypervolumes returned remain higher, on average, than hypervolumes returned by $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$ (see Section 4.5.1), which is the reference MOEA/D considered (without the knowledge discovery mechanism). In particular, it shows that using the new construction of groups in the knowledge discovery mechanism is still beneficial.

The variant $\mathrm{KD2}^{5,s^i_I}_{\mathrm{MOEA/D}}$ always leads to the best average rank (1.46) and average uHV (0.828). Moreover, the returned results are statistically better than those returned by other variants. In particular, it shows an interest in using more than one group in this context.

Using the diversification strategy with five groups worsened a lot the returned results. More precisely, $\mathrm{KD2}^{5,s^d_I}_{\mathrm{MOEA/D}}$ ranks 5.73 on average, which is the second highest average rank. Only $\mathrm{KD2}^{3,s^i_I}_{\mathrm{MOEA/D}}$ is behind with a higher average rank of 6.32. In addition, the variant using the diversification strategy for the injection $\mathrm{KD2}^{3,s^d_I}_{\mathrm{MOEA/D}}$ has also a high rank but it is slightly lower than $\mathrm{KD2}^{5,s^d_I}_{\mathrm{MOEA/D}}$. As a result, using three groups is not adapted to the situation. Furthermore, considering the results obtained by $\mathrm{KD2}^{1}_{\mathrm{MOEA/D}}$ show that with only one group it is able to reach good results. We recall that using one group is equivalent to using the extraction with a diversification strategy. Thus, it seems that when very few groups are defined, it is better to use diversification strategies than intensification ones.

The variants $\mathrm{KD2}^{M,s^i_I}_{\mathrm{MOEA/D}}$ and $\mathrm{KD2}^{M,s^d_I}_{\mathrm{MOEA/D}}$ provide average uHV that are close in value. It is 0.767 for $\mathrm{KD2}^{M,s^i_I}_{\mathrm{MOEA/D}}$ and 0.770 for $\mathrm{KD2}^{M,s^d_I}_{\mathrm{MOEA/D}}$. The conclusion still holds when looking at each category separately. Hence, when many groups are used, there is not a significant difference between intensification and diversification strategies for the injection. It was already our conclusion in Section 4.3.

Table 5.3: Average ranks of the variants according to their average uHV over the different categories of instance. Bold results are statistically significant. Each variant is represented by a pair (number of groups, injection strategy).

Category	$(1, -)$	$(3, s_I^i)$	$(3, s_I^d)$	$(5, s_I^i)$	$(5, s_I^d)$	(M, s_I^i)	(M, s_I^d)
R	2.52	6.65	4.09	**1.26**	5.59	4.61	3.28
RC	2.16	6.94	4.72	**1.56**	5.53	3.53	3.56
C	4.09	5.29	5.50	**1.62**	6.12	2.21	3.18
All	2.89	6.32	4.70	**1.46**	5.73	3.57	3.33

Table 5.4: Average uHV of the algorithms on the different categories of Solomon's instance of size 100. Bold results are statistically significant. Each variant is represented by a pair (number of groups, injection strategy).

Category	$(1, -)$	$(3, s_I^i)$	$(3, s_I^d)$	$(5, s_I^i)$	$(5, s_I^d)$	(M, s_I^i)	(M, s_I^d)
R	0.730	0.627	0.703	**0.764**	0.667	0.682	0.706
RC	0.738	0.590	0.695	**0.781**	0.665	0.713	0.705
C	0.889	0.848	0.848	**0.959**	0.831	0.934	0.919
All	0.780	0.684	0.745	**0.828**	0.716	0.767	0.770

Finally, it is not interesting to use a too-large or a too-small number of groups. The goal is to provide a "good" intermediate value. Here, the best results are obtained with five groups, with an intensification strategy for the injection and the extraction. Moreover, using a diversification strategy for the injection does not seem interesting when considering a high (≥ 5) number of groups, while the diversification strategy for the extraction tends to artificially reduce the number of groups.

To conclude this section, we proposed a new construction of knowledge groups, independent from MOEA/D and allowing the use of any number of groups in a bi-objective context. Moreover, we formalized the strategies that extraction and injection can follow, and we instantiated them to obtain an intensification and a diversification strategy. We integrated our propositions into a MOEA/D framework, and we tested them on a bVRPTW. Briefly, the results showed that the variant using five knowledge groups with an intensification strategy for both the injection and extraction was statistically better than the others. In the next section, we push further the analysis of the components of our hybridization, by studying the impact of the values of the different parameters (with the number of groups), when intensification is the strategy used for extraction and injection.

5.3 Parameters Analysis of the KD MOEA/D

In the former section, we presented a new construction for the knowledge groups, as well as strategies for the injection and extraction mechanisms. To better understand the synergy between the different components of the hybridization developed, we propose to conduct a deep analysis of the parameters used.

The study considers a version of the original MOEA/D, provided in Algorithm 9, where the mutation is a local search. Here, the strategies of the local search (the exploration strategy and the metric defining the neighborhood of a customer) are tuned by irace too. Concerning the hybridization, the Algorithm 12 is used. In addition to the parameters already tuned in the former section, we consider the strategies for the local search and the number of knowledge groups created. However, the strategies of extraction and injection are set to $d_e = 1$ and $d_i = 1$ according to the results obtained in the former section. Indeed, by fixing the strategies we reduce the size of the configuration space, and besides, these parameters seem highly correlated with the number of groups constructed. Moreover, by considering the most intensive strategy, we hope to obtain more easily explainable results. More precisely, when multiple groups are considered during the extraction and injection steps, the algorithm may have a more chaotic behavior (i.e., it is harder to understand from where the knowledge comes and where it is exploited the best).

As before, the tuning is supported by irace. We start with a preliminary tuning in Section 5.3.1, where the configuration space defined is huge. The configurations obtained are studied in Section 5.3.2 to know if they are local optima configurations for the algorithms. Moreover, the different parameters are analyzed in parallel to examine their influence on the algorithms. Following that study, the configuration space is reduced to refine the search, and a final tuning is performed. Finally, the performances of the tuned algorithms are compared in Section 5.3.3, where experiments are conducted on Solomon's benchmark with instances of size 100, and on Gehring and Homberger's benchmark with instances of size 200.

During this study, to compute the hypervolumes, we updated the ideal and nadir points from our former experiments to normalize correctly the objectives of the solutions found.

5.3.1 Preliminary Tuning

Two main algorithms are tuned: the MOEA/D with a local search, indicated with $\text{R}_{\text{MOEA/D}}$, and the hybrid MOEA/D with the fixed strategies for the extraction and injection steps, indicated with $\text{KD2}_{\text{MOEA/D}}$. The tuning of $\text{R}_{\text{MOEA/D}}$ allows seeing which neighborhood exploration strategy and metric are better when not performing learning steps. In addition, six variants of $\text{KD2}_{\text{MOEA/D}}$ are tuned and compared. Each variant is characterized by a pair (exploration strategy, granularity metric) defining the local search performed. We recall that (see Section 4.4) there are two exploration strategies (*best* and *first-best*) and two metrics (d_1 and d_2). Here, we consider in addition a third metric d_3 where the weights in d_2 are set to 0.5. This metric may be particularly interesting when other algorithms than MOEA/D are considered, which avoids the definition of weight vectors. Hence, it leads to the creation of the six following variants $\text{KD2}^{fb,d_1}_{\text{MOEA/D}}$, $\text{KD2}^{fb,d_2}_{\text{MOEA/D}}$, $\text{KD2}^{fb,d_3}_{\text{MOEA/D}}$, $\text{KD2}^{b,d_1}_{\text{MOEA/D}}$, $\text{KD2}^{b,d_2}_{\text{MOEA/D}}$, $\text{KD2}^{b,d_3}_{\text{MOEA/D}}$, which are respectively shortened fbd_1, fbd_2, fbd_3, bd_1, bd_2, bd_3. We had two main motivations for considering all these variants. First, by tuning $\text{KD2}_{\text{MOEA/D}}$, we obtain one of the six additional variants, which allows us to compare the configuration obtained for $\text{KD2}_{\text{MOEA/D}}$ and the configuration obtained for the corresponding additional variant. If the two configurations are similar, it assesses the robustness of irace when more parameters are considered. The second motivation is to compare more precisely the impact of the different strategies used for the local search (similarly to what has been done in Section 4.4). In addition, all the returned configurations provide a set of good configurations that can be used as a starting point to analyze the influence of the different parameters.

In Section 5.3.1.1 we present the configuration space defined in irace (i.e., the parameters tuned and their associated domain). The results of the tuning are discussed in Section 5.3.1.2.

Table 5.5: Configuration space given to irace. The granularity and the number of patterns injected are expressed as a percentage of the size of the instance. The parameter representing the number of groups is defined as a percentage of the number of subproblems (when this parameter is equal to 1, only 1 group is created, since 0 groups is not a valid value). The space contains 388 800 configurations.

Parameter	Domain
Number of subproblems: M	(20, 40, 60, 80, 100)
Granularity: δ	(10%, 25%, 50%, 75%, 100%)
Probability of PMX: p_x	(0.00, 0.10, 0.25, 0.50, 0.75, 1.00)
Probability of LS: p_m	(0.00, 0.10, 0.25, 0.50, 0.75, 1.00)
Exploration strategy: $\mathcal{S}$	{*best*, *first-best*}
Granularity metric: d	$\{d_1, d_2, d_3\}$
Number of groups: k_G	(1, 10%, 25%, 50%, 75%, 100%)
Maximum size of pattern: s_p	(2, 5, 8)
Number of patterns injected: N_i	(25%, 50%, 75%, 100%)
Probability of injection: p_{inj}	(0.00, 0.10, 0.25, 0.50, 0.75, 1.00)

5.3.1.1 Configuration Space and Protocol

The reference MOEA/D ($\mathrm{R}_{\mathrm{MOEA/D}}$) uses the following parameters: M, the number of subproblems considered, and m the size of the neighborhood of each subproblem. The probabilities associated with each mechanism are p_x for the crossover and p_m for the local search. The granularity parameter δ is used to reduce the neighborhood during local search. The choice of the exploration strategy $\mathcal{S}$ (either *best* or *first-best*) and the granularity metric d (either d_1, d_2, or d_3) are left to the tuning. The hybrid MOEA/D ($\mathrm{KD2}_{\mathrm{MOEA/D}}$) has, in addition, the following parameters: the number of knowledge groups created k_G, the probability of injection p_{inj}, the maximal size s_p of the patterns extracted, and the number N_i of patterns injected, chosen among the N_f most frequent patterns. According to a preliminary study [Legrand et al., 2023a] presented in Section 5.2 and existing works [Arnold et al., 2021], we set $m = 1/4 \times M$ and $N_f = 5 \times N_i$. The domain of each remaining parameter is presented in Table 5.5 to define the configuration space in irace. The six derived variants from $\mathrm{KD2}_{\mathrm{MOEA/D}}$ have the same parameters, except that the exploration strategy $\mathcal{S}$ and the granularity metric d are already set.

The first tuning is performed on all the variants, with the whole configuration space defined above. A budget of 2000 runs is granted to irace, spread between 8 iterations. Among all the 96 instances generated, we kept only a subset of 20 instances (of size 100) per category (R and C) to perform the tuning. Indeed, we remarked during the previous tuning that irace used approximately 20 instances with the budget provided. As a result, to guarantee that all training instances are used, we decide to consider a subset of the generated instances. Moreover, we separate the tuning of the algorithms on instances R and C, since their structure highly differ. Hopefully, this separation allows us to obtain a more appropriate tuning for all instances, leading to better performances of the algorithms.

5.3.1.2 Preliminary Configurations and Discussion

The tuning performed by irace on C (resp. R) instances led to the elite configurations stored in Table 5.6 (resp. Table 5.7). Only the best elite configuration for each variant is reported (i.e., the configuration of rank 1). The reference algorithm, $\mathrm{R}_{\mathrm{MOEA/D}}$, seems more efficient with the *first-best* exploration strategy on both categories of instances. The same conclusion is reached for the hybrid one ($\mathrm{KD2}_{\mathrm{MOEA/D}}$). Moreover, the metric d_3 seems more appropriate to C instances, and d_2 to R instances, according to the results obtained for $\mathrm{R}_{\mathrm{MOEA/D}}$ and $\mathrm{KD2}_{\mathrm{MOEA/D}}$. In fact, C instances are, in general, more simple to solve. In this situation, the closeness between customers is structurally integrated into the instance, thus a simplified metric, that is d_3, where the waiting time between customers discriminates easily close customers, seems enough to obtain good results. The metric d_2 is more relevant for the less structured R instances since the weights between the distance and the waiting time vary according to the aggregation of the subproblem solved. Hence, d_2 discriminates customers according to each aggregated objective. Note that, in neither case the metric d_1 is chosen, meaning that d_1 does not seem adapted to solve these instances in a bi-objective context, as expected.

Concerning the other parameters, we remark that the local search is, in most cases, applied with probability 0.10, which is relatively low, but it is a costly step in terms of running time. In addition, the granularity remains high (with a value of the 50% or 75% closest customers considered during the search), and the choice of the metric does not seem to have a great impact on it (see the values obtained for the six sub-variants). Moreover, the local search is much more impacting at the beginning of the algorithm, when solutions are bad, thus it may not be interesting to use it too frequently. On the other hand, the injection is applied with a much higher probability (either 0.75 or 1.00). Indeed, the injection operator needs fewer resources and is useful throughout the execution of the algorithms. During the injection, at least 50 patterns are tentatively injected, and the maximal size of the extracted patterns is almost always set to 5 (surprisingly, it was set to 8 only for the fbd_2 variant on C instances). The value of 5 was already the value chosen by the authors of PILS [Arnold et al., 2021]. The crossover operator is applied with a probability less than or equal to 0.50 when using the knowledge discovery mechanism, i.e., in all the hybrid variants ($\mathrm{KD2}_{\mathrm{MOEA/D}}$, fbd_1, fbd_2, fbd_3, bd_1, bd_2, bd_3), whereas it is applied in every iteration with $\mathrm{R}_{\mathrm{MOEA/D}}$. Indeed, in $\mathrm{R}_{\mathrm{MOEA/D}}$, the crossover is the only operator that introduces diversity in the solutions, necessary to escape local optima. It is not the case with the other variants, due to the presence of the injection operator. The number of aggregations used is coherent with the values used in general (around 40 aggregations). Only $\mathrm{R}_{\mathrm{MOEA/D}}$ requires 80 aggregations on R instances to generate better and more diverse solutions. Finally, the number of knowledge groups used is very variable, but at least four groups are always created for each variant. Furthermore, in C instances fewer groups are created than in R instances. To be more precise, at most 20 groups are created when solving clustered instances, whereas it rises to 60 groups for random instances. Our main hypothesis concerning the use of many groups is that the structure between two solutions (in terms of patterns extracted) in the objective space may vary significantly, even between solutions with *close* objective values. This can be verified in Figure 4.1 for Solomon's instances. By considering the two first columns (regions "r0" and "r1"), which represent good quality solutions, we can see a bigger dispersion of the average similarity of a solution in R instances than in C instances. Moreover, the shape of the optimal Pareto front may impact as well the number of groups created. More precisely, we already remarked that Pareto fronts of C instances contain in general fewer solutions than Pareto fronts of R instances. As a consequence, many groups are created to reduce the *structural gap* between solutions in the same group.

Table 5.6: Best elite configurations returned by irace when tuning on C instances. The symbol "-" reflects the absence of the parameter during the tuning.

Parameter	$R_{\text{MOEA/D}}$	$KD2_{\text{MOEA/D}}$	fbd_1	fbd_2	fbd_3	bd_1	bd_2	bd_3
M	40	40	20	40	40	20	40	40
δ (%)	50	75	50	75	50	75	50	50
p_x	1.00	0.50	0.50	0.25	0.25	0.50	0.10	0.25
p_m	0.10	0.10	0.10	0.10	0.10	0.10	0.25	0.10
$\mathcal{S}$	first-best	first-best	-	-	-	-	-	-
d	d_3	d_3	-	-	-	-	-	-
k_G (%)	-	50	25	10	25	100	10	50
s_p	-	5	5	8	5	5	5	5
N_i (%)	-	100	50	50	75	50	50	75
p_{inj}	-	0.75	1.00	1.00	1.00	1.00	1.00	1.00

Table 5.7: Best elite configurations returned by irace when tuning on R instances. The symbol "-" reflects the absence of the parameter during the tuning.

Parameter	$R_{\text{MOEA/D}}$	$KD2_{\text{MOEA/D}}$	fbd_1	fbd_2	fbd_3	bd_1	bd_2	bd_3
M	80	40	20	60	40	40	40	20
δ (%)	75	75	50	75	75	50	50	50
p_x	1.00	0.50	0.50	0.50	0.25	0.50	0.25	0.50
p_m	0.10	0.10	0.25	0.10	0.10	0.10	0.10	0.10
$\mathcal{S}$	first-best	first-best	-	-	-	-	-	-
d	d_2	d_2	-	-	-	-	-	-
k_G (%)	-	100	25	100	75	100	10	100
s_p	-	5	5	5	5	5	5	5
N_i (%)	-	100	75	75	75	75	75	75
p_{inj}	-	0.75	1.00	1.00	0.75	0.75	0.75	0.75

More generally, we observe some differences between the tuning performed for C instances and the tuning performed for R instances. For example, the probability of injection tends to be higher for C instances, while the number of injected patterns tends to be higher for R instances. The results obtained validate our choice to tune separately the algorithms on instances R and C. Moreover, many configurations are very similar when considering the same category of instance. Indeed most of them only differ by two or three parameters and use close parameter values.

In the following, we investigate the influence of the parameters on a subset of elite configurations, in order to see if the configurations returned are truly local optima in the configuration space provided, and if the configurations can be improved by reducing the size of the configuration space.

5.3.2 Configuration Analysis and Influence of Parameters

In this section, we analyze the elite configurations returned for the hybrid variants. Each configuration is a local optimum for irace and thus should be the best configuration (statistically) in its close neighborhood. For each configuration, we generate all the possible configurations distant by one (using the hamming distance). More precisely, only one parameter is changed at a time (the possible values are the values available in Table 5.5), while keeping the values of the other parameters unchanged. In order to check whether the configurations returned by irace are good or not, we evaluate the generated configurations on the tuning instances (same as before), with the same seed to be fair with the methodology of irace. Then, the average hypervolume obtained is computed and compared.

Given an elite configuration from Table 5.6 (resp. Table 5.7), the set of its neighbors obtained by modifying one parameter (e.g., the number of aggregations) is evaluated on the 20 generated instances of category C (resp. R). The same seed is used for all the evaluations, which is similar to a run performed by irace. The results are reported in Figure 5.4 (resp. Figure 5.5). The (blue) boxplot on the row *Initial* represents the performances of the elite configuration as returned by irace. In other words, it is the reference boxplot. Each boxplot on the rows below represents the performance of the configurations obtained when the corresponding parameter is modified. We remark that only a few parameters have a noticeable impact on the configuration. The most impacting parameters are the exploration strategy ($\mathcal{S}$) used during the local search, the probability of applying the PMX (i.e., the crossover), and the probability of applying the injection.

The strategy only takes two values, and according to the results obtained during the tuning, it seems coherent that changing the *best* strategy to the *first-best* strategy improves the results obtained, while the contrary deteriorates the results.

The results obtained, when the probability of applying the local search varies, are shown in Figure 5.6. For each possible value of the parameter, the hypervolumes obtained with the corresponding configuration are represented. If there is nothing noticeable with the exploration strategy *best*, with the *first-best* exploration strategy it seems better to consider lower values (between 0.10 and 0.25). The results obtained with the other parameters are provided in Appendix F.

Similarly, the uHV tends to increase when the probability of PMX increases until a plateau is reached after the value of 0.75 (see Figure F.3). The influence of the crossover is even more important with the *best* strategy.

Concerning the probability of injection, the average uHV increases, along with the probability. In particular, when the injection is disabled (probability 0.0), we retrieve the fact that *first-best* strategy performs better than *best* strategy when no learning is used. Moreover, the injection should be applied with high probability (above 0.75) in general (see Figure F.10). For the other parameters, there are almost no variations of the uHV when the value of the parameter changes. In theory, it means that we could choose any possible value for the remaining parameters, without changing the uHV too much. Moreover, it also explains why the set of elite configurations returned by irace for one variant, contains in general very different values for those parameters. In particular, considering the number of groups (see Figure F.7), no value seems more interesting than another one to obtain good results. It is very dependent on the other parameters (and in particular, the number of subproblems considered). We think, that this parameter should also depend on the instance considered since the Pareto fronts between two instances can be very different.

The next section focuses on the reduction of the configuration space initially proposed, following the results presented above. Then, a new tuning is performed for the six sub-variants of $\text{KD2}_{\text{MOEA/D}}$. Finally, an experimental study is performed to compare the six variants to $\text{R}_{\text{MOEA/D}}$.

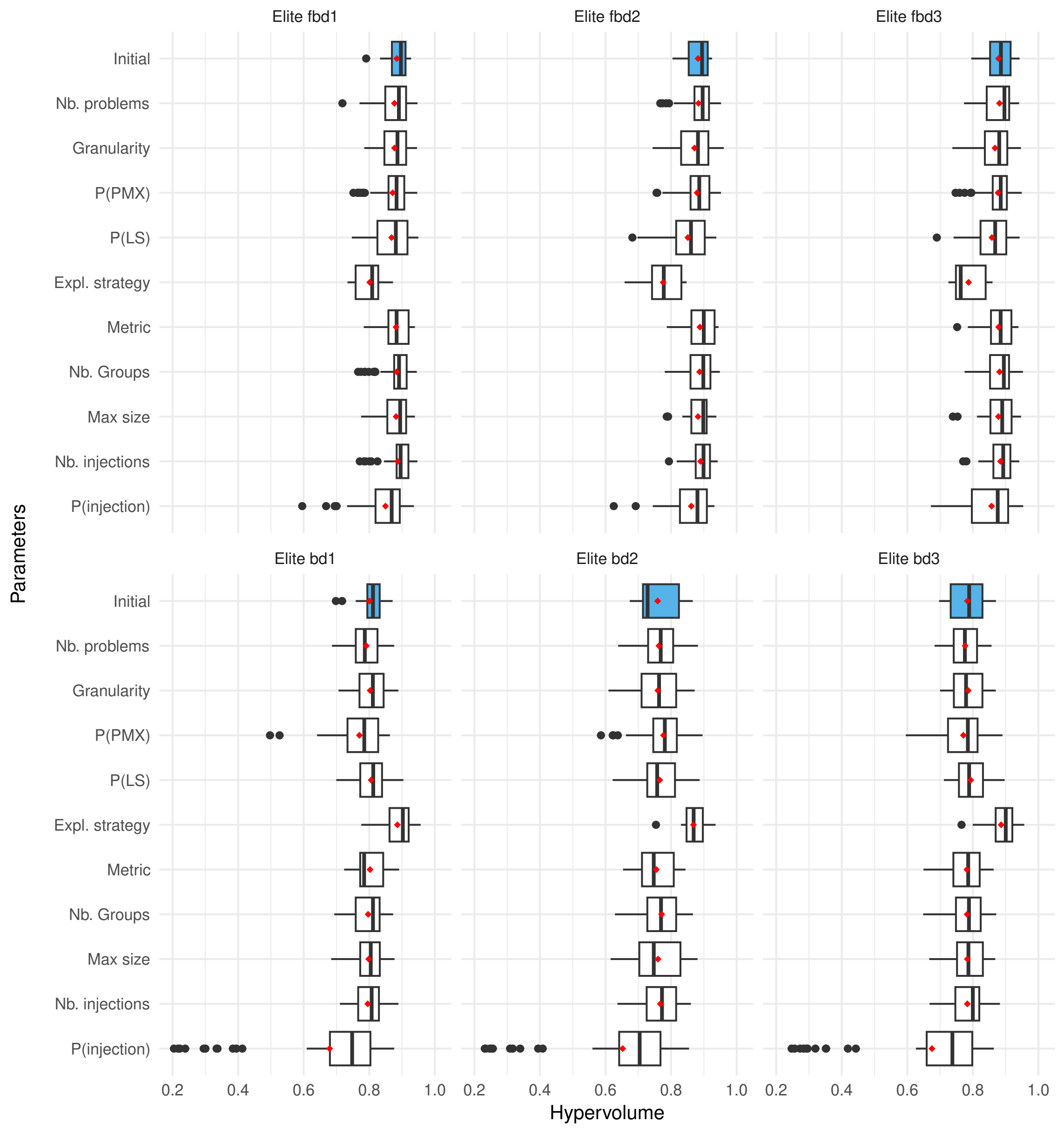


Figure 5.4: Performances of elite configurations obtained on C instances, when one parameter is modified at a time. The (blue) boxplot refers to the elite configuration as returned by irace.

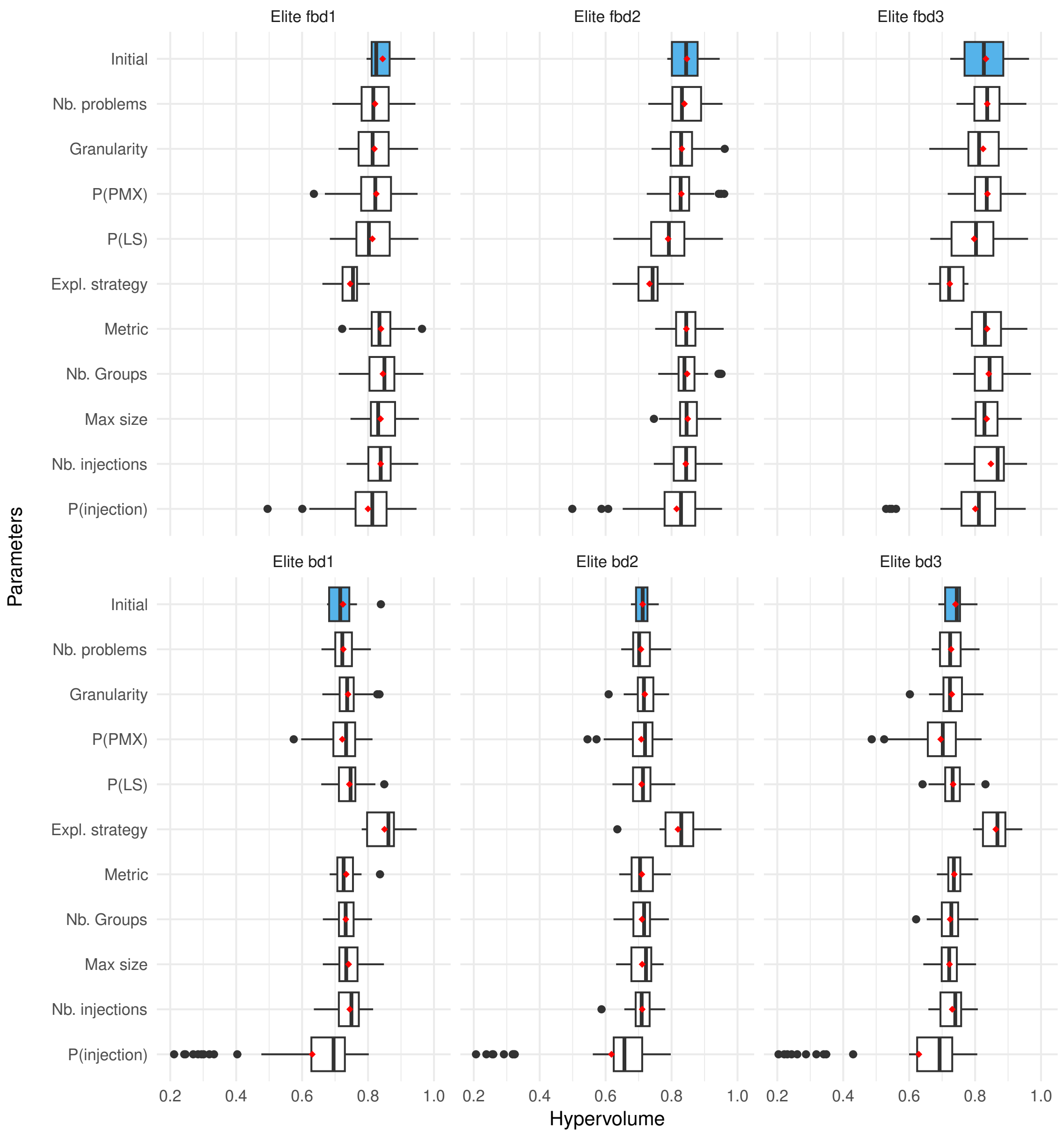
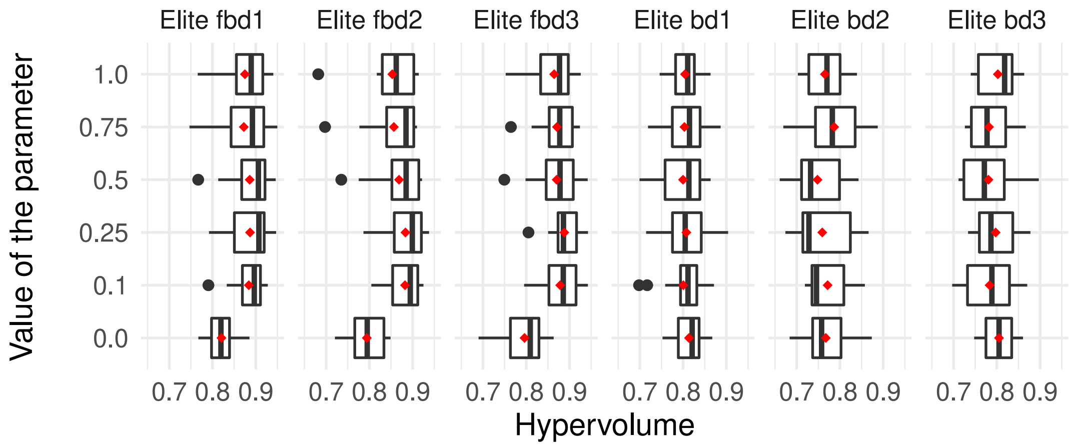
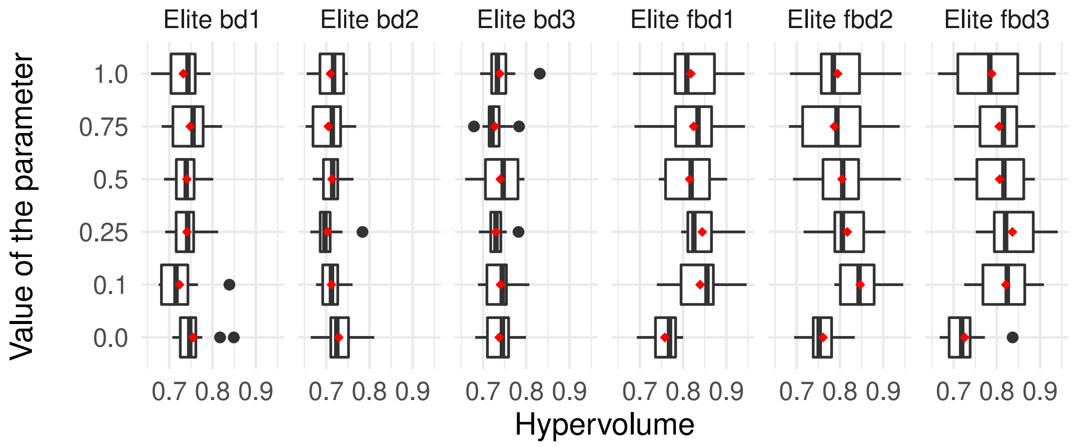


Figure 5.5: Performances of elite configurations obtained on R instances, when one parameter is modified at a time. The (blue) boxplot refers to the elite configuration as returned by irace.



(a) Category C.



(b) Category R.

Figure 5.6: Influence of the probability of the local search on instances of (a) category C and (b) category R. The red dot represents the mean uHV.

Table 5.8: Reduced configuration space. The parameters with a modified domain are in blue. The space now contains 1 458 different configurations.

Parameter	Domain
Number of subproblems: M	(40)
Granularity: δ	(25%, 50%, 75%)
Probability of PMX: p_x	(0.25, 0.50, 0.75)
Probability of LS: p_m	(0.10, 0.20, 0.30)
Number of groups: k_G	(1, 10%, 25%, 50%, 75%, 100%)
Maximum size of pattern: s_p	(2, 5, 8)
Number of patterns injected: N_i	(100%)
Probability of injection: p_{inj}	(0.50, 0.75, 1.00)

5.3.3 Comparison of the Tuned Variants

In this section, we use the results obtained in the former section to adapt and reduce the size of the configuration space (Section 5.3.3.1). By reducing the size of the search space, irace should return more accurate results. We perform a new tuning of the six variants fbd_1, fbd_2, fbd_3, bd_1, bd_2, and bd_3 considering the reduced space. The training instances are those used for the first tuning. The configurations returned are discussed in Section 5.3.3.2. Finally, the comparison of the performances of the algorithms is presented in Section 5.3.3.3.

5.3.3.1 Reduction of the Configuration Space

As explained in the former section, some parameters do not have much influence on the performance of the procedure. That is why we decided to set the number of patterns injected to 100% of the size of the instance, i.e., 100 patterns are tentatively injected (we recall that a pattern is truly injected only in case of improvement) on instances of size 100. Note that, it was the value chosen by Arnold et al. [2021] in their own experiments. Moreover, we decided to set the number of aggregations to 40, to see if the number of groups used can be more precisely tuned. In addition, by setting this parameter, it is easier to see how the number of groups impacts the performances of the algorithms. According to what has been said in the previous section, we reduce the range of values for the probability of PMX to $(0.25, 0.50, 0.75)$, and for the probability of injection to $(0.50, 0.75, 1.00)$, being the most interesting values. Concerning the probability of LS, we changed the range to $(0.10, 0.20, 0.30)$, in order to give more adapted parameters to irace. Finally, the granularity is also reduced to $(25, 50, 75)$, since extreme values never belong to elite configurations. All the changes are reported in Table 5.8.

5.3.3.2 Final Configurations

Using the reduced space of configuration, we tuned accordingly the six variants: fbd_1, fbd_2, fbd_3, bd_1, bd_2, and bd_3. Indeed, $\mathrm{R}_{\mathrm{MOEA/D}}$ already had a small configuration space (it has fewer parameters to tune), and the elite configurations returned for $\mathrm{KD2}_{\mathrm{MOEA/D}}$ were already in the reduced configuration space defined. The budget allocated to irace is reduced from 2000 to 1000. The best elite configurations are reported in Table 5.9.

Table 5.9: Best elite configurations returned by irace on instances of (a) category C, and (b) category R, by exploring the reduced configuration space.

(a) Category C.

Param.	fbd_1	fbd_2	fbd_3	bd_1	bd_2	bd_3
δ (%)	50	50	50	50	50	50
p_x	0.50	0.50	0.50	0.50	0.50	0.50
p_m	0.10	0.10	0.10	0.10	0.10	0.10
k_G (%)	50	50	100	100	100	100
s_p	5	5	5	5	2	2
p_{inj}	0.75	1.00	1.00	1.00	0.75	0.75

(b) Category R.

fbd_1	fbd_2	fbd_3	bd_1	bd_2	bd_3
75	75	75	50	50	25
0.50	0.50	0.50	0.50	0.50	0.50
0.10	0.20	0.10	0.10	0.10	0.10
50	25	50	100	100	50
8	8	8	8	8	2
1.00	1.00	0.75	0.75	1.00	0.75

At first, we notice that the differences between the two categories of instance are more important, considering the granularity parameter (δ), and the maximum size of patterns extracted (s_p). Indeed, the granularity is always set to 50 on C instances, which seems coherent, since these instances contain clusters of customers. The granularity tends to be slightly higher on R instances, except for the variant bd_3, which has the lowest granularity (among all tuned variants). The maximum size of the patterns is now 8 for most of the variants on R instances, meaning that bigger patterns are interesting when instances are not well structured. For C instances, the maximum size of the pattern remains coherent, since smaller patterns are enough to perform interesting moves.

Another major change concerns the number of knowledge groups used during the execution of the algorithms. With the variant being set, more groups are generated on C instances than on R instances. In addition, more groups are generated with strategy *best* than with strategy *first-best*.

Concerning the other parameters, the probability of PMX is set to 0.50 for all variants, the probability of LS is set to 0.10 (except for the variant fbd_2 where it is 0.20), and the probability of injection remains similar to what we saw during the first tuning (see Table 5.6 and Table 5.7).

5.3.3.3 Experimental Results

From now, $\mathrm{R}_{\mathrm{MOEA/D}}$ is used with the configurations presented in Table 5.6 and Table 5.7, and the variants fbd_1, fbd_2, fbd_3, bd_1, bd_2, and bd_3 with the configurations in Table 5.9.

The first experiment is performed on Solomon's instances of size 100 (categories R and C). Each algorithm is executed 30 times on the instances. For each algorithm, the k-th run of an instance is executed with the same seed, being $10(k-1)$, allowing a fair comparison. We recall that the termination criterion is set to 720 seconds (i.e., 12 minutes) for all algorithms. For each category of instance (either R or C), we compute the average uHV obtained over the 30 runs. We perform a Friedman test on the average uHV, to know if all algorithms are equivalent, and if this is not the case, we apply a pairwise Wilcoxon test with the Bonferroni correction to determine the best algorithms. A second experiment is performed on Gehring and Homberger's instances of size 200 (on categories R and C too). We keep the same configurations as before (parameters are scaled accordingly, in particular, the granularity and the number of patterns injected). Again, we perform 30 runs on each instance, and each algorithm is executed with the same seeds. The termination criterion is now set to 2880 seconds (i.e., 48 minutes). We perform the same statistical tests to compare the average uHV obtained.

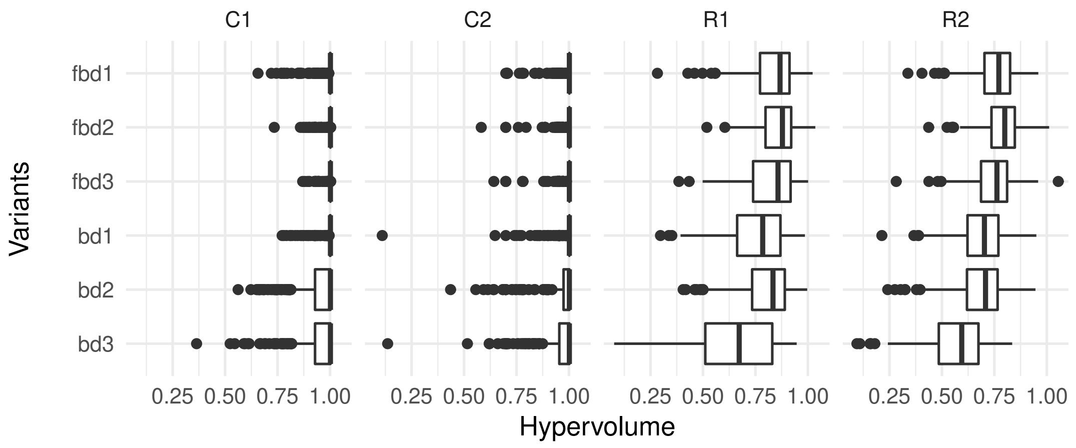
The average uHV obtained through the 30 runs, for each category of instances, is shown in

Figure 5.7. The results obtained with the *first-best* strategy are slightly (resp. significantly) more robust, with smaller boxplots, than those obtained with the *best* one on Solomon's benchmark (resp. Gehring and Homberger's benchmark). The results obtained on Solomon's benchmarks show that the variant fbd_2 is slightly better on average than the others. However, in Gehring and Homberger's benchmark, we clearly see that variants using the *first-best* strategy outperform the variants using the *best* strategy. Moreover, when the exploration strategy is set, there is almost no change when the granularity metric is modified. One hypothesis should be that the given values for the granularity remain too high.

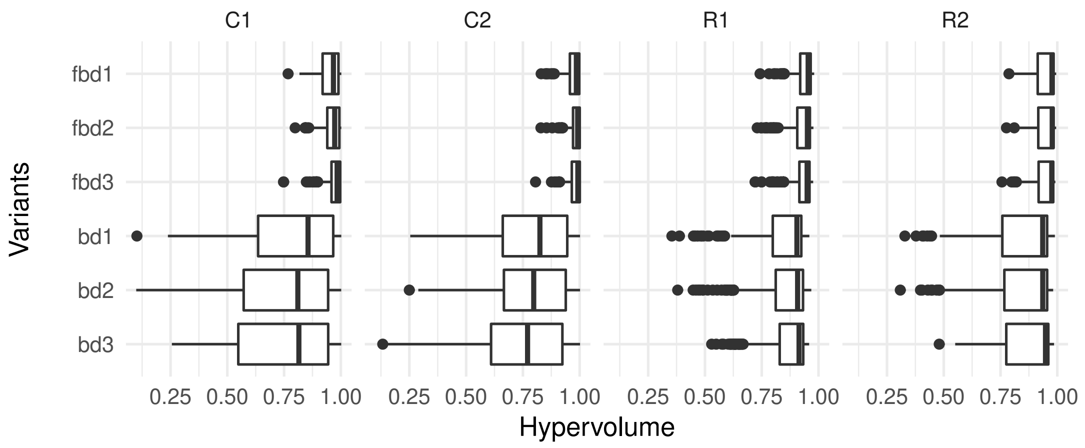
Table 5.10 contains the average ranking of the different algorithms on all classes of instances. In this table, the ranks of statistically best algorithms on a class of instances are put in bold. In particular, considering Solomon's benchmark, the variants fbd_2 and fbd_3 are statistically equivalent on $C1$ instances and are better than the other variants. On $C2$ instances, the variants fbd_1, fbd_2, fbd_3, and bd_2 are equivalent. It is not surprising since $C2$ instances are the easiest ones. Overall, the variant $\mathrm{KD2}_{\mathrm{MOEA/D}}^{fb,d_3}$ returns good results, explaining why this combination of exploration strategy and granularity metric has been returned by irace in Table 5.6. On R instances, statistically, the best overall algorithm is fbd_2. However, fbd_1 is equivalent to fbd_2 on R1 instances. As a consequence, the strategy *first-best* with the metric d_2, seems to be the best combination, considering all the classes of instances of size 100. This result has already been observed since these parameters were selected for the configuration returned by irace for $\mathrm{KD2}_{\mathrm{MOEA/D}}$ (see Table 5.7). In addition, all the hybrid MOEA/Ds are better than the reference $\mathrm{R}_{\mathrm{MOEA/D}}$, which has very low uHV. In fact, $\mathrm{R}_{\mathrm{MOEA/D}}$ gets easily stuck in local optima, and particularly when the Pareto fronts do not contain many non-dominates solutions. Table 5.11 and Table 5.12 show more detailed results about the convergence of the algorithms fbd_2, bd_2, and $\mathrm{R}_{\mathrm{MOEA/D}}$. We can remark that, as expected, the final uHV obtained by fbd_2 is higher than the one obtained by bd_2, and $\mathrm{R}_{\mathrm{MOEA/D}}$. Moreover, fbd_2 is able to reach 80% of the uHV of the reference front significantly faster than $\mathrm{R}_{\mathrm{MOEA/D}}$ and bd_2 on many instances. When the algorithm does not reach 80% of the uHV at the end of the execution, the maximal time allocated is used instead. It shows that using the learning mechanism with the *first-best* strategy speeds up the convergence process (recalling the conclusion made in Section 4.5).

We focus now on Gehring and Homberger's benchmark. Considering the Table 5.10, the variant fbd_3 is statistically better on instances C1 (with tight time windows), while the variants fbd_2 and fbd_3 are statistically equivalent on instances C2 (with wide time windows). Moreover, these two algorithms remain better than all the others. On $R2$ instances, the three variants fbd_1, fbd_2, and fbd_3 are statistically equivalent and the results returned by these three variants are very close. However, it is the variant fbd_1 that is statistically better on $R1$ instances. It is important to notice that with a specific tuning on instances of size 200, we may have reached slightly different conclusions. That being said, strategy *first-best* is always better than strategy *best*. Concerning the metrics, we finally observe that they have a relatively low impact on instances of a bigger size. We meet again the result obtained in Section 5.3.2, where the granularity metrics did not seem to have an impact on the elite configurations. The source code, reference fronts, and additional convergence results are publicly available[1]. The results associated with this benchmark are given in Appendix F.

[1] https://github.com/Clegrandlixon/data_itor2023



(a) Solomon's benchmark.



(b) Gehring and Homberger's benchmark.

Figure 5.7: Results obtained on the different categories of instances, of (a) Solomon's benchmark for size 100, and (b) Gehring and Homberger's benchmark for size 200.

Table 5.10: Average ranking of all the algorithms on the different categories of instances. Bold ranks are statistically equivalent.

Size	100				200			
Category	C		R		C		R	
Class	C1	C2	R1	R2	C1	C2	R1	R2
$\text{R}_{\text{MOEA/D}}$	7.0	7.0	6.8	5.3	7.0	4.8	6.8	4.0
fbd_1	3.7	**2.4**	**2.3**	2.3	3.0	2.9	**1.1**	**1.7**
fbd_2	**2.3**	**2.3**	**1.4**	**1.0**	1.9	**1.4**	2.6	**1.7**
fbd_3	**2.2**	**2.9**	2.5	2.7	**1.1**	**1.7**	2.4	**2.6**
bd_1	3.6	**3.3**	5.1	4.7	4.0	5.3	6.1	6.5
bd_2	4.9	4.2	3.9	5.0	5.3	5.6	5.0	6.5
bd_3	4.4	5.9	6.1	7.0	5.7	6.5	4.3	5.0

Table 5.11: Detailed results for algorithms fbd_2, bd_2, and $\text{R}_{\text{MOEA/D}}$ on Solomon's C instances of size 100. From left to right: the average size of the front, the average uHV, and the average time to reach 80% of the reference uHV.

	fbd_2			bd_2			$\text{R}_{\text{MOEA/D}}$		
Inst.	$\|F\|$	uHV	Time (s)	$\|F\|$	uHV	Time (s)	$\|F\|$	uHV	Time (s)
C101	1.0	**1.002**	**57.3**	1.0	**1.002**	197.1	1.8	0.354	682.7
C102	1.1	**0.988**	**120.7**	1.0	**0.980**	362.6	3.2	0.033	720.3
C103	1.5	**0.984**	**211.6**	2.0	0.712	614.6	5.3	0.000	720.4
C104	1.7	**0.905**	**316.4**	2.8	0.588	702.9	4.5	0.000	720.4
C105	1.0	**0.989**	**111.0**	1.0	**0.989**	248.7	1.9	0.068	715.2
C106	1.0	**1.002**	**69.0**	1.0	**1.002**	255.5	1.9	0.208	720.4
C107	1.0	**0.972**	**140.2**	1.0	**0.962**	324.6	1.5	0.013	720.2
C108	1.0	**0.971**	**158.4**	1.0	0.925	424.8	1.4	0.001	720.3
C109	1.0	**0.944**	**191.5**	1.0	0.802	505.5	1.5	0.000	720.3
C201	1.0	**1.002**	**33.8**	1.0	**1.002**	171.0	1.2	0.948	205.0
C202	1.0	0.994	**97.4**	1.0	**1.002**	308.1	1.0	0.743	573.3
C203	1.0	**0.937**	**106.4**	1.1	0.890	404.0	1.1	0.704	529.1
C204	1.0	**0.849**	**333.4**	1.4	0.664	658.9	1.5	0.450	657.7
C205	1.0	**0.990**	**47.2**	1.0	0.973	259.8	1.1	0.865	345.0
C206	1.0	**1.002**	**55.6**	1.0	0.978	372.1	1.0	0.821	481.5
C207	1.0	**1.002**	**49.5**	1.0	0.970	332.0	1.1	0.749	487.9
C208	1.0	**1.002**	**56.3**	1.1	0.987	371.7	1.3	0.751	507.6

Table 5.12: Detailed results for algorithms fbd_2, bd_2, and $\mathrm{R}_{\mathrm{MOEA/D}}$ on Solomon's R instances of size 100. From left to right: the average size of the front, the average uHV, and the average time to reach 80% of the reference uHV.

	fbd_2			bd_2			$\mathrm{R}_{\mathrm{MOEA/D}}$		
Inst.	$\lvert F\rvert$	uHV	Time (s)	$\lvert F\rvert$	uHV	Time (s)	$\lvert F\rvert$	uHV	Time (s)
R101	72.2	**0.915**	**65.0**	70.6	**0.911**	229.7	47.7	0.907	73.4
R102	45.6	**0.923**	**137.8**	45.0	**0.911**	296.8	27.1	0.876	232.7
R103	23.7	**0.937**	**277.8**	23.4	0.899	457.0	20.3	0.724	721.1
R104	6.0	**0.915**	**341.2**	6.3	0.881	533.4	6.4	0.499	721.2
R105	12.3	**0.941**	**166.2**	13.9	0.906	299.6	7.7	0.862	438.6
R106	6.2	**0.894**	**258.8**	8.0	**0.878**	406.5	5.5	0.763	700.1
R107	6.0	**0.851**	**496.2**	5.1	0.828	602.9	6.8	0.524	720.9
R108	1.8	**0.880**	**358.2**	1.5	0.851	532.8	2.5	0.389	720.9
R109	1.2	**0.915**	**259.5**	1.4	0.877	450.8	1.5	0.727	680.3
R110	1.1	**0.843**	**368.9**	1.3	**0.849**	496.6	1.3	0.637	717.1
R111	1.9	**0.850**	**423.1**	1.9	0.809	583.4	2.4	0.517	720.7
R112	1.0	**0.845**	**480.2**	1.0	0.804	559.9	1.0	0.387	720.4
R201	52.1	**0.854**	**230.3**	52.2	**0.843**	376.4	33.6	0.814	525.9
R202	44.7	**0.857**	**311.6**	42.1	**0.849**	457.3	25.8	0.824	515.1
R203	34.5	**0.859**	**338.4**	31.2	0.819	600.0	21.3	0.797	613.9
R204	13.2	**0.806**	**548.7**	11.9	0.752	679.7	8.1	0.719	706.6
R205	20.6	**0.839**	**413.7**	22.3	**0.833**	509.6	14.4	0.798	637.2
R206	22.5	**0.867**	**349.4**	22.9	0.839	534.9	13.0	0.821	581.8
R207	19.4	**0.828**	**467.1**	16.4	0.805	622.5	10.7	0.776	680.6
R208	5.5	**0.808**	**541.6**	6.5	0.790	605.5	3.8	0.720	718.8
R209	12.0	**0.808**	**543.2**	12.6	0.776	648.0	10.0	0.755	712.1
R210	20.4	**0.855**	**368.3**	20.5	0.830	555.0	12.3	0.791	649.3
R211	1.0	**0.786**	**548.7**	1.0	**0.768**	618.9	1.0	0.732	673.5

5.4 Conclusion

In this chapter, we proposed a new manner to construct knowledge groups that is independent of the multi-objective algorithm used, for example, MOEA/D. This construction is based on weight vectors that decompose the objective space in equally sized regions. With this construction, it is possible to adjust the number of knowledge groups wanted, as well as the strategies for the extraction and injection steps. We proposed a new hybridization between MOEA/D and the knowledge discovery mechanism, taking into account the new construction. At the same time, we changed how the extraction step was performed. Instead of applying it with a probability at the end of the resolution of a subproblem, we add all the final solutions found to a set, and the extraction is performed on all the solutions contained in that set, once all subproblems are considered.

The results obtained highlight the importance of the injection step, showing interest in using the proposed knowledge discovery mechanism. Moreover, it is important to consider enough knowledge groups to learn efficiently during the execution. In particular, the best results are obtained when 10 or 20 knowledge groups are constructed (depending on the category of instance). Concerning the local search, we found that using the *first-best* exploration strategy, that we proposed, in combination with the granularity metric d_2, which considers both the distance and the waiting time between customers, was the best combination. However, the use of d_2 as a granularity metric implies determining the neighborhood of each customer for every weight vector that is used during the execution of the algorithm. This may not be very practical if weight vectors are dynamically updated. That being said, the metric d_3 should be preferred when weight vectors are not used or when they are updated during the execution.

In this first part, we started with a simple, yet efficient, multi-objective algorithm, MOEA/D. Since MOEA/D is based on the resolution of several single-objective problems, we have accordingly extended the PILS mechanism, initially proposed to solve CVRP, to solve a bVRPTW. The resulting hybridization showed good performances on a standard benchmark. To improve the results obtained, we decided to focus on the creation of a local search adapted to the problem, which led to the proposition of a new exploration strategy, called *first-best* and different granularity metrics to reduce the size of the explored neighborhood. Using a local search enabled us to consider learning from local optima solutions only, improving the overall quality of the solutions returned, to the detriment of a bit of speed. Secondly, we improved the construction of knowledge groups by making them independent of MOEA/D, allowing us to control the number of groups created. We proposed a second (better) hybridization and analyzed the influence of different parameters, showing the importance of the steps related to knowledge discovery.

In the next part of the manuscript, we attempt to go one step further by presenting a knowledge discovery-based model that can be integrated into classical multi-objective algorithms.

Part III

Solution-based Knowledge Discovery Model

Chapter 6

Unification of Multi-objective Genetic Evolutionary Algorithms and Multi-objective Local Search Algorithms

Contents

This chapter's contributions are linked to the following accepted paper:

- Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, ME. 2024. Solution-based Knowledge Discovery for Multi-objective Optimization. International Conference on Parallel Problem Solving from Nature (PPSN 2024). https://hal.science/hal-04639219.

6.1 Introduction

In a desire of integrating our knowledge discovery mechanism into other multi-objective algorithms, we observed that most of classical multi-objective algorithms have more in common than it seemed at first sight. Two families of algorithms are often used when solving multi-objective combinatorial optimization problems: Multi-Objective Evolutionary Algorithms (MOEA), and Iterated Multi-Objective Local Search (simply abbreviated MOLS), which are the multi-objective version of

Iterated Local Search. These two families of algorithms have already been presented in Chapter 1. The objective of this introductory chapter of this third part of the thesis is to show that these algorithms share similar components. As a result, we propose a unified view of these two families of multi-objective algorithms, allowing the integration of the knowledge discovery mechanism into these classical algorithms.

MOEA consider evolutionary principles, like mutation and crossover operators, to evolve a population of solutions. The best solutions survive to the next generations, conducting the search. Components associated with MOEA are summarised in Section 6.2. MOLS rely on neighborhood exploration to find better solutions. Historically, the first MOLS were extension of single-objective local search (like multi-objective simulated annealing [Serafini, 1994] and multi-objective tabu search [Hansen et al., 1997]), then local search based on the Pareto dominance relation, known as Pareto Local Search [Paquete et al., 2004], have been developed to avoid using aggregation of objectives to measure the quality of solutions. We present in Section 6.3 the unification of MOLS proposed by Blot et al. [2018b]. Furthermore, Iterated MOLS are MOLS with an additional perturbation step occurring when convergence is detected. Finally, our unified view is presented in Section 6.4.

6.2 Components of MOEA

Here we focus only on Genetic Algorithms (GA), which are the most known and the most used evolutionary algorithms in the literature. We recall that Algorithm 4 (see Section 1.2.2) details the main steps of a GA in a single-objective context. In a multi-objective context, the main difference is the output, which is the archive containing the best non-dominated solutions (instead of a single solution corresponding to the best known found far). Note that, in multi-objective algorithms an external archive is commonly used in addition to the current population, to store the best non-dominated solutions. Moreover, the current population and/or the external archive may be bounded and use diversity mechanisms as presented in Section 1.3.4.1.

With a reference to the Algorithm 4, one can observe that the steps `Crossover` and `Evaluate` are problem-dependent and generally do not differ between single-objective and multi-objective optimization. The `Mutation` step is problem-dependent too and in most cases, a standard mutation is used. However, similarly to single-objective optimization where the mutation can be replaced by a local search, in a multi-objective context the mutation can also be replaced by a local search. When it is possible, the local search can be single-objective (like in MOEA/D), otherwise it is a MOLS (without perturbation). The two remaining steps, `Select` and `Update`, naturally differ between single-objective and multi-objective optimization since they require comparing solutions. When a single objective is considered, several strategies have already been presented in Section 1.2.2 (elite selection, tournament selection, and roulette wheel selection). With multiple objectives, archiving strategies must be considered. Several ones have been described in Section 1.3.4.2 and in Section 1.3.4.1 to maintain diversity (e.g., crowding distance with Pareto ranking). In the particular case of MOEA/D (see Chapter 3), several single-objective subproblems are generated and iteratively solved. The `Select` step returns two solutions associated with neighboring subproblems of the one being solved. Concerning the `Update` step, an external archive is completed with the new solutions found, and solutions of neighboring subproblems can be replaced by better solutions (which is specific to MOEA/D). In addition, for more complex variants of MOEA/D, the weight vectors and the subproblems themselves can be modified during the `Update` step. In NSGA-II (see Section 1.3.4.2), the `Select` step employs common selection strategies (e.g., elite selection), but the

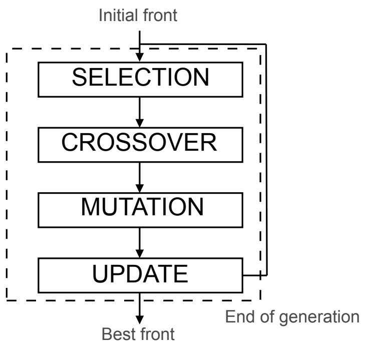


Figure 6.1: Main components of a MOEA.

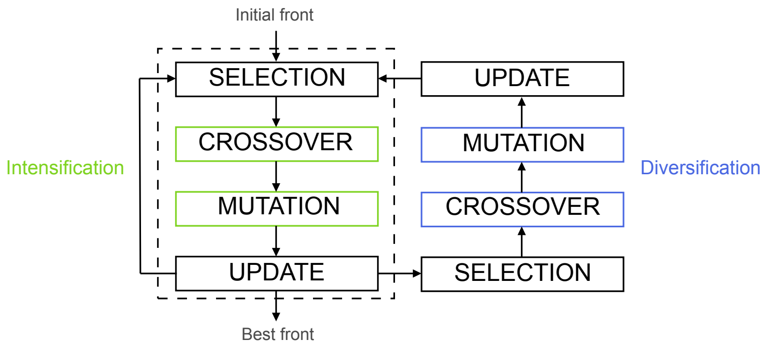


Figure 6.2: Structure of a MOEA with two kinds of generation strategies. An intensification strategy is performed until convergence is detected, then a diversification strategy is applied to start from a new population.

`Update` step is based on Pareto ranking and crowding distance.

Figure 6.1 shows the main steps of a MOEA. We recall that an important point about MOEA, and more generally with evolutionary algorithm (EA), is the trade-off between exploitation (i.e., intensification) and exploration (i.e., diversification). According to the strategies followed in `Crossover` and `Mutation` steps, it is possible to alternate between exploitation and exploration. For example, the solutions selected for the crossover can be close (exploitation) or far distant (exploration) in the objective space. The mutation can be a local search (exploitation) or a random perturbation (exploration). Hence, it is possible to apply an intensification strategy during a few generations and, when the population has converged, switch to a diversification strategy for one generation to create a new population. This situation is represented in Figure 6.2.

6.3 Components of MOLS

6.3.1 Overview of the History of MOLS

A MOLS is an algorithm that iteratively explores solutions selected from a current population, by using neighborhood operators, accepts candidate solutions during the search, and then updates an archive of non-dominated solutions. The latter is returned when a termination criterion is reached. In the literature, we find different families of MOLS.

Originally, the first ones were extensions of single-objective local search based on the aggregation of the objectives to exploit existing single-objective algorithms. For example in the multi-objective simulated annealing [Serafini, 1994, Ulungu et al., 1995] (MOSA) the acceptance of a neighbor solution is based on probabilities, which are themselves based on whether the neighbor dominates, is dominated by, or is incomparable to the current solution. Hansen et al. [1997] proposed the first multi-objective algorithm based on tabu search (MOTS), where the best non-tabu neighbor of the current solution is accepted. These algorithms use a single-solution trajectory, meaning that only one solution is considered during the execution (opposite from population-based algorithms). However, they maintain an archive of solutions containing the current best non-dominated solutions. Czyżak and Jaszkiewicz [1998] extended the MOSA by using a set of current solutions instead of a single current solution, allowing the algorithm to converge into multiple optima at the same time. The two-phase local search procedure (TPLS) form Paquete and Stützle [2003], developed for bi-objective optimization, starts by generating an initial solution considering the first objective only. Then an aggregation, more oriented towards the second objective is used to perform a local search starting from the last solution obtained. This step is iterated until the final local search which considers the second objective only.

Then local search techniques have been hybridized with MOEA to benefit from evolutionary mechanisms. In particular, the multi-objective genetic local search (MOGLS) was first proposed by Ishibuchi and Murata [1998]. MOGLS hybridizes a genetic algorithm with a single-objective local search. Each time the local search is applied, new random weights are chosen to aggregate the objectives. Knowles and Corne [1999] described the Pareto archived evolutionary strategy (PAES), presented as "the simplest non-trivial approach to a multi-objective local search procedure". PAES is an evolutionary algorithm, without crossover, using only local search techniques. A neighbor is accepted only if it dominates the current solution or if it belongs to a less crowded region of the population.

Finally, we find MOLS which neither aggregate objectives nor use evolutionary mechanisms, but instead rely on Pareto dominance to accept neighbors, like the Pareto Local Search [Paquete et al., 2004] (PLS), where a single solution not yet considered is taken from the current archive to explore its neighborhood, and all of its neighbors are used to update the archive (only non-dominated solutions are kept). An indicator-based multi-objective local search is proposed by Basseur et al. [2012]. A neighbor is accepted if by replacing one solution of the archive it improves the current hypervolume. To escape from local optima, using restart mechanisms is a necessary step. The Iterated PLS (IPLS) from Drugan and Thierens [2012] is a multi-restart version of the PLS, which restarts from a new random solution when the neighborhood of all solutions of the archive has been explored. Moreover, after a given number of iterations, IPLS applies a mutation to a solution randomly selected from the archive, and restarts from the resulting solution. In the literature, we already find two generalizations of the PLS algorithms: the dominance-based multi-objective local search (DMLS) proposed by Liefooghe et al. [2012], and the stochastic Pareto local search (SPLS) from Drugan and Thierens [2012].

6.3.2 Basic Components of MOLS

In order to unify all the existing MOLS, Blot et al. [2018b] highlighted the basic components that are used in all of them. These components are described thereafter.

The *archive* is the set containing all the current best non-dominated solutions. This is the set of solutions returned at the end of the execution. If there are many Pareto optimal solutions the archive can be bounded, following the strategies described in Section 1.3.4.1.

The *memory* is a second set of solutions containing the current population of solutions, i.e., the solutions that can be explored. Some of these solutions may be dominated by solutions of the archive. Similarly to the archive, bounding mechanisms can be used. However, it seems advantageous to limit only the memory and keep the archive unbounded. As already seen above, the memory can contain either a single solution or multiple solutions.

The exploration strategies describe which neighbors to accept as candidate solutions and when the exploration stops. The acceptance criterion is related to the quality of the solution, regarding a reference. The reference can be the current solution or a set of solutions (like the memory or the archive). To measure the quality of a neighbor, it is possible to aggregate the objectives or to use Pareto dominance. When Pareto dominance is used, the neighbor can be accepted when it dominates the reference, or when it is non-dominated (regarding the reference).

Like in single-objective optimization, the neighborhood can be fully (e.g., *best* improvement strategy) or partially (e.g., *first* improvement strategy) explored. For example, in a multi-objective context, the exploration can end when the first non-dominating neighbor of the current solution is found, and all non-dominated neighbors encountered are accepted. The accepted neighbors can be used to update directly the archive and/or the memory. When the memory contains multiple solutions, a subset of solutions can be explored (instead of exploring all solutions), however, if the archive or the memory is updated during exploration, the order of exploration may strongly impact the performance.

When the memory has been explored and the archive updated with all the accepted neighbors, the memory must be updated for the next iteration. An entire memory can be constructed by considering solutions from the archive or only explored solutions can be replaced either by one of their accepted neighbor or by an unexplored solution from the archive.

A natural termination criterion for the local search is reached when no more solution is to be explored (i.e., the memory is empty). In that case, all solutions from the archive are Pareto local optima. A possibility to force quick convergence or ensure diversification, is to remove from the memory partially explored solutions, leading to an empty memory stopping the algorithm. A more common termination criterion can be used like the total computational time, the total number of iterations or evaluations, and the number of successive iterations without improvement (of the archive, or regarding a quality indicator).

Various mechanisms have been developed to escape from local optima in single-objective optimization. Similar mechanisms can be adopted in a multi-objective context when the algorithm is trapped in sets of Pareto local optima. For example, MOSA and MOTS respectively exploit simulated annealing and tabu search properties. Another possibility is to exploit an iterated local search scheme [Drugan and Thierens, 2012]. In that case, a convergence condition is considered (e.g., a threshold in the convergence rate) to restart from new solutions of the search space, or from close solutions to the current or best ones, by using a *kick* strategy. Some solutions are selected from the memory or the archive to undergo a given number of random moves. The resulting non-dominated solutions form a new Pareto set from which the algorithm restarts.

6.3.3 Unification of MOLS

From the components described above, Blot et al. [2018b] defined a unified structure of MOLS algorithms. This unification is composed of three main procedures: the main loop of the local search, the exploration of the neighborhood of solutions (Algorithm 14), and the iterated version (Algorithm 15).

The main loop of the local search, presented in Algorithm 13, iterates the three following steps until the memory is empty or as soon as the stopping condition is met. For each solution of the memory, a reference is defined (`Reference`), which is used to accept neighbors as candidates (`Explore`), and accepted neighbors may be used to update the memory (`Update`). All accepted neighbors are used to update the archive (`Combine`) after the exploration of all solutions of memory or the realization of an early stopping condition. The last step updates the memory for the next iteration (`Select`).

The `Explore` procedure is described in Algorithm 14. The neighbors of the current solution are generated until they all have been generated or a stopping condition is met. If the neighbor satisfies the acceptance criterion (depending on the reference), it is added to the set of accepted neighbors (`Accept`). Moreover, the current solution, the reference set and the archive can be directly updated during the exploration (`Update`).

Likewise iterated local search, the MOLS can be iterated, until a stopping criterion is reached, to improve the convergence of the final set obtained. An additional Pareto set $archive^*$ tracks the overall non-dominated solutions found during local search iterations. By using a perturbation step, a new memory and a new archive are created (`Perturb`), used as inputs of Algorithm 13. The final archive obtained is combined with $archive^*$ (`Combine`).

Algorithm 13: Multi-Objective Local Search (MOLS) framework.

```
Input: memory, a set of solutions to explore, archive a Pareto set of solutions.
Output: the updated archive
1  while not local search stopping condition and memory ≠ ∅ do
2      allAccepted ← ∅
3      while not iteration stopping condition and not all memory considered do
4          let current ∈ memory
5          ref ← Reference(current, memory, archive, allAccepted)
6          accepted ← Explore(current, ref, archive)
7          memory ← Update(memory, current, accepted)
8          allAccepted ← allAccepted ∪ accepted
9      archive ← Combine(archive, allAccepted)
10     memory ← Select(memory, archive, allAccepted)
11 return archive
```

Algorithm 14: `Explore` procedure.

```
Input: current, a solution to explore the neighborhood, ref set of solutions to compare
       neighbors with, archive a Pareto set of solutions.
Output: accepted, the set of accepted neighbors
1  while not exploration stopping condition and not all $\mathcal{N}(current)$ considered do
2      let neighbor ∈ $\mathcal{N}(current)$
3      accepted ← Accept(accepted, neighbor, ref)
4      current, ref, archive ← Update(ref, accepted, current, archive, neighbor)
5  return accepted
```

Figure 6.3 contains a simplified overview of Algorithm 13 and Algorithm 15, representing a MOLS. A MOLS iterates three main steps, the selection of solutions to put in the memory (SELECTION), the exploration of the neighborhood of the solutions of the memory (EXPLORATION), representing l.3-8 of Algorithm 13, and the update of the archive (UPDATE). After convergence, a perturbation step occurs to generate a new archive (PERTURBATION), and a MOLS is applied again until a global termination criterion is reached.

Algorithm 15: Iterated MOLS framework.

```
Input: $archive$ a Pareto set of solutions.
Output: the final $archive^*$ set
1 $archive \leftarrow$ MOLS($archive$)
2 $archive^* \leftarrow archive$
3 while $not\ global\ stopping\ condition$ do
4     $memory, archive \leftarrow$ Perturb($archive, archive^*$)
5     $archive \leftarrow$ MOLS($memory, archive$)
6     $archive^* \leftarrow$ Combine($archive, archive^*$)
7 return $archive^*$
```

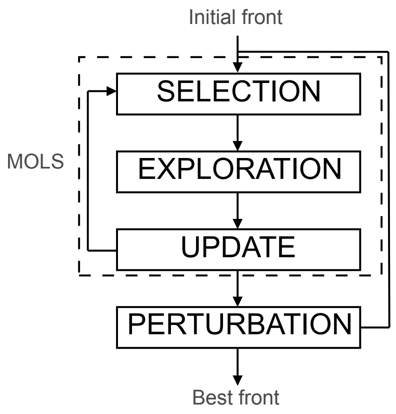


Figure 6.3: Main components of a MOLS.

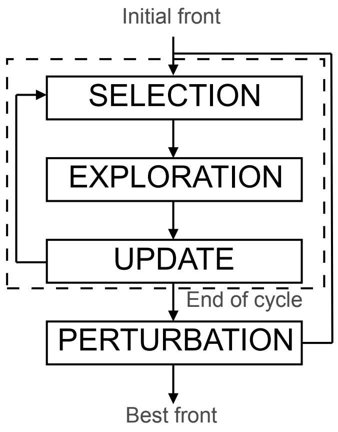


Figure 6.4: The proposed unified view for MOLS and MOEA metaheuristics.

6.4 Proposition of a Unified View for MOLS and MOEA

This section shows the structural similarities between MOLS and MOEA through a unified view, based on the considerations made in the two former sections.

The MOLS and MOEA frameworks can be abstracted with the following four main steps: `Selection`, `Exploration`, `Update`, and `Perturbation`. The Figure 6.4 shows how these steps interact together. The `Exploration` step is used for intensification while the `Perturbation` one for diversification. A *cycle* is defined as a succession of a fixed number of iterations of the three first steps (i.e., `Selection`, `Exploration`, and `Update`). In MOLS, a cycle corresponds to an execution of MOLS, while in MOEA, a cycle represents a number of generations where an intensification strategy is applied to produce the best possible population. When a cycle ends and a specific condition is met, a `Perturbation` occurs to update the current population before the next `Selection`, and so forth, until a termination criterion is reached (generally based on time or a number of iterations). Three sets of solutions are handled during the execution: the external archive (i.e., $archive^*$ in MOLS), which is the set containing the best overall solutions found during the execution, the (internal) archive (i.e., the current population in MOEA), which contains the Pareto set of the current cycle, and the selected solutions (i.e., the memory in MOLS). This is the external archive that is finally returned. The steps are discussed below with details about their instantiation in MOLS and MOEA.

The `Selection` step chooses one or several solutions to explore in the archive, initialized with the initial front provided. This choice can be done randomly, or following a criterion to focus on a specific region of the objective space. In MOLS, the selection is directly performed from the archive, producing the memory. In MOEA, solutions are selected from the population to undergo crossover and mutation steps. More precisely, in MOEA/D, the selection depends directly on the current subproblem solved, since solutions are selected in the subproblem neighborhood.

`Exploration` is the intensification step of the algorithm where the search focuses on specific regions of the search space (i.e., related to the selected solutions). During this step, a mechanism associates the selected solutions with new *better* candidate solutions. Note that, during this step, the set of selected solutions can be directly updated (e.g., replacement of the solution explored). In MOLS, the exploration consists of accepting either non-dominated or dominating neighbors of the selected solutions, considering a reference set. Consequently, many iterations are needed to reach a Pareto local optima. In MOEA the exploration consists of applying the crossover and mutation steps to improve the current population. The mutation can be replaced by a local search (which can be single-objective in MOEA/D), producing local optima solutions.

When new solutions are found after the exploration, the `Update` step tentatively integrates them into the archive (and possibly in the external archive) where only non-dominated solutions are kept. While the archive generally relies on bounding mechanisms, the external archive remains unbounded.

In both neighborhood-based and evolutionary algorithms, it is necessary to perturb solutions to escape regions with local optima. The `Perturbation` generates new solutions to be explored by applying random moves, destroy and repair mechanisms, or genetic operators to solutions from the external archive. It acts like a diversification step where new regions of the search space can be identified and then explored. After the perturbation, the solutions are used to create a new current population, and a new cycle is started. In MOLS, the perturbation relies on local search mechanisms. In MOEA, it corresponds to a crossover and/or a mutation.

Our unified view captures the global behavior of MOLS and MOEA metaheuristics, simplifying the integration of the solution-based knowledge discovery mechanism into multi-objective algorithms fitting this view.

Chapter 7

Unified Solution-based Knowledge Discovery Model

Contents

This chapter's contributions are linked to the following accepted paper:

- Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, ME. 2024. Solution-based Knowledge Discovery for Multi-objective Optimization. International Conference on Parallel Problem Solving from Nature (PPSN 2024). `https://hal.science/hal-04639219`.

7.1 Introduction

In this chapter we develop a unified Solution-based Knowledge Discovery (SKD) model, using the unified view proposed in the former chapter. The SKD model is based on three main steps: the creation of the knowledge groups, the extraction and injection procedures.

The model is described in Section 7.2, where a focus is made on the three steps composing it. In particular, a new construction of knowledge groups is presented. In Section 7.3, the model is instantiated in a MOLS and a MOEA/D. Finally, the algorithms are evaluated and compared in Section 7.4.

7.2 Unified Solution-based Knowledge Discovery Model

In this section, the SKD model is described within the unified view proposed in Chapter 6. The SKD model integrates three additional steps. One focuses on the creation of knowledge groups `Creation`, and the two other ones are `Extraction` and `Injection`, respectively related to the extraction and injection steps.

The creation of the knowledge groups is performed at the beginning of the execution. When an initial front is provided, the groups can directly exploit it by extracting the knowledge from its solutions. Moreover, the region associated with each group can take into account the front provided to directly focus on interesting part of the objective space. A strategy to create the knowledge groups has already been proposed in Section 5.2.1 and is recalled in Section 7.2.1. In addition, a new construction strategy is presented, which adapts to the current Pareto front.

The extraction (resp. injection) procedure is made more generic in Section 7.2.2 (resp. Section 7.2.3) to facilitate the integration into other metaheuristics. Furthermore, it becomes easier to change the extraction and injection strategies with the model presented.

Once the three components are described, they are integrated into the unified view in Section 7.2.4, forming the SKD model.

7.2.1 Adaptive Knowledge Groups

This section introduces a new construction of knowledge groups, with the particularity that the regions of the knowledge groups is adapted regarding the current Pareto front. We recall that a knowledge group gathers structural elements of the solutions belonging to the same region of the objective space. Our first proposition of knowledge groups, presented in Section 4.3.1 was linked to the subproblems defined in MOEA/D, associating a knowledge group with a single subproblem. Then, we proposed an enhanced construction, named WG in the following, defined in Section 5.2.1, which associates each knowledge group with a region of the objective space. To define the region of each knowledge group we used weight vectors. However, once the regions defined, they do not change during the execution of the algorithm, which may not be appropriate to take into account the evolution of the Pareto front during the execution.

Our new strategy, named EG, to define the representatives of the knowledge groups, is represented in Figure 7.1. In a bi-objective context, a straight line links the extreme points of the current front, and then, k_G points (including the extreme points) are regularly created on the line. Each created point corresponds to a representative of a group. The proximity of a solution to a group is then evaluated by the Euclidean distance between the objective vector of the solution and the representative. When more objectives are considered, we consider the hyperplane passing through all the extreme points of the front and k_G points are sampled in the hyperplane. With this strategy, it is possible to dynamically update the representatives of each group, before the extraction, if the extreme points vary. In Figure 5.1 (from Section 5.2.1) and Figure 7.1, each solution of the Pareto front is linked to its closest representative, which leads to different distributions for each construction.

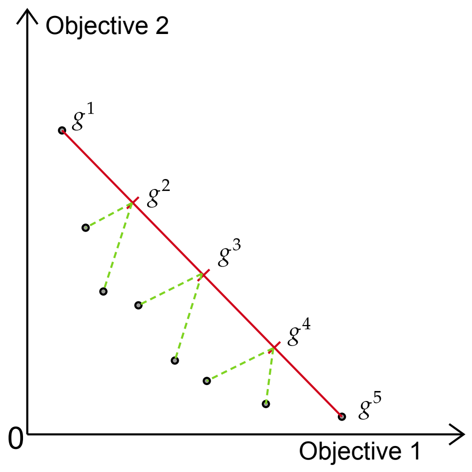


Figure 7.1: Creation of five representatives, each one defining the region of a knowledge group. The representatives are placed on the line linking the extreme points of the Pareto front. This construction is denoted EG.

7.2.2 Interaction between Extraction and Knowledge Groups

In this section, we present the extraction procedure and its interaction with knowledge groups. Although the extraction has already been presented in Chapter 4 and Chapter 5, the mechanism was too dependent on the algorithm used, that is MOEA/D. Here we define the extraction as a standalone component, with its own strategies and mechanisms.

The extraction procedure is presented in Algorithm 16. Before applying the extraction, the representatives of the groups can be updated (if the EG construction is used) with the current archive. For the extraction procedure, a *learning set* L of solutions obtained during the execution of the algorithm is provided. This set contains solutions from which it could be interesting to learn the structure. However, multi-objective algorithms explore plenty of solutions during their execution, and learning from all of them would scramble the knowledge added to the groups. Consequently, the procedure `Filter` generates a subset of L, containing the solutions that undergo the extraction procedure. For example, `Filter` can only keep the non-dominated solutions of L. In particular, this strategy allows the learning to focus on the most interesting solutions. Other possibilities are also available like a random sample or a mix of dominated and non-dominated solutions, if there is not enough non-dominated solutions.

Once L is filtered, knowledge is extracted from each remaining solution x. It is then added to the d_e closest groups (function `SelectGroups`, l.4 of Algorithm 16) of x, following the evaluation of the proximity between a solution and a group provided. The parameter d_e allows the control of the diversification of the mechanism: smaller values correspond with fewer groups being updated, resulting in an intensification strategy. Then, the elements of knowledge are added to the corresponding groups (`Update` procedure), and a score (basically the frequency of appearance in our case) reflecting the relevance of each element is updated. Moreover, we choose not to allow the same solution to contribute more than once to a group, to avoid the bias induced by local optima. The set L is emptied after updating the groups.

The construction of L and the function `Filter` used in Algorithm 16 are presented in Section 7.3

since they are algorithm-dependent.

The functions `Extract` (l.3) and `Update` (l.5) are problem-dependent and are discussed hereafter, in this section. `Extract` did not change from the beginning (see Section 2.8.3): sequences of consecutive customers in routes (excluding the depot) are extracted. The maximum size of the patterns extracted is controlled by the parameter s_p. Concerning `Update`, initially all sequences have a score of 0, and each time a sequence is added to a knowledge group its associated score is incremented by 1. Moreover, when a threshold l_f is exceeded by the score the associated sequence is tagged *frequent*, meaning that it can be used during the injection procedure. In the first part of the thesis, we did not consider such a threshold (equivalent to $l_f = 1$), however, many patterns are extracted only once, and should not be considered during the injection step. Moreover, this threshold filters the patterns that can be used during injection (i.e., the candidate patterns) limiting the choice to the most promising patterns.

Algorithm 16: `Extraction` procedure.

```
Input: $\mathcal{A}$ the current archive, $\mathcal{G}$ the knowledge groups, $L$ the learning set, and $d_e$ the
       number of groups to update.
Output: The updated knowledge groups.
1 $\mathcal{S} \leftarrow$ Filter$(L)$
2 for $x \in \mathcal{S}$ do
3     $\mathcal{K} \leftarrow$ Extract$(x)$
4     $G = \{G_1, \ldots, G_{d_e}\} \leftarrow$ SelectGroups$(\mathcal{G}, d_e, x)$
5     Update$(G, \mathcal{K})$
6 $L \leftarrow \varnothing$
7 return $\mathcal{G}$
```

7.2.3 Interaction between Injection and Knowledge Groups

As it was done for the extraction step, this section presents the injection procedure as a standalone, making it independent from the algorithms used. Any solution found can undergo the injection procedure described in Algorithm 17.

First, the knowledge to inject in the solution x must be selected. Similarly to the extraction, a subset of d_i groups containing the closest groups to x is created (function `SelectGroups`). The d_i parameter controls the diversification of the injection. The function `SelectOne` selects among the d_i candidate groups the final group that will produce the knowledge to inject in the solution x. The selection can be done at random, or following a specific criterion if a group is preferred. Then, some knowledge is selected from the resulting group with the `SelectKnowledge` function. In particular, the knowledge should be selected considering the scores of the elements in the group. In that case, it is possible to select the elements with the highest score or by means of a roulette wheel mechanism. Each element k of knowledge is tentatively injected into a solution x' (initially x) using the function `Inject`. All solutions accepted (e.g., those dominating x') during the injection of k are added to a set S'. Indeed, when we hybridized the knowledge discovery with MOEA/D, only the solution with the best fitness was accepted during the injection, and more precisely, during the reconnection step. Using algorithms that do not use an aggregation of objectives implies defining an acceptance condition to select the next solution. The next solution x' to undergo the injection can be replaced by taking one of the solutions of S' (function `SelectNextSolution`). For that choice, it is possible to select a solution at random, with a dominance criterion, or with an aggregation when it is defined. Finally, after the injection of all the elements of knowledge, all the accepted solutions are returned.

The functions `SelectOne` (l.2), `SelectKnowledge` (l.3), and `SelectNextSolution` (l.7) used in Algorithm 17 are defined in Section 7.3 since they are algorithm-dependent. The problem-dependent function `Inject` (l.6) has been described in Section 2.8.2. We briefly recall that the injection mechanism starts by removing the arcs connected to the patterns, forming pieces of routes that are then reconnected to create feasible solutions. Reversed patterns are discarded and in a multi-objective context, there is possibly no best solution, then the Pareto dominance relation should be used instead as a criterion to accept solutions and discard others.

Algorithm 17: `Injection` procedure.

```
   Input: $\mathcal{G}$ the knowledge groups, $x$ the current solution, and $d_i$ the number of candidate
          groups.
   Output: Accepted solutions containing (at least) one injected pattern.
1  $G = \{G_1, \ldots, G_{d_i}\} \leftarrow$ SelectGroups($\mathcal{G}, d_i, x$)
2  $G' \leftarrow$ SelectOne($G$)
3  $\mathcal{K} \leftarrow$ SelectKnowledge($G'$)
4  $S \leftarrow \varnothing$
5  $x' \leftarrow x$
6  for $k \in \mathcal{K}$ do
7      $S' \leftarrow$ Inject($k, x'$)
8      $x' \leftarrow$ SelectNextSolution($S', x'$)
9      $S \leftarrow S \cup S'$
10 return $S$
```

7.2.4 Presentation of the SKD Model

The Solution-based Knowledge Discovery (SKD) uses knowledge groups and the procedures of extraction and injection suited to multi-objective algorithms. These components have been described in the sections above. The Unified View presented in Section 6.4 contains successive steps of intensification and diversification. The intensification is usually the core of the multi-objective algorithms where identified regions of the search space are deeply explored using an underlying local knowledge given by the neighborhood. In this section, we integrate the SKD into multi-objective algorithms using our unified view in Figure 6.4. We aim to improve the diversification phase (symbolized by the `Perturbation` step), by exploring larger regions of the search space with the knowledge stored in the groups. The conception of the SKD model is presented in Figure 7.2.

At the beginning of the execution, given an initial front, the knowledge groups are created following one strategy presented in Section 7.2.1.

It is possible to deactivate the extraction until a certain execution time is reached, to balance low-quality and high-quality solutions. Indeed, if the initial front contains solutions of poor quality, it should be better to wait a few iterations to learn from solutions of higher quality. In the following, we activate the procedure with no delay, at the beginning of the execution, since an initial front

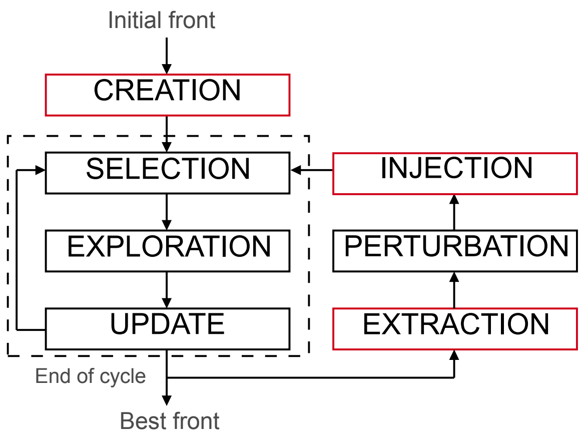


Figure 7.2: The unified view integrating the three steps of the Solution-based Knowledge: the `Creation` of knowledge groups, the `Extraction`, and the `Injection` procedures.

containing improved solutions is provided.

Applying the extraction procedure at every iteration risks biasing the knowledge groups towards the currently explored region of the objective space. In particular, waiting a few iterations allows the learning set to contain more interesting solutions. Hence, the `Extraction` step should be applied only after the end of a cycle, on a subset of explored solutions (i.e., the learning set). Note that, before applying the extraction, it is possible to update the representatives of the knowledge groups.

Any solution can undergo the injection phase but, like the `Extraction`, applying it to all the explored solutions would waste computational resources. In addition, the `Injection` can be a costly step for some problems (e.g., in routing problems). Thus, we consider that the injection should be applied only after the end of a cycle and more precisely:

- after the `Perturbation` if it occurred; in that case, the injection is performed on perturbed solutions

- after the `Extraction`; in that case, we recommend applying the injection on solutions from the current population or archive.

or After the injection, a new cycle (i.e., an intensification step) is started by updating the archive and the current population.

7.3 Instantiation of the Model for MOEA/D and MOLS

This section presents instantiations of the SKD model presented above in two multi-objective algorithms. The model is integrated into a MOEA/D (Section 7.3.1), and another integration is proposed in a MOLS (Section 7.3.2), and more precisely in a Dominance-based Multi-objective Local Search (DMLS) [Liefooghe et al., 2012].

7.3.1 Integration in MOEA/D

First, we provide an instantiation of the MOEA/D framework (described in Chapter 3), which is called $\text{R}^{fb,d_2}_{\text{MOEA/D}}$. We generate M weight vectors uniformly distributed, assuming that is enough to obtain diverse subproblems. Each weight vector defines a scalar problem with a weighted sum of the objectives. To generate a set of solutions to explore, a Partially Mapped Crossover (PMX) is applied with probability p_x to generate a new starting solution for each subproblem (this is the `Perturbation` step from Figure 6.4). In particular, only one solution is randomly chosen after the crossover to keep the population's size constant. When the crossover is not applied (due to the probability), the solution associated with the subproblem is kept. Then a cycle starts, during which all subproblems are solved starting from the solution generated before. The exploration is ensured by the mutation, which is a local search applied with probability p_m. The neighborhood operators remain the `2-opt`*, `Relocate`, and `Swap` (described in Section 2.6). The local search is a randomized variable neighborhood descent, where the order of the operators is kept during descent (until a local optimum is reached) but shuffled each time the local search is applied. The components of the local search have already been detailed in Section 4.4: the *first-best* exploration strategy is used with the granularity metric d_2, limiting the size of the neighborhood. We recall that the *first-best* strategy explores subsets of neighboring solutions (here, one customer is considered at a time, and all the moves involving that customer generate the subset to explore), and the search stops once an improving move is found (i.e., it corresponds to apply the best possible move for one customer). The granularity metric d_2 assigns to each arc between two customers a value representing the relevance of the arc (a low value increases the relevance), then some arcs can be discarded if their value is too high. The metric proposed (d_2), has the advantage of adapting the value according to the weight vector used to solve a subproblem in MOEA/D, and the value takes into account both objectives to optimize.

Following the model presented in Section 7.2.4, the `Extraction` and `Injection` procedures are added to the reference MOEA/D, $\text{R}^{fb,d_2}_{\text{MOEA/D}}$. The algorithm using the weight vectors (resp. the extreme points of the front) to create the groups is called $\text{SKD}^{\text{WG}}_{\text{MOEA/D}}$ (resp. $\text{SKD}^{\text{EG}}_{\text{MOEA/D}}$). Concerning the `Extraction`, each solution that is tentatively added to the external archive (and to the current population) during the `Update` step of Figure 6.4, is also added to the learning set. In particular, these solutions are those obtained at the end of the resolution of a subproblem. Then, the knowledge is extracted from non-dominated solutions of the learning set (`Filter` step of the `Extraction`). The `Injection` procedure is applied to all the solutions of the current population (i.e., the current best solution of each subproblem). The `SelectOne` function randomly selects the group (among the d_i closest groups) that gives the knowledge to inject. For the `SelectKnowledge` function, we rely on the scores (i.e., the frequency of appearance in our case) of the elements learned. We consider N_i elements, randomly selected among the N_f elements with the highest scores, as it was done in the former chapters. The next solution (`SelectNextSolution`) is the best (considering the aggregation of the associated subproblem) accepted during the injection.

7.3.2 Integration in MOLS

To present the integration of SKD in a MOLS, we follow the scheme of the DMLS originally introduced by Liefooghe et al. [2012]. We only consider the components related to the algorithm, since the dominance relation and the components related to the problem have already been defined. The algorithm starts from an initial front given by the user, which is integrated into a bounded archive, *archive*, of size U_a, representing the current population. The archive is bounded by using the crowding distance [Deb et al., 2002] (defined in Section 1.3.4.1). Then, U_m randomly selected solutions from the archive, among the not entirely explored ones (i.e., solutions that produced at least one accepted candidate solution during the neighborhood exploration), form the set to explore. The DMLS algorithm iteratively explores the selected solutions. During the local search, the neighborhood of a solution x is explored until a non-dominated solution is found (i.e., this is a *first* improvement strategy), considering all solutions of *archive* (i.e., the reference set is the archive) [Blot et al., 2017]. The neighborhood of a solution is obtained by shuffling the neighborhood operators `2-opt`*, `Relocate`, and `Swap`. Note that, we do not use a granularity mechanism to reduce the size of the neighborhood with this algorithm. If no neighboring solution is accepted, x is tagged as explored and is no longer selected during the current cycle, moreover tagged solutions cannot be selected during the local search. If any, the accepted solution is tentatively added to *archive*.

In the iterated variant, we manage a second (unbounded) archive, $archive^*$, containing the best non-dominated solutions found during the execution. After l_c iterations (denoting the length of a cycle), the uHV of *archive* is evaluated, and the solutions of *archive* are integrated into $archive^*$. Before starting a new cycle, if all solutions of *archive* are tagged as explored or the uHV has not been increased by at least e_{uHV} after two consecutive cycles, a `Perturbation` step occurs. During this step, all existing tags are removed from solutions, all elements from *archive* are tagged as explored and a new archive *archive* is created by perturbing solutions from $archive^*$.

To perturb a solution x, we apply three moves of the local search, with the following acceptance criterion: a solution y is accepted when $x \succeq_{\epsilon_p} y$, i.e., when x ϵ_p-dominates y, with ϵ_p a parameter of the algorithm (see Section 1.3.1). We remarked during preliminary experiments that for a few instances, the archive *archive* did not contain many solutions (less than ten). This point severely impacts the performance of the algorithm since too few solutions are explored. Thus, we use an additional set of solutions together with the archive to have access to more solutions during the exploration phase. The additional solution set is obtained during the perturbation step and it contains solutions generated that are not added to *archive*. As a consequence, the additional set is empty at the beginning of the execution. Moreover, the solutions are selected from *archive* in priority, and then from the additional set when no more solutions are available for exploration. Another change concerns the update of the additional set after the exploration. Suppose a solution from the additional set is explored and provides an accepted solution not added to the archive. In that case, the explored solution is replaced by its accepted neighbor in the additional set. This version of MOLS is called $\mathrm{R_{MOLS}}$.

Following the model presented in Section 7.2.4, the `Extraction` and `Injection` procedures are added to $\mathrm{R_{MOLS}}$. The algorithm using the weight vectors (resp. the extrema of the front) to create the groups is called $\mathrm{SKD^{WG}_{MOLS}}$ (resp. $\mathrm{SKD^{EG}_{MOLS}}$). Concerning the `Extraction` procedure, we have to define how the learning set is managed and how its elements are filtered. Every solution tentatively added to *archive* after the exploration step should be added to the learning set, since it may produce interesting knowledge to exploit. The `Filter` function keeps the non-dominated solutions from the learning set. Concerning the injection procedure, it is sequentially applied to all the solutions from *archive*. The functions `SelectOne` and `SelectKnowledge` are exactly the same as described in Section 7.3.1. Finally, for the `SelectNextSolution` function, the initial solution is returned (x in Algorithm 17). Indeed, since we work with a MOLS algorithm, we prefer staying locally around the solution by attempting to inject knowledge into it rather than trying to highly optimize the solution. Finding a better solution is interesting, but could dominate a large part of the archive, resulting in a loss of diversity.

7.4 Performance Analysis

In this section, we evaluate the performances of the SKD model proposed in the former chapter. The compared algorithms are $\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$, $\mathrm{SKD}^{\mathrm{EG}}_{\mathrm{MOEA/D}}$, $\mathrm{SKD}^{\mathrm{WG}}_{\mathrm{MOEA/D}}$, $\mathrm{R}_{\mathrm{MOLS}}$, $\mathrm{SKD}^{\mathrm{EG}}_{\mathrm{MOLS}}$, and $\mathrm{SKD}^{\mathrm{WG}}_{\mathrm{MOLS}}$. The reference algorithms are $\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$, a version of MOEA/D, and $\mathrm{R}_{\mathrm{MOLS}}$, a version of MOLS. Two variants of the SKD model are integrated into each reference algorithm. The difference between the two SKD variants concerns the construction strategy of the knowledge groups, either using weight vectors (WG) or extreme points of the current Pareto front (EG).

The remainder of the section focuses on the experimental study performed. The value of each parameter is discussed in Section 7.4.1. The experimental protocol is described in Section 7.4.2, and the results obtained are presented in Section 7.4.3.

7.4.1 Choice of Parameters Value

In the first part of the thesis, in Chapter 5, we analyzed the components of a learning variant of MOEA/D. The MOEA/D considered as a reference for this experimental study, $\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$, is the same as the reference MOEA/D, $\mathrm{R}_{\mathrm{MOEA/D}}$, described (and tuned) in Chapter 5. Although the learning mechanism was structured differently (concerning the extraction and injection procedures, which were directly integrated and dependent on MOEA/D), the strategy employed was similar to the one used in our new integration (i.e., an intensification strategy was used for both extraction and injection, the solutions used for the extraction step were identical, as well as the solutions receiving patterns). This allows us to use the values obtained for the parameters at the end of the first part of the thesis.

The tuning of the parameters for the $\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$ variant comes from the tuning with irace performed in Section 5.3.1.2. $M = 40$ subproblems are created, with $m = 10$ neighbors. At most 2 neighbors may have their solution replaced during the update step. The crossover is applied with probability $p_x = 1.00$, and the local search with probability $p_m = 0.10$. The granularity is set to $\delta = 50\%$. Note that, for this experimental study, there are no distinctions between instances of different categories, thus we considered the configuration which returned the best results by testing it on the other category of instance. Hence the configuration selected is the one reported in Table 5.6 for C instances.

For $\mathrm{R}_{\mathrm{MOLS}}$, the parameters are chosen to be fair with $\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$ (i.e., the size of the bounded sets is equivalent, and the time allocated to the different components is similar). The archive is bounded to $U_a = 30$ solutions, and the additional set of solutions contains up to $U_s = 10$ solutions. Each iteration, $U_m = 1$ solution is explored, selected from the archive, and, if needed, from the additional set. The perturbation occurs when the uHV does not increase by at least $e_{uHV} = 10^{-2}$, and during the perturbation, $\epsilon_p = 1.02$. A cycle performs $l_c = 100$ iterations, so that, the duration (in seconds) of a cycle in $\mathrm{R}_{\mathrm{MOLS}}$ is similar to the average duration of a cycle in $\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$.

The parameters value of $\mathrm{SKD}^{\mathrm{EG}}_{\mathrm{MOEA/D}}$ and $\mathrm{SKD}^{\mathrm{WG}}_{\mathrm{MOEA/D}}$ (resp. $\mathrm{SKD}^{\mathrm{EG}}_{\mathrm{MOLS}}$ and $\mathrm{SKD}^{\mathrm{WG}}_{\mathrm{MOLS}}$) are similar and follow from the discussion reported in Section 5.3.3.3. Again, for the variants of MOEA/D, we considered the configuration of fbd_2 (the closest variant) from Table 5.9a, obtained on instances of category C. With the new model introduced, there are no more probabilities related to the injection or the extraction steps (before, the probability of injection was 1.00). The crossover probability (for MOEA/D) becomes $p_x = 0.50$. There are $k_G = 20$ knowledge groups (i.e., 50% of the number of subproblems). The maximum size s_p of extracted patterns is set to 8 (resp. 5) for

Table 7.1: Parameters' values for the MOEA/D variants. The symbol "-" reflects the absence of the parameter in the variant. The EG-MOEAD (resp. WG-MOEAD) variant stands for $\text{SKD}^{\text{EG}}_{\text{MOEA/D}}$ (resp. $\text{SKD}^{\text{WG}}_{\text{MOEA/D}}$).

Parameter	$\text{R}^{fb,d_2}_{\text{MOEA/D}}$	EG/WG-MOEAD
Number of subproblems: M	40	40
Subproblem neighborhood size: m	10	10
Granularity: δ (%)	50	50
Crossover probability: p_x	1.00	0.50
Mutation probability: p_m	0.10	0.10
Exploration strategy: $\mathcal{S}$	first-best	first-best
Granularity metric: d	d_2	d_2
Number of knowledge groups: k_G	-	20
Extraction diversity: d_e	-	1
Maximum pattern size: s_p	-	5 or 8
Threshold frequency: l_f	-	2
Injection diversity: d_i	-	1
Number of patterns selected for injection: N_i	-	100
Number of candidate patterns: N_f	-	250

instances of class 2 (resp. class 1) since large (resp. short) routes are designed. Only this parameter is modified to be more adapted to the instances solved. The knowledge is added to $d_e = 1$ group, and the knowledge to inject is provided by at most $d_i = 1$ group. As a result, both injection and extraction follow an intensification strategy, like in Section 5.3. $N_i = 100$ patterns of the same size are tentatively injected into each solution. They are selected among the $N_f = 250$ most frequent patterns of the corresponding size in the group. The threshold frequency for patterns is set to $l_f = 2$.

The parameters' values are summarized in Table 7.1 (resp. Table 7.2) for MOEA/D (resp. IMOLS) variants. The parameters related to the learning mechanism are the same for all variants, while each reference algorithm has its own set of parameters.

7.4.2 Experimental Protocol

As we did in the first part, the experiments are run on two computers "Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz", with 24 cores each. Our framework is implemented in the jMetalPy framework [Benitez-Hidalgo et al., 2019]. The source code and our results are publicly available[1].

We use Solomon's benchmark (see Section 2.3.1 for details) and Gehring and Homberger's benchmark (see Section 2.3.2) to evaluate the performance of the algorithms.

To fairly compare the algorithms, they are all initialized with the same fronts. The strategy to create the initial fronts is inspired from Blot et al. [2018a], who optimize solutions following directions (like in MOEA/D). However, the directions are updated to focus on the regions of the objective space with few solutions and the initial solution for a subproblem is the best possible (non-dominated) solution currently found. Our strategy starts by generating a set of 12 random solutions,

[1]https://gitlab.univ-lille.fr/clement.legrand4.etu/skd_integration

Table 7.2: Parameters' values for the IMOLS variants. The symbol "-" reflects the absence of the parameter in the variant. The EG-IMOLS (resp. WG-IMOLS) variant stands for $\text{SKD}^{\text{EG}}_{\text{MOLS}}$ (resp. $\text{SKD}^{\text{WG}}_{\text{MOLS}}$).

Parameter	R_{MOLS}	EG/WG-IMOLS
Size of archive: U_a	30	30
Size of additional set: U_s	10	10
Size of memory: U_m	1	1
Increase of uHV required not to restart: e_{uHV}	10^{-2}	10^{-2}
Epsilon value for perturbation: ϵ_p	1.02	1.02
Length of cycle: l_c	100	100
Number of knowledge groups: k_G	-	20
Extraction diversity: d_e	-	1
Maximum pattern size: s_p	-	5 or 8
Threshold frequency: l_f	-	2
Injection diversity: d_i	-	1
Number of patterns selected for injection: N_i	-	100
Number of candidate patterns: N_f	-	250

as well as a set of 12 uniformly spread weight vectors. Each solution is randomly associated with one weight vector and is optimized with the same local search as defined in $\text{R}^{fb,d_2}_{\text{MOEA/D}}$ (i.e., a variable neighborhood descent with the *first-best* exploration strategy and the metric d_2 with a granularity of $\delta = 25\%$ to obtain quick results). During the local search, the non-dominated solutions found are added to the current Pareto set. Then during 20 iterations, for all the weight vectors, the best possible solution in the current Pareto set is selected. If the best possible solution is the last found (which is a local optimum), then the weight vector is replaced by a new random one. The starting solution is a solution randomly selected from the current Pareto set and is optimized with the local search. The final Pareto set is returned. With these parameters, it takes approximately 2 (resp. 8) minutes to generate a front for instances of size 100 (resp. 200). We generate 30 initial fronts for each instance with this procedure, all publicly available[2]. In IMOLS, the initial front is directly used as the initial archive (bounded if needed), however, in MOEA/D, each subproblem is initialized with the best possible solution of the front.

The six algorithms are then executed over 30 seeds on each instance, each seed being associated with a different initial front. The termination criterion for each run is set to 10 (resp. 20) minutes for instances of size 100 (resp. 200). The average uHV obtained over the 30 runs is compared with Pairwise Wilcoxon tests with Bonferroni correction. The reference fronts, composed of all the best non-dominated solutions found during all the experiments performed for the thesis are publicly available[1]. Moreover, to normalize the objectives we use the best ideal and nadir points found during our experiments.

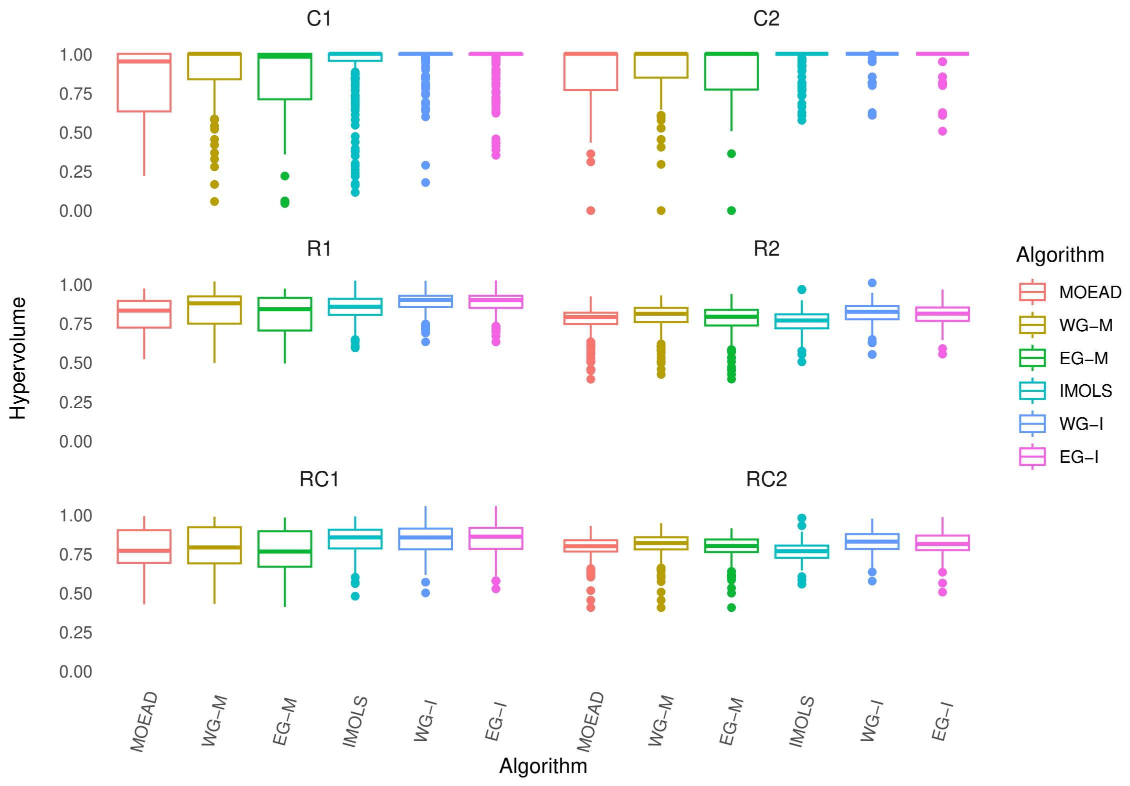


Figure 7.3: Hypervolumes obtained by all the algorithms on the different categories of instance of Solomon's benchmark.

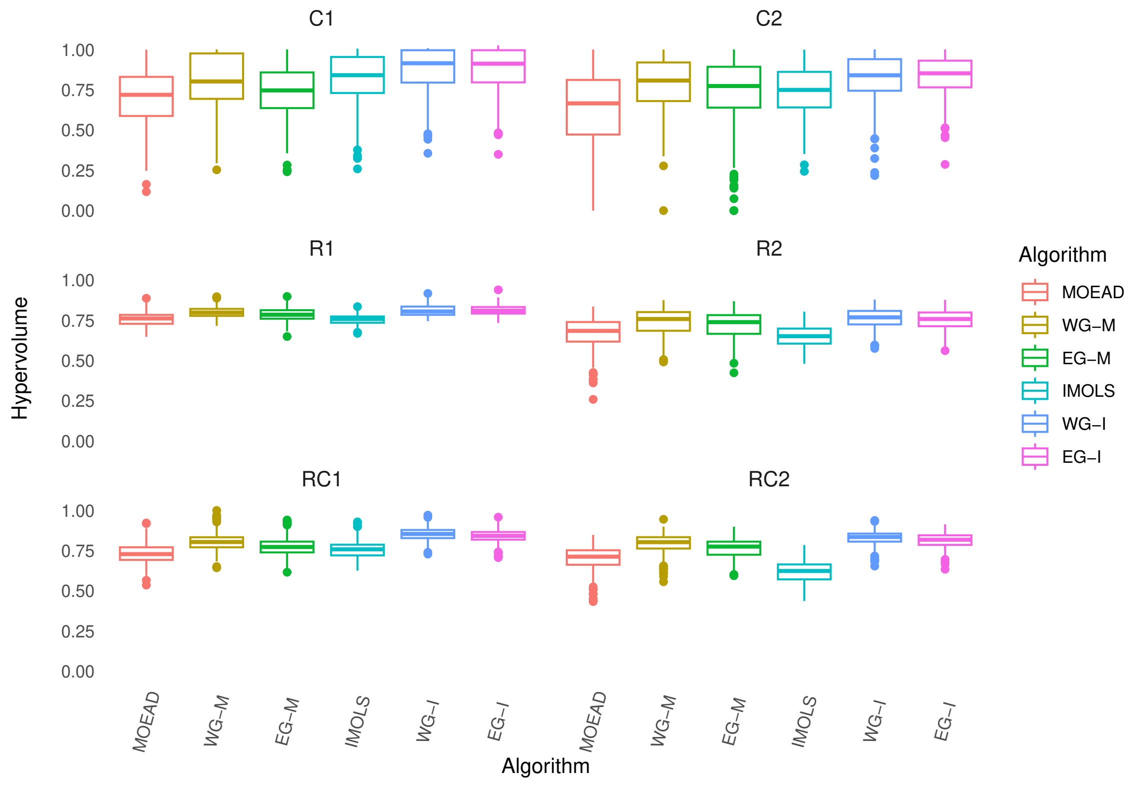


Figure 7.4: Hypervolumes obtained by all the algorithms on the different categories of instance of Gehring and Homberger's benchmark.

Table 7.3: Average uHV ($\times 10^3$) of the algorithms on the different categories of instances. MOEAD ($R^{fb,d_2}_{MOEA/D}$) and IMOLS (R_{MOLS}) are the reference algorithms. EG-M ($SKD^{EG}_{MOEA/D}$), WG-M ($SKD^{WG}_{MOEA/D}$), EG-I (SKD^{EG}_{MOLS}), and WG-I (SKD^{WG}_{MOLS}) are the learning variants. Gray cells identify algorithms that are statistically better on the corresponding set of instances when compared to the other six variants. Bold values represent the best-performing algorithms when MOEA/D (resp. IMOLS) variants are compared together (i.e., three rows each).

Size	100						200					
Categ.	C		R		RC		C		R		RC	
Class	C1	C2	R1	R2	RC1	RC2	C1	C2	R1	R2	RC1	RC2
MOEAD	833	888	805	773	776	792	703	613	755	668	733	702
WG-M	**904**	**912**	**834**	**795**	**784**	**808**	**793**	**788**	**800**	**741**	**806**	**792**
EG-M	856	902	806	778	762	792	744	740	784	723	774	765
IMOLS	923	966	850	761	**837**	766	822	746	754	654	758	619
WG-I	**970**	**987**	**886**	**814**	**844**	**823**	**885**	**826**	811	**761**	**854**	**830**
EG-I	958	**986**	**885**	807	**844**	814	**875**	**835**	**814**	751	842	814

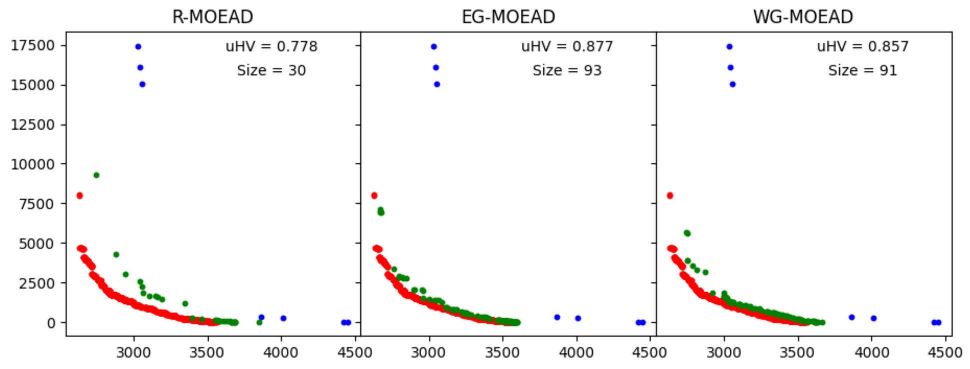
7.4.3 Experimental Results and Discussion

Table 7.3 summarises the results obtained. Detailed results per instance are publicly available[1]. Figure 7.3 (resp. Figure 7.4) illustrates the results obtained on Solomon's (resp. Gehring and Homberger's) benchmark. First, R_{MOLS} returns better results than $R^{fb,d_2}_{MOEA/D}$ except on instances R2 and RC2 of size 100, and RC2 of size 200. Indeed, instances of category 2 are less constrained, leading to a larger exploration space. In that case, it seems preferable to use MOEA/D rather than IMOLS to intensify the search. However, this consideration does not apply to C2 instances, probably due to the presence of clusters, leading to more local optima.

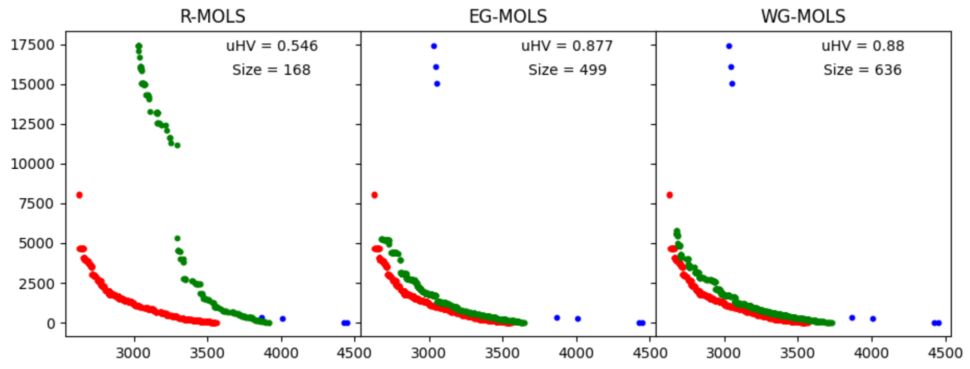
We can see that using SKD (no matter the strategy used to create the groups) positively impacts R_{MOLS} in all instances. The same conclusion holds for $R^{fb,d_2}_{MOEA/D}$ except on RC1 instances of size 100, with EG groups. Moreover, using SKD is even more beneficial in instances of bigger sizes.

In MOEA/D, using the strategy with the weight vectors to create the groups is statistically better than using the other one. This is probably because the algorithm itself uses weight vectors to decompose the search space. Concerning the IMOLS algorithm, both strategies are often equivalent, but using the weight vectors leads to slightly better results. Thus, this strategy should be preferred in general. Additionally, using SKD allows the creation of more diversified Pareto fronts for MOEA/D and IMOLS. Figure 7.5 compares each algorithm's final fronts obtained (blue dots) on one run of instance RC2_2_6 from the Gehring and Homberger benchmark. The reference front is represented (orange dots), and the initial front is provided too (first picture on the left). For each front, the associated hypervolume is computed regarding the reference front, and the size of the front (i.e., the number of non-dominated solutions) is provided. In particular, each front obtained with a learning variant contains more solutions than a front obtained with the reference algorithm (which does not use learning mechanisms). This highlights the capacity of the proposed mechanism to generate new promising solutions, increasing the diversity of the solutions obtained.

[2]https://gitlab.univ-lille.fr/clement.legrand4.etu/skd_integration



(a) Reference MOEA/D and variants.



(b) Reference MOLS and variants.

Figure 7.5: Results of the execution on instance RC2_2_6 (run 6), from the Gehring and Homberger set. The associated hypervolume and size of the final fronts (green dots) are shown, as well as the reference front (red dots) and the initial front (blue dots).

7.5 Conclusion

In this chapter, we have presented our SKD model, which benefits from the unified view of MOLS and MOEA. In particular, the knowledge discovery mechanism can be integrated into algorithms sharing the same structure. Our model requires three additional steps, which are the `Creation` of the knowledge groups, by defining relevant regions of the objective space, the `Extraction` of knowledge from solutions found during the `Exploration` step, and the `Injection` of knowledge into the current population (or archive) to explore other regions of the objective space.

The integration of our model into a MOLS ($\mathrm{R}_{\mathrm{MOLS}}$) and a MOEA/D ($\mathrm{R}^{fb,d_2}_{\mathrm{MOEA/D}}$) leads to four learning variants (there are two possible constructions for groups): $\mathrm{SKD}^{\mathrm{WG}}_{\mathrm{MOEA/D}}$, $\mathrm{SKD}^{\mathrm{EG}}_{\mathrm{MOEA/D}}$, $\mathrm{SKD}^{\mathrm{WG}}_{\mathrm{MOLS}}$, and $\mathrm{SKD}^{\mathrm{EG}}_{\mathrm{MOLS}}$. The six algorithms were compared, showing the performance of our model. In particular, adding SKD to a MOEA/D and a MOLS greatly contributes to the increased quality of the returned Pareto fronts. Moreover, the gain obtained in terms of speed-up, although the codes are executed in Python, shows the relevance of the model developed.

Furthermore, the model proposed is expected to be integrated into other metaheuristics (like NSGA-II), and the integration should not require too many modifications to enhance the performance of the metaheuristics. Depending on the algorithm considered, the parameters related to the learning mechanism should be adjusted. In addition, it is possible to change and adapt the strategies of the learning mechanism without modifying the structure of the algorithm: the major components of SKD have been developed to be as independent as possible from the algorithm in which it is used.

Conclusion and Research Perspectives

Main Conclusion The main goal of this thesis was to smartly and efficiently integrate knowledge discovery mechanisms into multi-objective algorithms to learn (i.e., extract knowledge) from the solutions generated during the execution, and to exploit (i.e., inject) this knowledge in other solutions to explore regions of the solution space that are hard to reach by exploring classical neighborhoods. This milestone was successfully achieved during the thesis, with the emergence of a final model, called Solution-based Knowledge Discover (SKD), that can be integrated into different multi-objective algorithms. Moreover, we described and discussed different strategies to instantiate the SKD model. This model was integrated into a multi-objective evolutionary algorithm (MOEA/D) and a multi-objective local search, to solve a bi-objective vehicle routing problem with time windows, where the total transportation cost and the total waiting time of drivers are minimized. Although we have only discussed the bi-objective case, the model proposed remains available with more objectives.

The idea of extracting sequences of customers from solutions to solve routing problems was not entirely new, since it has already been exploited for the capacitated vehicle routing problem (in a single-objective context), however doing it in a multi-objective context was a challenge. Our methodology was then conducted by the questions raised in the introduction. To answer the first one, "How to manage these sequences in a multi-objective context?", we developed the notion of knowledge groups, each group focusing on a specific region of the objective space. Moreover, we based this strategy on the fact that close solutions in the objective space (i.e., with close objective vectors) shared structural similarities.

The second question concerned the solutions used for extraction and injection: "From which solutions should we extract sequences, and into which solutions should we inject them?". During our study, we tried to learn from all generated solutions, local optima only (in MOEA/D), Pareto local optima only (in MOLS), and non-dominated solutions. The problem with (Pareto) local optima is that it takes time to achieve such solutions, and the algorithm may not generate enough of these solutions to learn efficiently. When we learn from all generated solutions, there are too many solutions, and consequently, too many sequences are added to the groups, with a risk of overfitting the mechanism towards the current solutions found (i.e., always the same sequences will be selected, and new sequences will not be able to reach a high enough score to be selected). As a result, learning from the current non-dominated solutions seems to be a good trade-off, and allowed us to achieve good results. Concerning the solutions used for injection, we did not investigate many strategies. Any solution can undergo the injection step, but it could be interesting to analyze the performance of the injection step, depending on the solution provided in the input. In our final model, the injection is performed on either perturbed solutions or the current non-dominated solutions. Overall, applying the injection on solutions that are not local optima seems preferable since it could be easier to inject sequences in them. Indeed, the structure of local optima prevents the improvement of the solution with a small change.

The last question raised in the introduction was "At what point in the runtime should the injection and extraction steps be carried out?". At the beginning of our study, extraction and injection were both applied with a probability and finally, we removed these probabilities from our final model. Indeed, eliminating the probabilities from the model allows a better comprehension of it, leading, in our opinion, to a more interpretable model, in the sense that it is easier to know where the knowledge comes from and where it is injected. As a consequence, extraction and injection steps are performed at the end of *cycle*, representing a number of iterations, that can be adapted to the algorithm used. Another important aspect concerns the start of the knowledge discovery itself. If it is started too early, random sequences are more likely to be learned, and if it is started too late, not enough different sequences are learned due to the convergence of the algorithm. In this study, we always considered that the mechanism is activated from the beginning of the execution. In a preliminary study, we remarked that learning from the very beginning was not the best strategy, thus we generated initial fronts to not start from random solutions and still activate the mechanism from the beginning of the execution. In addition, the impact of the initial front provided could be further investigated.

In the following, we start by summarizing the contributions obtained during the thesis. Then, we present research perspectives linked to our current works (short-term perspectives) and more global research perspectives that would be interesting to investigate in future works (long-term perspectives).

Summary of the Contributions In the second part of the thesis, we investigated the development of a knowledge discovery mechanism suited to MOEA/D, to solve the bi-objective vehicle routing problem considered. We described six main contributions in this part, which are: the analysis of the genotypic similarity between solutions, the development of the concept of knowledge groups, the development of a local search suited to our problem, the impact of extracting knowledge from local optima, the enhancement of the concept of knowledge groups to make it independent from MOEA/D, and finally the analysis of the different parameters used.

Genotypic Similarity vs. Phenotypic Similarity The analysis of the genotypic similarities between solutions was necessary to know if it was relevant to associate each knowledge group with a region of the objective space. In this study, we analyzed the proportion of arcs in common between one solution and its neighborhood, when the size of the neighborhood varies. We remarked that, with close solutions (i.e., small neighborhood), the proportion of arcs in common was relatively high (at least for solutions close to the Pareto front), i.e., between 25% and 40% of arcs in common depending on the instance. Moreover, the structural similarity decreases when the size of the neighborhood increases.

Concept of Knowledge Groups The first milestone of the thesis was reached with the concept of knowledge groups. Our first construction was based on the subproblems defined in MOEA/D, where a subproblem defined a knowledge group. With the use of MOEA/D in this first part, we were able to use existing concepts developed for single-objective optimization as a basis of our work. The results obtained with this first approach showed an interest in learning sequences from solutions for our bi-objective problem.

Neighborhood Strategies In order to improve the results obtained, we replaced the random mutation used in MOEA/D with a local search, more precisely, a variable neighborhood search using common operators. In the literature, the two most common exploration strategies are the *first* acceptance and *best* acceptance. Here we developed a *first-best* exploration strategy, which considers the best neighbor of a subset of the entire neighborhood, based on existing works in permutation flow-shop problems. In addition, to reduce the size of the neighborhood to explore, we exploited the concept of *granular* search. It considers only moves involving close enough customers for a metric. Thus, we introduced a metric, based on the cost and the waiting time between two customers. Using the *first-best* exploration strategy with our metric in the granular search proved its efficiency in solving the problem.

Learning from Local Optima At that point, having a local search available allowed us to find local optima during the search, raising a natural question: should we extract sequences from local optima only? Indeed, in single-objective optimization, local optima are known to have more interesting structural information. We decided to investigate the benefit of extracting knowledge from local optima in a multi-objective context. Our conclusion highlights that learning from local optima is interesting to reach solutions of better quality, however, in terms of speed-up, it is more interesting not to focus only on local optima since fewer are generated (compared to standard solutions), and it requires more time to generate them.

Enhanced Knowledge Groups Following that, we decided to enhance the construction of the knowledge groups, by making it independent from MOEA/D. At the same time, we also introduced diversification and intensification strategies for extraction and injection steps. The new construction was based on the generation of weight vectors to create the region of a knowledge group. A weight vector is considered as a representative of a knowledge group, and its associated region gathers all the closest solutions of that weight vector (in the objective space). In particular, this construction allows us to control the number of groups created for the execution. More precisely, the results obtained with different values of that parameter showed that using strictly more than one group returns better results. Concerning the strategies for the extraction and the injection, we decided to focus on intensification strategies (i.e., one solution contributes to its associated group only and one solution receives knowledge from its associated group only), which is easier to interpret.

Parameters Analysis Finally, we performed an analysis of the parameters used in the algorithms (a reference MOEA/D and the MOEA/D exploiting sequences). The analysis is based on the irace tool, returning a set of elite configurations (a configuration is a set containing the value of each parameter), being the best configurations tested. The possible values for each parameter are provided to irace to generate the configurations. The analysis of the configurations returned highlight the benefit of the knowledge discovery mechanism.

In the third part of the thesis, the two main contributions concern the highlight of a unified view between multi-objective evolutionary algorithms (MOEA) and multi-objective local search

(MOLS), the development of the SKD model, integrated into the unified view, and instantiated in a MOEA/D and a MOLS to solve the bi-objective routing problem with time windows.

Unified view of MOEA and MOLS The main issue when developing a new mechanism to integrate into different algorithms is to make it too dependent on an algorithm. To overcome this difficulty, we started by reviewing the literature concerning multi-objective evolutionary algorithms and multi-objective local search, to understand what are the main components of each family. By analyzing the structures found, we remarked that both families are based on four main steps: *select* solutions, *exploit* the selected solutions, *update* the sets of solutions (current population and archive), and *perturb* some solutions to favor diversity. These steps have led to the conception of a unified view between MOEA and MOLS.

Unified SKD model Based on the unified view proposed, we developed the SKD model, which requires three additional steps. A *construction* step where the knowledge groups are initialized, an *extraction* step, extracting knowledge from a set of generated solutions, and an *injection* step, exploiting the extracted knowledge in different solutions. Once the model is described, we instantiated it into a MOEA/D and a MOLS, to compare the performances obtained with the reference algorithms. In addition, we developed another construction for the knowledge groups, adapting to the current Pareto front, which has been compared to the static other one (using weight vectors). The results obtained, show the efficiency of the model developed, and the relevance of the strategies employed.

Short Term Perspectives In the following, we explicit several short-term perspectives linked to the improvement of the knowledge groups, and more generally to the improvement of the SKD model developed. In particular, we can investigate the use of different representatives for the knowledge groups, we can develop new formula to update the scores associated with the sequences. On the other hand, it could be interesting to analyze deeper the components developed.

Using Different Representatives for the Knowledge Groups In this thesis, we developed two constructions of knowledge groups, however, it could be interesting to develop other constructions to compare the efficiency of the different constructions and to adapt the construction according to the problem solved. For example, instead of linking the extreme points of the current Pareto front with a straight line to generate the representatives of the knowledge groups, we could use any other curve passing through the extreme points (like Bezier curves), to better approximate the shape of the Pareto front. Moreover exploiting adaptive strategies to update the representative of the groups, and consequently adapt the region of the objective space each group focuses on seems to be promising. The number of knowledge groups could also be adapted during the execution. New groups could be created in crowded regions, while groups containing too few solutions could be discarded. Since it is not guaranteed that close solutions in the objective space share structural similarities for any problem, the structural similarity to the representative could be considered in addition to the (phenotypic) distance to the representative to measure the distance between a solution and a group. In that case, the representatives should be true solutions to the problem. When the distance between a solution and all groups is too high, a new group could be created using that solution as a representative.

More Sophisticated Update of the Pattern Score Another interesting improvement for knowledge groups that could be envisaged concerns the update of the score of the sequences. In this thesis, we only considered the frequency of appearance of each sequence, however, this score does not take into account the characteristics of the sequence itself. Indeed, a sequence could have a high score and be a bad sequence, in the sense that, it does not belong to the solutions of the optimal Pareto front. Moreover, we could receive feedback from the injection of patterns in solutions. For example, some patterns are maybe more interesting to choose when the solution belongs to a specific region of the objective space, while some others should be avoided. In addition, the synergy between patterns could be investigated. For example, if a solution contains a sequence having a high score in one group (A), and we have to select sequences from another group (B), maybe sequences belonging in A and B should be avoided (or preferred). It could also be interesting to reduce the score of sequences over the iterations, to give less importance to sequences found many iterations ago in high quantity, but which have not been found anymore since.

Analysis of the Injection Procedure A deeper analysis of the results obtained after the injection step could also help to understand which patterns should be injected. In particular, concerning the size of the pattern to inject. For that purpose, we could analyze the improvement gap (for all the objectives) in the function of the size of the pattern or its score. Moreover, around 100 sequences were selected each time for the injection. It could be interesting to generate more data to know how many sequences are successfully injected in solutions. The number of successfully injected sequences should also be put in relation to the group that gave the sequences and the group associated with the solution undergoing the injection.

Influence of the Initial Solutions Concerning the best moment to start the knowledge discovery, it could be interesting to analyze the impact of the initial front provided, according to different characteristics. For example, to obtain good results with the knowledge discovery mechanism, it may be necessary to have enough structurally diverse solutions when the mechanism starts. Hence, many different sequences are added to the groups, leading to the creation of more diverse solutions. Since we have already generated several initial fronts, we can record, for example, the number of solutions in it, its average diversity (in terms of objectives and structures), and its convergence, and compare these characteristics with the final front returned.

Analysis of the Knowledge Learned Finally, the sequences learned by each group could be analyzed, to know if the sequences learned, with the highest scores, are similar to those that can be found in the reference Pareto front (or even the optimal Pareto front if it is available). Knowing the characteristics of the sequences belonging to the reference/optimal Pareto front could also help to make a warm start of the knowledge groups, i.e., to generate the beginning sequences for each group. In a preliminary study, we found that initializing the knowledge groups with the sequences belonging to the reference Pareto front, and starting with randomly generated solutions, led quickly to very good high-quality solutions. As a result, learning the *good* sequences from the beginning is the key to quickly converge to good solutions. Another important point concerns the minimum number of *good* sequences that are required. Obviously, the number of *good* sequences is very low compared to the number of *bad* sequences, and we should quickly discard sequences detected as *bad*. However, using *bad* sequences can also be important to diversify the solutions obtained.

Long Term Perspectives In the following, we present some long-term perspectives that could be envisaged to make hybridize our work with other research fields. For example, our model could be adapted to solve different problems. Probabilistic models based on the sequences learned could be interesting to generate new solutions in specific regions of the objective space. Finally, our model could be used as a component for automatic algorithm selection.

Exploiting the Model to Solve Other Permutation Problems Our knowledge discovery mechanism could be applied to different routing problems, to see if the SKD model remains efficient when the problem changes (like the bi-objective TSP, a bi-objective CVRP). More generally, it could be applied to problems where solutions can be represented by permutations (a sequence is a piece of the permutation), like permutation flow-shop problems. For other problems it is required to define an extraction and an injection steps adapted to the problem solved. For the extraction, it is required to replace the sequences learned by a knowledge (which can easily be extracted from a solution) specific to the problem solved. Concerning the injection, it is required to define a procedure for injecting an element of knowledge from a solution into another solution.

Development of Bayesian Models At the end of the execution, the sequences stored in the groups represent the best sequences belonging to solutions from a specific region of the objective space. The frequencies obtained can be turned to probabilities, to generate probabilistic models. These models could then be used to generate feasible solutions with specific trade-offs without running the algorithm again. In particular, these models could be useful for a decision-maker, if they want to generate quickly new solutions respecting some properties.

Adapt the Model with Automatic Algorithm Configuration In the thesis, we integrated the developed model into two classical multi-objective algorithms. However, many components and strategies have been developed in the literature to improve the algorithms used. In particular, the choice of the best components to solve a specific problem can be left to an automatic algorithm configurator (like ParamILS). The knowledge discovery components proposed in this thesis can be proposed to a configurator to be used for solving a specific problem. Indeed, the strategies developed for our model are independent from the algorithm used, and different strategies for extraction and injection can easily be considered by a configurator to increase the possible synergies between the knowledge discovery mechanism and the other components selected.

Appendix A

Relation between Genotype and Phenotype

This appendix is related to Section 4.2 about the analysis of the genotype and the phenotype of solutions to the bVRPTW. We recall that to perform this study we generated an approximation of the solution space for all the instances, and we divided the objective into different regions, from "r0" (best solutions) to "r4" (worst solutions). For each region, for each solution in it, we computed its $k \in \{1, 2, 5, 10, 25, 50, 75, 100\}$ closest neighbors and we evaluated the average similarity between the solution and its neighbors. Summarised results are available in Section 4.2.3.

Table A.1: Average similarity between a solution and its neighborhood of different sizes on instances of category C, R, and RC.

	C1								C2							
	1	2	5	10	25	50	75	100	1	2	5	10	25	50	75	100
r0	39.3	38.5	37.6	37.0	36.4	35.9	35.7	35.5	26.9	25.5	24.1	23.2	22.0	21.1	20.7	20.3
r1	28.6	27.6	26.4	25.6	24.8	24.3	24.0	23.8	24.3	23.2	21.7	20.7	19.3	18.3	17.7	17.2
r2	27.6	26.3	24.6	23.6	22.4	21.7	21.3	21.0	24.5	23.5	21.9	20.9	19.4	18.2	17.4	16.9
r3	29.7	28.0	25.9	24.4	22.8	21.7	21.2	20.8	29.7	28.2	26.3	24.7	22.6	21.0	19.9	19.1
r4	38.2	36.3	33.6	31.4	28.7	26.7	25.6	24.8	40.4	38.6	35.5	33.0	29.4	26.6	25.0	23.9
	R1								R2							
	1	2	5	10	25	50	75	100	1	2	5	10	25	50	75	100
r0	42.4	41.7	40.8	40.0	38.8	37.9	37.2	36.7	28.2	27.7	26.9	26.1	24.8	23.7	23.0	22.4
r1	32.3	31.5	30.5	29.8	28.9	28.3	28.0	27.8	18.7	18.3	17.4	16.7	15.7	14.8	14.3	13.9
r2	29.0	27.9	26.6	25.7	24.5	23.7	23.3	23.0	15.1	14.6	13.6	12.9	11.8	11.1	10.6	10.3
r3	28.8	27.7	26.2	25.1	23.7	22.5	21.9	21.4	18.7	18.1	17.0	16.1	14.8	13.8	13.1	12.6
r4	34.9	33.3	30.9	29.0	26.5	24.6	23.4	22.5	27.5	26.5	24.7	23.2	20.5	18.3	16.9	16.0
	RC1								RC2							
	1	2	5	10	25	50	75	100	1	2	5	10	25	50	75	100
r0	46.1	45.4	44.4	43.5	42.3	41.2	40.3	39.6	27.4	27.1	26.3	25.7	24.7	23.6	22.9	22.3
r1	33.6	33.0	32.1	31.3	30.5	29.8	29.4	29.2	22.3	21.8	21.2	20.6	19.7	18.9	18.3	17.8
r2	29.5	28.6	27.3	26.5	25.4	24.7	24.2	23.9	21.9	21.5	20.7	20.1	19.1	18.1	17.3	16.8
r3	29.9	28.8	27.4	26.4	25.0	24.0	23.4	22.9	25.2	24.8	24.1	23.3	22.1	20.7	19.8	19.1
r4	34.1	32.6	30.2	28.2	26.1	24.4	23.3	22.4	35.9	35.2	33.7	32.4	30.2	28.2	26.8	25.6

Table A.2: Average similarity between a solution and neighborhood of different sizes on R101 instance.

Size	1	2	5	10	25	50	75	100
r0	54.8	54.7	54.4	54.0	53.6	53.3	53.1	52.9
r1	49.1	49.0	48.6	48.3	47.9	47.4	47.1	46.8
r2	42.1	41.8	41.4	40.9	40.3	39.7	39.3	39.0
r3	33.4	33.1	32.5	31.9	31.2	30.4	29.8	29.3
r4	30.4	30.0	29.3	28.6	27.7	27.0	26.5	26.1

Table A.3: Average similarity between a solution and neighborhood of different sizes on R201 instance.

Size	1	2	5	10	25	50	75	100
r0	29.1	28.5	27.8	27.0	26.0	25.2	24.7	24.4
r1	21.1	20.6	19.9	19.2	18.4	17.7	17.3	16.9
r2	18.1	17.6	16.6	15.6	14.2	13.1	12.4	11.9
r3	40.3	39.5	37.7	35.8	32.8	30.1	28.4	27.2
r4	64.3	63.1	61.6	60.0	56.9	54.1	51.8	50.1

Table A.4: Average similarity between a solution and neighborhood of different sizes on C106 instance.

Size	1	2	5	10	25	50	75	100
r0	40.7	40.0	39.3	39.0	38.6	38.2	38.0	37.8
r1	29.3	28.3	27.3	26.6	25.9	25.5	25.2	25.1
r2	30.3	29.0	27.4	26.4	25.2	24.4	24.0	23.7
r3	34.2	32.8	31.2	29.9	28.1	26.8	26.1	25.6
r4	46.3	44.6	41.8	39.6	36.6	34.5	32.9	31.8

Table A.5: Average similarity between a solution and neighborhood of different sizes on RC108 instance.

Size	1	2	5	10	25	50	75	100
r0	54.8	54.9	54.6	54.0	52.5	50.5	48.5	46.6
r1	32.6	32.4	31.7	31.3	30.8	30.2	29.9	29.6
r2	27.0	26.7	26.4	26.1	25.4	24.9	24.5	24.2
r3	32.4	32.0	31.4	30.6	29.3	28.1	27.4	26.7
r4	33.3	31.8	29.7	28.0	26.0	23.9	22.3	21.0

Appendix B

First Knowledge Groups

This appendix is related to Section 4.3, where the concept of knowledge groups is introduced for the first time. The algorithms compared are a reference MOEA/D indicated with $R_{\mathrm{MOEA/D}}$ and two variants hybridized with the knowledge groups $\mathrm{KD1}^{i}_{\mathrm{MOEA/D}}$ and $\mathrm{KD1}^{d}_{\mathrm{MOEA/D}}$. Summarised results are presented in Section 4.3.5.

Table B.1: Average hypervolume obtained on Solomon's R instances.

Instance	c		$\mathrm{KD1}^{i}_{\mathrm{MOEA/D}}$		$\mathrm{KD1}^{d}_{\mathrm{MOEA/D}}$	
	50	100	50	100	50	100
R101	0.435	0.360	0.946	0.974	0.949	0.977
R102	0.664	0.561	0.976	0.986	0.983	0.988
R103	0.773	0.708	0.973	0.985	0.979	0.992
R104	0.865	0.827	0.976	0.983	0.975	0.990
R105	0.554	0.494	0.959	0.975	0.966	0.987
R106	0.823	0.699	0.979	0.988	0.983	0.989
R107	0.848	0.802	0.971	0.986	0.976	0.994
R108	0.819	0.751	0.969	0.985	0.981	0.988
R109	0.751	0.642	0.970	0.974	0.981	0.985
R110	0.827	0.795	0.960	0.985	0.968	0.991
R111	0.841	0.756	0.971	0.981	0.978	0.984
R112	0.864	0.828	0.973	0.986	0.978	0.988
R201	0.589	0.512	0.956	0.965	0.968	0.973
R202	0.703	0.516	0.949	0.971	0.965	0.985
R203	0.739	0.586	0.960	0.957	0.964	0.975
R204	0.803	0.612	0.970	0.962	0.978	0.982
R205	0.724	0.544	0.970	0.972	0.976	0.982
R206	0.780	0.599	0.958	0.972	0.973	0.985
R207	0.806	0.666	0.965	0.960	0.977	0.985
R208	0.626	0.578	0.933	0.956	0.963	0.980
R209	0.715	0.536	0.969	0.976	0.979	0.992
R210	0.761	0.544	0.972	0.966	0.976	0.983
R211	0.759	0.626	0.977	0.991	0.987	0.993

Table B.2: Average hypervolume obtained on Solomon's RC instances.

Instance	$R_{MOEA/D}$		$KD1^i_{MOEA/D}$		$KD1^d_{MOEA/D}$	
	50	100	50	100	50	100
RC101	0.514	0.442	0.988	0.985	0.991	0.985
RC102	0.736	0.660	0.984	0.991	0.993	0.993
RC103	0.835	0.791	0.975	0.985	0.984	0.989
RC104	0.672	0.824	0.984	0.988	0.993	0.991
RC105	0.719	0.593	0.966	0.989	0.977	0.991
RC106	0.643	0.680	0.905	0.985	0.931	0.985
RC107	0.780	0.747	0.962	0.984	0.988	0.988
RC108	0.662	0.838	0.984	0.992	0.989	0.994
RC201	0.591	0.481	0.952	0.973	0.961	0.982
RC202	0.680	0.496	0.950	0.968	0.959	0.982
RC203	0.719	0.648	0.956	0.970	0.964	0.984
RC204	0.619	0.573	0.934	0.964	0.949	0.980
RC205	0.635	0.532	0.957	0.968	0.960	0.981
RC206	0.653	0.552	0.966	0.981	0.979	0.991
RC207	0.712	0.621	0.980	0.986	0.988	0.994
RC208	0.794	0.639	0.994	0.985	0.998	0.989

Table B.3: Average hypervolume obtained on Solomon's C instances.

Instance	$R_{MOEA/D}$		$KD1^i_{MOEA/D}$		$KD1^d_{MOEA/D}$	
	50	100	50	100	50	100
C101	0.508	0.400	1.001	1.001	1.001	1.000
C102	0.684	0.570	0.982	1.001	0.997	1.001
C103	0.788	0.655	0.973	0.992	0.989	1.000
C104	0.731	0.620	0.959	0.981	0.975	0.989
C105	0.606	0.516	1.000	1.001	1.001	1.001
C106	0.577	0.533	1.001	1.001	1.001	1.001
C107	0.651	0.560	0.998	1.001	1.000	1.001
C108	0.761	0.643	0.998	1.001	1.001	1.001
C109	0.758	0.777	0.999	1.001	1.001	1.001
C201	0.404	0.425	0.998	1.001	0.996	1.001
C202	0.569	0.435	0.996	1.001	0.997	1.001
C203	0.540	0.523	0.973	0.997	0.991	1.000
C204	0.808	0.662	0.977	0.990	0.990	0.997
C205	0.514	0.430	0.998	1.001	0.998	1.001
C206	0.477	0.389	0.995	1.001	0.998	1.001
C207	0.597	0.487	0.987	1.001	0.994	1.001
C208	0.522	0.464	0.996	1.001	0.999	1.001

Appendix C

Local Search for the bVRPTW

This appendix is related to Section 4.4, where exploration strategies are developed to design an efficient local search for the bVRPTW. In particular, the *first-best* exploration strategy is developed to select the best move from a subset of the neighborhood. A granular search is also introduced, requiring a metric to measure the proximity between customers. Four algorithms, using our knowledge discovery mechanism, are compared: $\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$, $\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$, $\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$, $\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$. When needed, they are represented by a pair (exploration strategy, granularity metric). Summarized results are presented in Section 4.4.5.

Table C.1: Average hypervolume obtained on Solomon's R instances. Each algorithm is used with the parameters returned by irace.

	50				100			
Instance	(b, d_1)	(fb, d_1)	(b, d_2)	(fb, d_2)	(b, d_1)	(fb, d_1)	(b, d_2)	(fb, d_2)
R101	0.949	0.953	0.951	**0.954**	0.972	**0.974**	0.970	0.973
R102	0.975	**0.978**	0.976	**0.978**	0.981	**0.986**	0.979	0.984
R103	0.978	0.988	0.984	**0.989**	0.984	0.988	0.983	**0.989**
R104	0.640	**0.725**	0.651	0.682	0.972	0.977	0.966	**0.979**
R105	0.989	**0.995**	0.990	0.993	0.991	**0.995**	0.991	**0.995**
R106	0.675	0.657	0.658	**0.710**	0.987	0.989	0.986	**0.990**
R107	0.605	**0.659**	0.608	0.650	0.563	0.586	0.469	**0.651**
R108	0.577	0.747	0.651	**0.754**	0.518	**0.642**	0.408	0.623
R109	0.661	0.748	0.657	**0.759**	0.504	**0.775**	0.451	0.721
R110	0.454	0.511	0.482	**0.532**	0.615	**0.800**	0.438	0.738
R111	0.455	0.562	0.539	**0.663**	0.595	**0.739**	0.573	0.721
R112	0.639	0.695	0.607	**0.735**	0.592	**0.658**	0.431	0.632
R201	0.756	**0.790**	0.775	0.785	0.966	**0.975**	0.969	0.973
R202	0.714	**0.762**	0.745	0.760	0.737	0.803	0.741	**0.806**
R203	0.703	0.743	0.704	**0.744**	0.637	**0.766**	0.675	0.750
R204	0.773	**0.932**	0.784	0.865	0.538	**0.720**	0.522	0.706
R205	0.702	**0.772**	0.733	0.767	0.633	0.729	0.616	**0.741**
R206	0.641	0.678	0.643	**0.681**	0.651	**0.753**	0.626	0.695
R207	0.699	0.775	0.737	**0.796**	0.590	**0.736**	0.565	0.734
R208	0.447	**0.613**	0.467	0.553	0.467	**0.644**	0.443	0.636
R209	0.699	**0.760**	0.682	0.749	0.578	0.730	0.580	**0.744**
R210	0.738	**0.800**	0.771	**0.800**	0.575	**0.720**	0.612	0.701
R211	0.454	0.592	0.552	**0.674**	0.512	**0.780**	0.414	0.726

Table C.2: Average hypervolume obtained on Solomon's RC instances. Each algorithm is used with the parameters returned by irace.

	50				100			
Instance	(d, b, d_1)	(d, fb, d_1)	(d, b, d_2)	(d, fb, d_2)	(d, b, d_1)	(d, fb, d_1)	(d, b, d_2)	(d, fb, d_2)
RC101	0.612	**0.851**	0.565	0.747	0.989	**0.991**	0.987	**0.991**
RC102	0.437	**0.612**	0.454	0.445	0.710	**0.828**	0.686	0.775
RC103	0.535	**0.718**	0.555	0.625	0.634	0.721	0.543	**0.759**
RC104	0.815	0.916	0.870	**0.921**	0.635	**0.637**	0.539	0.625
RC105	0.654	**0.777**	0.622	0.753	0.819	0.874	0.83	**0.881**
RC106	0.347	**0.627**	0.306	0.477	0.544	**0.788**	0.582	0.729
RC107	0.606	**0.783**	0.613	0.640	0.528	**0.607**	0.443	0.573
RC108	0.822	0.799	0.903	**0.905**	0.599	**0.619**	0.435	0.578
RC201	0.813	**0.838**	0.836	0.834	0.974	**0.980**	0.973	**0.980**
RC202	0.680	**0.703**	0.688	0.702	0.671	**0.787**	0.683	0.762
RC203	0.629	**0.675**	0.639	0.665	0.635	0.754	0.652	**0.755**
RC204	0.570	0.604	0.570	**0.624**	0.564	0.689	0.573	**0.701**
RC205	0.690	0.702	0.699	**0.712**	0.789	**0.846**	0.802	**0.846**
RC206	0.670	0.675	0.683	**0.708**	0.665	0.778	0.681	**0.788**
RC207	0.439	0.431	0.432	**0.463**	0.556	0.674	0.567	**0.694**
RC208	0.606	**0.831**	0.627	**0.831**	0.408	0.618	0.361	**0.623**

Table C.3: Average hypervolume obtained on Solomon's C instances. Each algorithm is used with the parameters returned by irace.

	50				100			
Instance	(d, b, d_1)	(d, fb, d_1)	(d, b, d_2)	(d, fb, d_2)	(d, b, d_1)	(d, fb, d_1)	(d, b, d_2)	(d, fb, d_2)
C101	0.404	**0.423**	0.422	0.422	0.904	**1.000**	**1.000**	**1.000**
C102	0.144	**0.202**	0.169	0.188	0.877	**0.991**	0.961	0.927
C103	0.851	**0.938**	0.910	0.906	0.919	**0.944**	0.865	0.896
C104	0.771	**0.950**	0.855	0.872	**0.716**	0.664	0.546	0.673
C105	**0.176**	0.169	0.169	**0.176**	0.924	**1.000**	0.970	0.966
C106	**1.002**	**1.002**	**1.002**	**1.002**	0.927	**1.000**	0.947	0.926
C107	0.443	**0.506**	0.464	**0.506**	0.833	**0.985**	0.777	0.818
C108	0.935	**1.000**	0.869	0.935	0.932	**0.978**	0.769	0.890
C109	0.945	**0.976**	0.849	0.943	0.898	**0.946**	0.783	0.838
C201	0.297	**0.328**	0.306	0.308	**1.000**	**1.000**	**1.000**	**1.000**
C202	**0.267**	0.264	0.266	0.263	0.935	**1.000**	0.935	**1.000**
C203	0.230	**0.317**	0.291	0.289	0.792	**0.887**	0.721	0.800
C204	0.722	**0.782**	0.728	0.766	0.715	**0.884**	0.594	0.866
C205	0.301	0.270	**0.342**	0.305	0.843	**0.977**	0.918	**0.977**
C206	0.309	0.341	**0.342**	0.341	0.969	0.999	0.902	**1.000**
C207	**0.380**	0.228	0.299	0.210	0.953	**0.992**	0.831	0.987
C208	0.724	0.734	**0.739**	**0.739**	0.988	**1.000**	0.965	**1.000**

Table C.4: Gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of category R. For each algorithm, the first column gives the gap with the best solution found. The second column contains the average gap over the 30 runs.

Instance	Size	Optimal value	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
			Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
R101	50	1044.0	0.0	0.3	0.0	0.2	0.0	0.2	0.0	0.1
R102	50	909.0	0.0	0.2	0.0	0.2	0.0	0.4	0.0	0.2
R103	50	772.9	0.0	0.9	0.0	0.3	0.0	0.9	0.0	0.2
R104	50	625.4	0.6	2.9	0.6	2.0	0.2	2.8	0.6	2.5
R105	50	899.3	0.0	0.5	0.0	0.2	0.0	0.5	0.0	0.3
R106	50	793.0	0.0	1.4	0.0	1.5	0.0	1.3	0.0	1.1
R107	50	711.1	0.4	2.5	0.0	2.0	0.6	2.5	0.1	2.2
R108	50	617.7	0.6	3.2	0.0	1.8	0.0	2.6	0.0	1.5
R109	50	786.8	0.0	1.4	0.4	1.0	0.1	1.4	0.0	1.0
R110	50	697.0	0.0	4.0	0.0	3.3	0.8	3.8	0.0	3.1
R111	50	707.2	0.6	2.4	0.0	2.0	0.6	2.0	0.5	1.5
R112	50	630.2	0.8	3.0	0.8	2.6	0.9	3.2	0.8	2.4
Mean gap			0.2	1.9	0.2	1.4	0.3	1.8	0.2	1.3
R201	50	791.9	1.0	2.9	1.0	2.3	0.0	3.0	1.0	2.7
R202	50	698.5	2.2	4.5	2.1	4.1	0.9	3.7	2.1	3.8
R203	50	605.3	1.2	3.7	0.5	3.6	1.2	4.2	0.5	3.3
R204	50	506.4	0.4	2.1	0.4	0.8	0.4	2.2	0.4	1.3
R205	50	690.1	0.9	3.4	1.2	2.9	1.5	3.3	1.6	3.0
R206	50	632.4	0.8	2.8	1.3	2.8	1.3	2.6	0.0	2.4
R207	50	575.5	0.2	3.4	0.2	2.1	0.1	2.6	0.1	1.8
R208	50	489.5	0.0	3.1	0.4	2.1	1.0	3.2	0.2	2.3
R209	50	600.6	0.0	2.9	0.0	2.5	0.8	2.9	0.0	2.7
R210	50	645.6	1.9	3.8	0.9	3.0	1.3	3.3	1.5	3.1
R211	50	535.5	1.1	4.1	1.1	3.1	0.1	3.4	0.5	2.5
Mean gap			0.8	3.3	0.8	2.7	0.8	3.1	0.7	2.6

Instance	Size	Optimal value	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
			Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
R101	100	1637.7	0.2	0.9	0.1	0.2	0.1	0.8	0.1	0.2
R102	100	1466.6	0.7	2.0	0.5	1.0	0.4	2.0	0.0	1.0
R103	100	1208.7	2.2	4.6	1.5	3.6	2.3	4.7	1.9	3.4
R104	100	971.5	4.7	8.4	4.8	7.5	5.3	9.2	3.6	7.2
R105	100	1355.3	0.6	2.2	0.4	1.4	0.4	2.4	0.4	1.4
R106	100	1234.6	0.9	4.1	2.3	3.6	2.6	4.2	1.3	3.4
R107	100	1064.6	1.9	6.2	2.5	6.0	4.2	7.2	3.4	5.3
R108	100	932.1	4.1	8.3	5.1	7.2	5.0	9.3	4.8	7.4
R109	100	1146.9	2.7	5.7	1.6	3.1	2.1	6.2	0.9	3.6
R110	100	1068.0	4.7	7.5	3.7	5.7	6.0	9.2	3.9	6.3
R111	100	1048.7	3.7	7.3	3.1	5.6	5.1	7.5	2.5	5.8
R112	100	948.6	4.0	8.6	4.0	7.7	4.6	10.6	3.5	8.1
Mean gap			2.5	5.5	2.5	4.4	3.2	6.1	2.2	4.4
R201	100	1143.2	4.2	9.5	2.2	5.1	3.3	6.9	2.9	4.9
R202	100	1029.6	6.2	9.6	3.8	6.5	1.5	8.6	3.1	6.0
R203	100	870.8	6.0	11.4	3.4	6.3	3.6	9.0	1.7	6.4
R204	100	731.3	3.3	9.6	3.0	5.9	1.9	9.0	3.4	5.6
R205	100	949.8	3.4	7.1	0.9	4.9	3.5	7.2	0.8	4.0
R206	100	875.9	3.4	6.8	2.6	5.3	2.7	6.9	2.3	6.2
R207	100	794.0	5.0	8.9	3.7	6.0	2.7	9.2	1.8	5.9
R208	100	701.0	4.8	8.4	3.0	6.2	3.7	8.4	2.3	6.2
R209	100	854.8	2.8	7.0	2.2	4.9	3.8	6.8	1.7	4.6
R210	100	900.5	6.1	9.6	4.1	6.6	4.2	8.4	3.0	6.7
R211	100	746.7	5.5	8.6	2.4	5.2	5.4	9.9	2.5	5.9
Mean gap			4.6	8.8	2.8	5.7	3.3	8.2	2.3	5.7

Table C.5: Gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of category RC of size 50. For each algorithm, the first column gives the gap with the best solution found. The second column contains the average gap over the 30 runs.

Instance	Size	Optimal value	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
			Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
RC101	50	944.0	0.0	1.0	0.0	0.0	0.0	1.2	0.0	0.5
RC102	50	822.5	2.0	2.0	0.0	1.4	0.0	1.9	2.0	2.0
RC103	50	710.9	6.1	7.1	0.2	4.3	6.0	6.8	0.4	5.8
RC104	50	545.8	0.0	1.9	0.0	0.7	0.0	1.1	0.0	0.5
RC105	50	855.3	0.0	3.0	0.0	1.6	0.0	3.2	0.0	1.9
RC106	50	723.2	6.3	13.1	0.0	7.6	6.3	13.9	0.6	10.5
RC107	50	642.7	3.3	6.2	0.0	3.4	2.0	6.1	2.3	5.6
RC108	50	598.1	0.0	1.2	0.0	1.4	0.0	0.7	0.0	0.6
Mean gap			2.2	4.4	0.0	2.6	1.8	4.4	0.7	3.4
RC201	50	684.4	0.1	0.1	0.2	0.4	0.1	0.2	0.1	0.1
RC202	50	613.6	0.0	1.3	0.0	0.8	0.0	0.4	0.0	0.2
RC203	50	555.3	0.0	7.6	1.1	3.1	0.7	6.5	0.7	5.4
RC204	50	444.2	0.0	2.9	0.0	3.4	0.6	3.2	0.0	1.7
RC205	50	630.2	0.1	0.9	0.6	1.5	0.0	1.0	0.0	0.5
RC206	50	610.0	0.0	1.6	0.3	2.7	0.0	0.6	0.0	0.6
RC207	50	558.6	0.3	1.8	1.0	2.6	0.7	2.0	0.7	1.6
RC208	50	489.1	0.0	2.4	0.0	1.0	0.0	2.3	0.0	1.0
Mean gap			0.1	2.3	0.4	1.9	0.3	2.0	0.2	1.4

Table C.6: Gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of category RC of size 100. For each algorithm, the first column gives the gap with the best solution found. The second column contains the average gap over the 30 runs.

Instance	Size	Optimal value	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
			Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
RC101	100	1619.8	2.4	4.0	1.5	3.2	1.3	4.2	1.6	3.2
RC102	100	1457.4	2.5	4.8	2.4	3.5	2.1	5.2	2.3	4.2
RC103	100	1258.0	7.2	10.3	7.6	9.5	7.5	11.0	7.3	9.2
RC104	100	1132.3	2.7	8.9	4.9	8.9	6.6	10.5	5.5	9.1
RC105	100	1513.7	4.6	7.0	3.4	5.5	3.3	6.7	2.4	5.3
RC106	100	1372.7	5.4	8.2	3.2	5.6	4.2	7.9	3.7	6.2
RC107	100	1207.8	6.9	11.0	5.6	10.0	8.0	12.1	4.8	10.4
RC108	100	1114.2	5.9	10.3	4.3	10.0	4.8	12.8	5.5	10.7
Mean gap			4.7	8.1	4.1	7.0	4.7	8.8	4.1	7.3
RC201	100	1261.8	2.0	7.5	2.0	4.2	2.3	6.8	1.2	3.9
RC202	100	1092.3	2.4	8.7	2.0	4.0	1.6	7.7	1.4	3.8
RC203	100	923.7	5.5	11.4	2.5	5.7	4.8	9.3	2.1	5.5
RC204	100	783.5	4.1	8.3	1.3	5.0	2.1	7.0	1.5	4.3
RC205	100	1154.0	4.9	10.7	2.3	5.7	2.6	7.6	0.5	4.6
RC206	100	1051.1	3.8	8.0	2.5	5.3	2.9	7.0	2.0	4.7
RC207	100	962.9	1.3	6.9	1.0	5.2	3.5	6.7	2.9	5.0
RC208	100	776.1	4.5	7.8	2.5	5.3	5.0	8.4	0.8	5.3
Mean gap			3.6	8.7	2.0	5.0	3.1	7.6	1.5	4.6

Table C.7: Gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of category C of size 50. For each algorithm, the first column gives the gap with the best solution found. The second column contains the average gap over the 30 runs.

Instance	Size	Optimal value	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
			Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
C101	50	362.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C102	50	361.4	0.0	0.0	0.0	0.2	0.0	0.2	0.0	0.0
C103	50	361.4	0.0	4.0	0.0	1.6	0.0	2.4	0.0	2.5
C104	50	358.0	0.0	6.8	0.0	1.5	0.0	4.4	0.0	3.8
C105	50	362.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C106	50	362.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C107	50	362.4	0.0	0.3	0.0	0.0	0.0	0.3	0.0	0.0
C108	50	362.4	0.0	0.5	0.0	0.0	0.0	1.0	0.0	0.5
C109	50	362.4	0.0	0.5	0.0	0.2	0.0	1.4	0.0	0.6
Mean gap			0.0	1.3	0.0	0.4	0.0	1.1	0.0	0.8
C201	50	360.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C202	50	360.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C203	50	359.8	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0
C204	50	350.1	2.3	4.6	0.0	3.5	0.9	4.4	0.0	3.8
C205	50	359.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C206	50	359.8	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0
C207	50	359.6	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0
C208	50	350.5	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.0
Mean gap			0.3	0.6	0.0	0.6	0.1	0.6	0.0	0.5

Table C.8: Gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of category C of size 100. For each algorithm, the first column gives the gap with the best solution found. The second column contains the average gap over the 30 runs.

Instance	Size	Optimal value	$\text{KD1}^{d,b,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_1}_{\text{MOEA/D}}$		$\text{KD1}^{d,b,d_2}_{\text{MOEA/D}}$		$\text{KD1}^{d,fb,d_2}_{\text{MOEA/D}}$	
			Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
C101	100	827.3	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0
C102	100	827.3	0.0	1.7	0.0	0.2	0.0	0.6	0.0	1.0
C103	100	826.3	0.0	7.3	0.1	5.1	0.0	12.0	0.0	9.3
C104	100	822.9	0.1	10.5	1.6	12.4	0.9	16.7	0.4	12.1
C105	100	827.3	0.0	2.3	0.0	0.0	0.0	0.9	0.0	1.0
C106	100	827.3	0.0	1.4	0.0	0.0	0.0	1.0	0.0	1.4
C107	100	827.3	0.0	2.2	0.0	0.2	0.0	3.0	0.0	2.4
C108	100	827.3	0.0	1.7	0.0	0.6	0.0	5.5	0.0	2.7
C109	100	827.3	0.0	2.7	0.0	1.5	0.0	5.8	0.0	4.4
Mean gap			0.0	3.4	0.2	2.2	0.1	5.1	0.0	3.8
C201	100	589.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C202	100	589.1	0.0	0.3	0.0	0.0	0.0	0.3	0.0	0.0
C203	100	588.7	0.0	1.4	0.0	0.8	0.0	1.9	0.0	1.4
C204	100	588.1	0.6	5.2	0.0	2.1	1.1	7.2	0.0	2.5
C205	100	586.4	0.0	0.7	0.0	0.1	0.0	0.4	0.0	0.1
C206	100	586.0	0.0	0.2	0.0	0.0	0.0	0.5	0.0	0.0
C207	100	585.8	0.0	0.1	0.0	0.0	0.0	0.2	0.0	0.0
C208	100	585.8	0.0	0.2	0.0	0.0	0.0	0.5	0.0	0.0
Mean gap			0.1	1.0	0.0	0.4	0.1	1.4	0.0	0.5

Appendix D

Impact of Local Optima

This appendix is related to Section 4.5, where we investigate the impact of learning from local optima only. Five algorithms are compared: $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$ (reference), $\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$, $\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$, $\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$, $\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$. The hybrid variants can be represented by a pair (solutions used: all or lo, injection strategy: i or d). Three different studies are performed. The first one compares the algorithms with the same parameters, the second one uses the configurations returned by irace, and the last one analyzes the speed-up obtained compared to the reference. Summarized results are available in Section 4.5.3.

Table D.1: Average hypervolume obtained on Solomon's R instances. All variants are executed with the same parameters.

	50					100				
Instance	Ref.	(all, i)	(all, d)	(lo, i)	(lo, d)	Ref.	(all, i)	(all, d)	(lo, i)	(lo, d)
R101	0.795	0.807	0.808	0.807	**0.809**	0.969	0.973	**0.974**	**0.974**	**0.974**
R102	0.971	**0.976**	0.975	**0.976**	0.975	0.970	**0.979**	**0.979**	0.978	**0.979**
R103	0.976	**0.987**	0.985	0.985	0.985	0.962	**0.978**	0.976	**0.978**	0.977
R104	0.475	0.710	**0.776**	0.726	0.775	0.591	0.741	0.746	0.740	**0.761**
R105	0.804	0.885	**0.897**	0.889	0.890	0.832	**0.926**	0.914	0.921	0.906
R106	0.541	0.700	0.707	**0.722**	0.717	0.937	0.966	0.966	**0.967**	0.964
R107	0.633	0.783	0.798	0.802	**0.821**	0.283	0.548	**0.585**	0.558	0.538
R108	0.518	0.800	0.762	0.796	**0.801**	0.211	0.566	0.545	**0.572**	**0.572**
R109	0.571	0.742	0.745	**0.749**	0.725	0.331	0.683	**0.741**	0.675	0.697
R110	0.466	0.591	0.616	**0.627**	0.618	0.211	0.486	**0.514**	0.491	0.510
R111	0.625	0.769	0.772	0.735	**0.779**	0.146	0.497	0.505	0.487	**0.510**
R112	0.513	**0.730**	0.709	0.708	0.704	0.154	**0.501**	0.463	0.489	0.489
R201	0.751	0.747	**0.748**	**0.748**	0.747	0.720	**0.749**	0.741	0.748	0.733
R202	0.711	0.726	**0.730**	0.727	**0.730**	0.704	0.743	0.741	**0.754**	0.742
R203	0.635	**0.680**	0.662	**0.680**	0.669	0.674	0.732	0.731	0.717	**0.739**
R204	0.862	0.898	0.916	**0.928**	0.923	0.393	0.517	0.482	**0.544**	0.506
R205	0.673	0.736	0.735	**0.744**	0.714	0.558	0.665	0.666	0.664	**0.667**
R206	0.646	0.679	0.678	**0.683**	0.679	0.585	**0.726**	0.720	0.705	0.714
R207	0.713	0.749	0.741	**0.754**	0.745	0.463	0.592	0.602	**0.621**	0.596
R208	0.621	**0.711**	0.697	0.704	0.677	0.256	0.361	0.389	**0.394**	0.342
R209	0.704	**0.746**	0.728	0.727	0.732	0.469	0.602	0.578	0.603	**0.604**
R210	0.668	0.728	0.732	0.731	**0.753**	0.556	**0.644**	0.640	0.636	0.639
R211	0.568	0.576	**0.645**	0.607	0.607	0.263	0.455	0.459	0.439	**0.479**

Table D.2: Average hypervolume obtained on Solomon's RC instances of size 50. All variants are executed with the same parameters.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	50	0.774	**0.918**	0.913	0.911	0.899
RC102	50	0.526	0.751	0.715	**0.753**	0.704
RC103	50	0.471	0.685	0.670	0.646	**0.753**
RC104	50	0.651	**0.894**	0.893	0.865	0.842
RC105	50	0.586	0.768	0.791	**0.812**	0.779
RC106	50	0.460	**0.698**	0.676	0.662	0.681
RC107	50	0.671	0.811	0.878	**0.893**	0.842
RC108	50	0.452	0.891	0.899	**0.924**	0.921
RC201	50	0.789	0.798	0.798	**0.800**	0.799
RC202	50	0.688	**0.717**	0.716	0.714	0.711
RC203	50	0.648	**0.691**	0.690	0.687	0.690
RC204	50	0.531	0.626	**0.627**	0.625	0.621
RC205	50	0.682	**0.710**	0.706	0.708	0.706
RC206	50	0.611	0.656	**0.665**	0.661	0.659
RC207	50	0.573	0.645	**0.653**	0.646	0.649
RC208	50	0.419	0.880	0.879	0.904	**0.911**

Table D.3: Average hypervolume obtained on Solomon's RC instances of size 100. All variants are executed with the same parameters.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	100	0.724	**0.867**	0.856	0.858	0.859
RC102	100	0.446	**0.780**	0.766	0.768	0.772
RC103	100	0.530	0.795	**0.802**	0.793	0.801
RC104	100	0.168	0.583	0.575	**0.593**	0.581
RC105	100	0.578	0.815	0.810	**0.820**	0.795
RC106	100	0.254	**0.666**	0.630	0.641	**0.666**
RC107	100	0.170	0.553	**0.574**	0.542	0.573
RC108	100	0.121	0.532	**0.574**	0.532	0.534
RC201	100	0.866	**0.894**	**0.894**	0.892	0.891
RC202	100	0.637	**0.734**	0.718	0.729	**0.734**
RC203	100	0.571	0.647	0.642	0.651	**0.658**
RC204	100	0.448	0.596	**0.619**	0.596	0.614
RC205	100	0.774	**0.810**	0.805	0.809	0.802
RC206	100	0.581	0.710	**0.716**	0.706	0.714
RC207	100	0.379	0.555	0.590	**0.594**	0.564
RC208	100	0.195	0.459	**0.496**	0.471	0.468

Table D.4: Average hypervolume obtained on Solomon's C instances of size 50. All variants are executed with the same parameters.

Instance	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
C101	50	0.419	0.427	**0.428**	**0.428**	0.427
C102	50	0.697	0.900	0.897	**0.907**	0.904
C103	50	0.431	0.923	**0.926**	0.923	0.919
C104	50	0.510	**0.960**	0.945	0.956	0.955
C105	50	0.754	**0.925**	**0.925**	**0.925**	**0.925**
C106	50	**0.518**	**0.518**	**0.518**	**0.518**	**0.518**
C107	50	0.584	**0.791**	0.790	0.788	0.788
C108	50	0.682	**1.001**	**1.001**	0.988	**1.001**
C109	50	0.424	**0.998**	0.997	0.990	0.990
C201	50	0.501	0.504	0.484	**0.520**	0.519
C202	50	0.713	**0.737**	**0.737**	**0.737**	**0.737**
C203	50	0.495	**0.554**	0.550	0.551	0.552
C204	50	0.481	0.748	0.750	0.736	**0.754**
C205	50	0.388	**0.400**	0.399	0.399	**0.400**
C206	50	0.437	**0.461**	0.460	0.459	0.458
C207	50	0.386	0.429	0.440	**0.445**	0.430
C208	50	0.713	0.761	0.751	**0.766**	0.761

Table D.5: Average hypervolume obtained on Solomon's C instances of size 100. All variants are executed with the same parameters.

Instance	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
C101	100	0.770	**1.001**	**1.001**	**1.001**	**1.001**
C102	100	0.286	0.977	0.990	0.990	**0.995**
C103	100	0.250	0.809	**0.833**	0.831	0.821
C104	100	0.173	**0.667**	0.665	0.650	0.685
C105	100	0.427	**1.001**	**1.001**	**1.001**	**1.001**
C106	100	0.362	**1.001**	**1.001**	**1.001**	**1.001**
C107	100	0.329	**1.001**	**1.001**	**1.001**	**1.001**
C108	100	0.281	**1.001**	**1.001**	0.997	0.993
C109	100	0.151	0.961	0.978	**0.985**	0.961
C201	100	**1.001**	**1.001**	**1.001**	**1.001**	**1.001**
C202	100	0.887	**1.001**	**1.001**	**1.001**	**1.001**
C203	100	0.405	0.966	**0.970**	0.949	0.968
C204	100	0.367	**0.874**	0.859	0.862	0.863
C205	100	0.771	**1.001**	**1.001**	**1.001**	**1.001**
C206	100	0.513	**1.001**	1.000	0.999	**1.001**
C207	100	0.646	0.999	**1.001**	0.998	0.999
C208	100	0.505	**1.001**	0.998	0.993	0.990

Table D.6: Average hypervolume obtained on Solomon's R instances of size 50. All variants are executed with the configurations returned by irace.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
R101	50	0.795	0.819	**0.822**	0.819	0.818
R102	50	0.971	**0.981**	0.98	0.979	0.979
R103	50	0.976	**0.991**	**0.991**	0.987	0.989
R104	50	0.475	0.888	**0.889**	0.833	0.749
R105	50	0.804	0.914	**0.916**	0.905	0.911
R106	50	0.541	**0.783**	0.773	0.740	0.735
R107	50	0.633	**0.854**	0.821	0.793	0.779
R108	50	0.518	0.837	**0.85**	0.834	0.778
R109	50	0.571	0.777	**0.833**	0.785	0.775
R110	50	0.466	**0.720**	0.710	0.676	0.590
R111	50	0.625	0.840	**0.855**	0.803	0.789
R112	50	0.513	0.815	**0.819**	0.716	0.719
R201	50	0.751	**0.803**	**0.803**	0.789	0.790
R202	50	0.711	0.771	**0.776**	0.766	0.764
R203	50	0.635	**0.760**	0.752	0.739	0.743
R204	50	0.862	0.944	**0.948**	0.906	0.934
R205	50	0.673	0.785	**0.796**	0.771	0.778
R206	50	0.646	**0.738**	0.721	0.701	0.704
R207	50	0.713	0.807	**0.816**	0.796	0.792
R208	50	0.621	0.698	**0.711**	0.666	0.697
R209	50	0.704	0.783	**0.794**	0.762	0.765
R210	50	0.668	0.789	**0.797**	0.780	0.793
R211	50	0.568	**0.700**	0.665	0.600	0.500

Table D.7: Average hypervolume obtained on Solomon's R instances of size 100. All variants are executed with the configurations returned by irace.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
R101	100	0.969	0.975	**0.977**	**0.977**	**0.977**
R102	100	0.970	0.986	0.987	0.987	**0.988**
R103	100	0.962	0.986	**0.991**	0.989	0.989
R104	100	0.591	**0.891**	0.89	0.888	0.886
R105	100	0.832	0.919	0.944	0.951	**0.955**
R106	100	0.937	0.974	**0.979**	0.976	0.978
R107	100	0.283	0.702	**0.771**	0.752	0.733
R108	100	0.211	0.764	0.778	**0.813**	0.802
R109	100	0.331	0.663	0.828	0.843	**0.860**
R110	100	0.211	0.641	0.709	0.723	**0.754**
R111	100	0.146	0.706	0.779	0.738	**0.789**
R112	100	0.154	0.686	**0.741**	0.713	0.705
R201	100	0.720	0.720	0.752	0.774	**0.800**
R202	100	0.704	0.757	0.796	0.800	**0.822**
R203	100	0.674	0.745	0.781	0.805	**0.823**
R204	100	0.393	0.646	**0.724**	0.685	0.714
R205	100	0.558	0.667	0.704	**0.753**	0.731
R206	100	0.585	0.742	0.774	0.789	**0.799**
R207	100	0.463	0.649	0.707	0.744	**0.758**
R208	100	0.256	0.530	0.622	**0.633**	0.626
R209	100	0.469	0.632	0.720	**0.738**	0.716
R210	100	0.556	0.692	0.763	0.751	**0.791**
R211	100	0.263	0.626	0.668	0.676	**0.698**

Table D.8: Average hypervolume obtained on Solomon's RC instances of size 50. All variants are executed with the configurations returned by irace.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	50	0.774	0.953	0.949	0.949	**0.958**
RC102	50	0.526	0.781	0.795	0.781	**0.837**
RC103	50	0.471	**0.822**	0.787	0.764	0.718
RC104	50	0.651	**0.945**	0.941	0.938	0.925
RC105	50	0.586	**0.923**	0.919	0.891	0.867
RC106	50	0.460	0.737	**0.815**	0.697	0.675
RC107	50	0.671	0.929	**0.931**	0.909	0.861
RC108	50	0.452	0.982	**0.987**	0.957	0.930
RC201	50	0.789	0.803	**0.804**	0.794	0.800
RC202	50	0.688	**0.748**	0.747	0.739	0.740
RC203	50	0.648	0.722	**0.735**	0.705	0.708
RC204	50	0.531	0.672	**0.685**	0.669	0.658
RC205	50	0.682	**0.744**	0.742	0.732	0.731
RC206	50	0.611	**0.727**	0.722	0.703	0.698
RC207	50	0.573	**0.716**	0.712	0.690	0.694
RC208	50	0.419	0.954	**0.956**	0.936	0.881

Table D.9: Average hypervolume obtained on Solomon's RC instances of size 100. All variants are executed with the configurations returned by irace.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	100	0.724	0.872	0.908	0.924	**0.929**
RC102	100	0.446	0.759	0.877	0.891	**0.917**
RC103	100	0.530	0.823	0.882	0.914	**0.915**
RC104	100	0.168	0.666	0.773	0.756	**0.813**
RC105	100	0.578	0.829	0.888	0.898	**0.91**
RC106	100	0.254	0.671	0.81	0.793	**0.826**
RC107	100	0.170	0.599	0.738	0.751	**0.773**
RC108	100	0.121	0.618	0.69	0.747	**0.790**
RC201	100	0.866	0.870	0.899	0.910	**0.924**
RC202	100	0.637	0.682	0.780	0.811	**0.848**
RC203	100	0.571	0.643	0.740	0.769	**0.806**
RC204	100	0.448	0.612	0.727	0.775	**0.779**
RC205	100	0.774	0.784	0.825	0.852	**0.866**
RC206	100	0.581	0.710	0.768	0.799	**0.826**
RC207	100	0.379	0.558	0.675	0.723	**0.730**
RC208	100	0.195	0.496	0.629	0.680	**0.741**

Table D.10: Average hypervolume obtained on Solomon's C instances of size 50. All variants are executed with the configurations returned by irace.

Instance	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
C101	50	0.419	**0.426**	0.425	0.423	0.423
C102	50	0.697	0.910	**0.911**	0.897	0.904
C103	50	0.431	0.984	**0.985**	0.953	0.948
C104	50	0.510	**0.993**	0.992	0.970	0.973
C105	50	0.754	**0.925**	**0.925**	**0.925**	0.924
C106	50	**0.518**	**0.518**	**0.518**	0.501	**0.518**
C107	50	0.584	0.788	**0.789**	0.786	0.785
C108	50	0.682	**1.002**	**1.002**	**1.002**	**1.002**
C109	50	0.424	**1.002**	**1.002**	**1.002**	0.994
C201	50	0.501	0.500	**0.512**	0.485	0.506
C202	50	0.713	0.733	**0.734**	**0.734**	0.733
C203	50	0.495	0.527	0.527	**0.532**	0.525
C204	50	0.481	**0.804**	0.800	0.701	0.743
C205	50	0.388	**0.399**	0.398	0.397	0.329
C206	50	0.437	**0.462**	**0.462**	0.457	0.46
C207	50	0.386	0.518	**0.53**	0.446	0.384
C208	50	0.713	**0.776**	**0.776**	0.766	0.771

Table D.11: Average hypervolume obtained on Solomon's C instances of size 100. All variants are executed with the configurations returned by irace.

Instance	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
C101	100	0.770	**1.002**	**1.002**	**1.002**	**1.002**
C102	100	0.286	0.895	0.921	0.960	**0.999**
C103	100	0.250	0.818	0.86	0.859	**0.928**
C104	100	0.173	0.733	**0.835**	0.762	0.832
C105	100	0.427	0.988	0.994	0.990	**1.002**
C106	100	0.362	0.995	0.99	1.001	**1.002**
C107	100	0.329	0.971	0.941	0.99	**0.998**
C108	100	0.281	0.951	0.974	0.958	**0.991**
C109	100	0.151	0.922	0.957	0.929	**0.973**
C201	100	**1.002**	**1.002**	**1.002**	**1.002**	**1.002**
C202	100	0.887	0.85	**1.002**	0.979	**1.002**
C203	100	0.405	0.687	0.866	0.909	**0.945**
C204	100	0.367	0.725	0.836	0.936	**0.944**
C205	100	0.771	0.85	0.933	0.976	**0.986**
C206	100	0.513	0.871	0.923	0.985	**1.0**
C207	100	0.646	0.822	0.954	0.996	**1.001**
C208	100	0.505	0.819	0.96	0.993	**0.999**

Table D.12: Average execution time needed to reach 95% of the mean hypervolume obtained with $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$ on Solomon's R instances of size 50.

Instance	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
R101	50	113.66	27.77	35.16	**22.78**	25.54
R102	50	38.13	16.84	15.05	11.12	**10.74**
R103	50	56.57	21.15	19.95	**13.84**	15.81
R104	50	276.28	61.0	71.22	**58.46**	68.86
R105	50	234.13	63.46	83.11	78.56	**57.66**
R106	50	253.48	45.04	52.83	**41.07**	41.66
R107	50	251.58	**55.53**	81.52	65.12	88.14
R108	50	244.63	**55.47**	71.86	57.8	80.93
R109	50	237.72	70.83	63.68	**56.67**	65.47
R110	50	233.94	45.45	53.82	**35.47**	51.12
R111	50	238.23	64.54	65.63	57.92	**53.87**
R112	50	253.81	**53.36**	72.67	57.36	58.74
R201	50	163.38	47.86	49.78	**42.32**	44.42
R202	50	185.45	48.77	60.9	**39.57**	44.45
R203	50	212.47	54.55	66.38	46.31	**42.77**
R204	50	243.15	**81.57**	112.17	99.63	95.51
R205	50	237.54	70.41	**67.27**	68.88	68.13
R206	50	233.53	**72.55**	108.82	121.05	89.35
R207	50	238.01	80.2	90.0	**75.27**	87.72
R208	50	243.07	**98.98**	123.8	99.38	125.43
R209	50	221.66	86.29	105.03	**83.21**	103.1
R210	50	226.49	69.85	73.31	63.67	**55.48**
R211	50	234.52	133.18	158.35	**114.85**	149.06

Table D.13: Average execution time needed to reach 95% of the mean hypervolume obtained with $\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$ on Solomon's R instances of size 100.

Instance	Size	$\mathrm{R}^{fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{all,d,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,i,fb,d_1}_{\mathrm{MOEA/D}}$	$\mathrm{KD1}^{lo,d,fb,d_1}_{\mathrm{MOEA/D}}$
R101	100	94.8	48.38	**33.61**	42.48	49.32
R102	100	104.97	54.62	**40.03**	45.18	53.86
R103	100	120.28	67.6	**56.95**	59.31	65.29
R104	100	507.08	182.86	**139.5**	167.5	219.54
R105	100	417.55	166.74	**136.19**	147.03	171.7
R106	100	115.91	100.39	74.26	**54.35**	57.32
R107	100	517.85	188.89	**151.62**	177.07	223.41
R108	100	529.36	176.93	**136.19**	158.71	209.27
R109	100	531.84	198.71	**154.29**	187.56	250.59
R110	100	528.5	191.24	**148.07**	187.17	232.35
R111	100	528.84	180.42	**139.9**	163.28	222.06
R112	100	561.27	183.67	**145.15**	171.63	223.26
R201	100	409.56	180.19	**163.01**	168.17	218.81
R202	100	436.48	189.51	**170.48**	190.18	234.76
R203	100	484.79	**200.92**	206.56	229.46	256.97
R204	100	514.52	265.52	236.61	**223.87**	287.1
R205	100	534.01	220.02	216.45	**209.64**	266.06
R206	100	490.44	232.34	**212.27**	217.3	247.83
R207	100	511.93	251.3	**219.46**	223.81	294.29
R208	100	537.5	237.13	**222.93**	244.16	302.03
R209	100	527.32	239.54	**191.71**	217.33	267.44
R210	100	503.8	253.1	239.9	**215.48**	256.21
R211	100	530.92	288.66	**214.3**	252.5	297.81

Table D.14: Average execution time needed to reach 95% of the mean hypervolume obtained with $R^{fb,d_1}_{\text{MOEA/D}}$ on Solomon's RC instances of size 50.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	50	251.22	26.74	35.22	**22.69**	26.05
RC102	50	248.88	26.22	36.12	41.21	**24.3**
RC103	50	245.07	29.21	40.75	**23.09**	29.7
RC104	50	215.03	**41.07**	44.15	55.61	44.27
RC105	50	258.35	33.29	38.79	**26.02**	33.59
RC106	50	255.98	57.46	**56.18**	72.93	66.6
RC107	50	238.45	55.98	66.25	**54.46**	63.14
RC108	50	251.69	27.99	39.63	**24.23**	26.82
RC201	50	121.09	35.1	44.17	35.19	**34.93**
RC202	50	168.25	**37.83**	50.9	38.93	41.02
RC203	50	208.45	**53.4**	64.73	61.22	63.29
RC204	50	241.97	48.95	62.49	48.38	**46.18**
RC205	50	166.8	34.37	45.31	**33.44**	39.41
RC206	50	195.95	32.9	42.89	**30.37**	31.58
RC207	50	226.69	**36.55**	48.58	38.62	42.47
RC208	50	225.72	45.78	56.01	**33.75**	35.65

Table D.15: Average execution time needed to reach 95% of the mean hypervolume obtained with $R^{fb,d_1}_{\text{MOEA/D}}$ on Solomon's RC instances of size 100.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	100	459.63	171.04	**118.94**	123.03	139.29
RC102	100	525.67	176.14	133.28	**129.55**	151.57
RC103	100	491.42	162.67	**122.84**	131.46	160.28
RC104	100	537.93	157.5	**116.54**	139.31	156.38
RC105	100	504.71	159.5	**132.1**	135.08	166.73
RC106	100	512.99	156.78	**128.07**	145.97	173.86
RC107	100	536.45	170.84	**128.46**	146.69	179.56
RC108	100	550.81	154.19	**123.81**	138.51	165.56
RC201	100	321.33	150.72	139.58	**120.71**	127.63
RC202	100	454.12	182.46	**171.28**	177.57	197.99
RC203	100	494.59	226.26	**183.27**	191.32	212.07
RC204	100	528.63	234.94	204.57	**201.65**	248.29
RC205	100	393.99	184.63	171.61	**167.56**	177.77
RC206	100	515.3	205.34	186.87	**179.9**	231.36
RC207	100	536.0	222.42	200.51	**190.04**	239.64
RC208	100	551.53	211.24	**169.04**	176.2	199.39

Table D.16: Average execution time needed to reach 95% of the mean hypervolume obtained with $R^{fb,d_1}_{\text{MOEA/D}}$ on Solomon's C instances of size 50.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
C101	50	147.27	38.88	44.29	**38.51**	78.82
C102	50	218.45	36.31	43.93	**35.63**	39.63
C103	50	264.2	35.8	51.24	**34.92**	37.03
C104	50	242.72	27.2	40.71	**22.31**	28.53
C105	50	188.1	**26.17**	36.63	37.44	31.05
C106	50	101.68	**22.44**	31.1	23.73	25.59
C107	50	237.5	**27.04**	35.9	31.16	29.19
C108	50	240.17	**25.33**	37.09	25.34	27.49
C109	50	248.66	24.09	34.8	**21.68**	23.82
C201	50	186.71	218.44	**137.42**	205.02	159.9
C202	50	153.44	40.52	48.93	**30.51**	38.2
C203	50	223.63	59.6	73.42	53.56	**49.9**
C204	50	244.02	46.89	61.71	**37.21**	37.51
C205	50	169.62	**37.16**	52.65	41.52	140.36
C206	50	209.6	37.23	50.61	**34.7**	34.9
C207	50	229.29	**74.69**	88.92	215.7	206.15
C208	50	175.01	39.56	43.48	39.62	**32.89**

Table D.17: Average execution time needed to reach 95% of the mean hypervolume obtained with $R^{fb,d_1}_{\text{MOEA/D}}$ on Solomon's C instances of size 100.

Instance	Size	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
C101	100	513.51	134.09	**92.13**	121.42	156.13
C102	100	553.78	144.83	**104.3**	125.58	158.76
C103	100	511.84	157.78	**112.74**	134.91	174.88
C104	100	528.38	156.24	**114.93**	150.93	179.05
C105	100	548.59	132.0	**92.24**	111.13	148.03
C106	100	555.35	136.35	**94.14**	121.46	156.73
C107	100	540.11	125.68	**87.83**	112.14	148.66
C108	100	501.1	136.18	**95.62**	128.09	150.14
C109	100	492.84	136.33	**95.34**	122.46	153.5
C201	100	99.88	85.8	62.8	55.22	**45.53**
C202	100	498.43	170.43	**125.07**	146.22	182.11
C203	100	547.72	202.82	**162.64**	167.17	221.86
C204	100	529.17	216.21	**182.17**	183.9	238.21
C205	100	499.81	148.58	**108.2**	132.54	159.76
C206	100	536.51	155.16	**115.0**	141.3	174.26
C207	100	499.89	163.65	**118.62**	141.11	173.07
C208	100	543.67	170.49	**114.48**	136.63	180.19

Table D.18: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of category R of size 50.

Instance	Size	Opt.	$\text{R}^{fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$
R101	50	1044.0	0.80	0.11	0.16	0.19	0.19
R102	50	909.0	2.19	0.00	0.05	0.06	0.19
R103	50	772.9	2.62	0.10	0.12	0.39	0.31
R104	50	625.4	4.84	1.05	0.98	1.44	2.04
R105	50	899.3	1.57	0.12	0.04	0.17	0.16
R106	50	793.0	2.86	0.94	1.26	1.55	1.52
R107	50	711.1	3.76	1.05	1.51	1.90	2.00
R108	50	617.7	4.32	1.41	1.24	1.36	1.78
R109	50	786.8	1.96	1.05	0.78	1.01	1.04
R110	50	697.0	4.23	1.77	1.85	2.18	3.31
R111	50	707.2	3.43	1.48	1.36	1.84	1.96
R112	50	630.2	4.55	1.74	1.70	2.66	2.63
R201	50	791.9	4.63	2.15	2.45	2.30	2.25
R202	50	698.5	6.68	3.93	3.79	3.75	4.11
R203	50	605.3	7.68	3.10	3.49	3.95	3.65
R204	50	506.4	1.63	0.68	0.68	1.06	0.78
R205	50	690.1	5.26	2.95	2.67	2.62	2.87
R206	50	632.4	3.79	2.03	2.27	2.77	2.74
R207	50	575.5	4.47	1.94	1.96	2.02	2.11
R208	50	492.2	2.79	1.79	1.89	2.17	2.09
R209	50	600.6	4.39	2.59	2.14	2.90	2.47
R210	50	645.6	6.13	3.09	2.96	3.24	2.98
R211	50	535.5	2.67	1.85	2.07	2.46	3.08

Table D.19: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of class R of size 100.

Instance	Size	Opt.	$\text{R}^{fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{all,d,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$\text{KD1}^{lo,d,fb,d_1}_{\text{MOEA/D}}$
R101	100	1637.7	1.48	0.62	0.37	0.30	0.21
R102	100	1466.6	4.04	1.44	1.20	1.25	0.96
R103	100	1208.7	9.24	4.53	3.49	3.84	3.60
R104	100	971.5	16.65	7.37	7.41	7.41	7.50
R105	100	1355.3	5.26	2.55	1.76	1.48	1.35
R106	100	1234.6	8.52	4.17	3.52	3.95	3.62
R107	100	1064.6	12.67	6.54	5.44	5.79	6.01
R108	100	932.1	17.11	7.87	7.59	7.05	7.21
R109	100	1146.9	8.39	5.05	3.36	3.17	3.06
R110	100	1068.0	12.19	7.04	6.24	6.06	5.66
R111	100	1048.7	13.07	6.56	5.68	6.16	5.58
R112	100	948.6	16.69	8.05	7.15	7.61	7.74
R201	100	1143.2	7.28	8.90	7.20	6.01	5.06
R210	100	900.5	9.90	8.81	7.34	7.30	6.56
R211	100	746.7	9.49	5.93	5.51	5.44	5.22
R202	100	1029.6	9.50	9.72	8.03	7.50	6.47
R203	100	870.8	10.39	9.66	8.22	7.46	6.34
R204	100	731.3	10.76	7.47	6.02	6.65	5.85
R205	100	949.8	8.07	6.49	5.62	4.54	4.93
R206	100	875.9	9.55	6.62	5.90	5.63	5.31
R207	100	794.0	11.42	8.59	7.35	6.41	5.96
R208	100	701.0	10.36	7.40	6.24	6.16	6.21
R209	100	854.8	8.68	6.37	4.97	4.56	4.86

Table D.20: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of class RC of size 50.

Instance	Size	Opt.	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	50	944.0	3.33	0.11	0.18	0.19	0.03
RC102	50	822.5	4.98	1.86	1.75	1.91	1.35
RC103	50	710.9	8.23	2.73	3.30	3.65	4.36
RC104	50	545.8	5.01	0.22	0.32	0.28	0.65
RC105	50	855.3	6.82	0.55	0.63	1.15	1.59
RC106	50	723.2	12.44	6.13	4.32	7.06	7.56
RC107	50	642.7	8.03	1.78	1.73	2.27	3.41
RC108	50	598.1	10.44	0.38	0.28	0.85	1.36
RC201	50	684.4	2.82	0.27	0.34	0.41	0.40
RC202	50	613.6	7.47	0.47	0.27	1.36	0.83
RC203	50	555.3	8.21	3.46	2.06	3.72	3.08
RC204	50	444.2	11.62	2.81	2.17	3.34	3.43
RC205	50	630.2	8.46	1.19	1.03	1.74	1.53
RC206	50	610.0	7.58	1.31	1.21	2.05	2.72
RC207	50	558.6	7.33	2.23	2.32	2.55	2.62
RC208	50	489.1	4.96	0.41	0.39	0.56	1.03

Table D.21: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of class RC of size 100.

Instance	Size	Opt.	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
RC101	100	1619.8	10.45	5.66	4.36	3.59	3.24
RC102	100	1457.4	14.88	7.73	4.69	4.33	3.54
RC103	100	1258.0	20.82	12.31	10.55	9.53	9.51
RC104	100	1132.3	25.36	12.65	9.89	10.31	8.87
RC105	100	1513.7	13.86	7.78	6.14	5.92	5.52
RC106	100	1372.7	14.88	8.13	5.81	6.10	5.56
RC107	100	1207.8	22.31	13.52	10.69	10.42	9.96
RC108	100	1114.2	28.19	14.70	12.75	11.20	10.03
RC201	100	1261.8	8.59	10.62	7.53	5.45	4.17
RC202	100	1092.3	11.74	12.85	7.20	5.87	3.95
RC203	100	923.7	15.11	14.10	8.89	7.82	5.67
RC204	100	783.5	13.12	10.10	6.90	5.44	4.95
RC205	100	1154.0	11.39	13.32	9.67	7.08	5.68
RC206	100	1051.1	10.86	9.26	7.20	6.45	5.34
RC207	100	962.9	11.12	8.46	6.32	5.50	5.20
RC208	100	776.1	14.24	9.34	7.16	6.32	5.33

Table D.22: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of class C of size 50.

Instance	Size	Opt.	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
C101	50	362.4	0.00	0.00	0.00	0.00	0.00
C102	50	361.4	8.64	0.00	0.00	0.49	0.24
C103	50	361.4	21.42	0.24	0.22	1.42	1.60
C104	50	358.0	27.21	0.42	0.46	1.67	1.51
C105	50	362.4	4.98	0.00	0.00	0.00	0.00
C106	50	362.4	0.00	0.00	0.00	0.02	0.00
C107	50	362.4	7.80	0.00	0.00	0.00	0.00
C108	50	362.4	10.15	0.00	0.00	0.00	0.00
C109	50	362.4	17.57	0.00	0.00	0.00	0.24
C201	50	360.2	0.00	0.00	0.00	0.00	0.00
C202	50	360.2	0.78	0.00	0.00	0.00	0.00
C203	50	359.8	2.90	0.04	0.05	0.04	0.07
C204	50	350.1	7.31	2.68	2.67	4.23	3.50
C205	50	359.8	0.46	0.02	0.00	0.02	0.00
C206	50	359.8	1.50	0.00	0.00	0.14	0.19
C207	50	359.6	2.35	0.06	0.04	0.24	0.52
C208	50	350.5	1.87	0.00	0.00	0.00	0.08

Table D.23: Average gaps (%) obtained for the total transportation cost objective, relatively to the best-known on instances of class C of size 100.

Instance	Size	Opt.	$R^{fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{all,d,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,i,fb,d_1}_{\text{MOEA/D}}$	$KD1^{lo,d,fb,d_1}_{\text{MOEA/D}}$
C101	100	827.3	2.63	0.00	0.00	0.00	0.00
C102	100	827.3	35.72	5.37	4.08	2.11	0.15
C103	100	826.3	50.71	12.58	9.75	9.81	5.13
C104	100	822.9	54.15	18.75	12.26	16.91	12.40
C105	100	827.3	27.95	0.67	0.41	0.57	0.00
C106	100	827.3	26.86	0.28	0.49	0.04	0.00
C107	100	827.3	35.52	1.69	3.30	0.67	0.23
C108	100	827.3	38.17	2.72	1.51	2.34	0.57
C109	100	827.3	43.53	4.12	2.34	3.75	1.49
C201	100	589.1	0.00	0.00	0.00	0.00	0.00
C202	100	589.1	0.79	1.04	0.00	0.16	0.00
C203	100	588.7	8.18	4.39	1.89	1.29	0.79
C204	100	588.1	22.64	10.01	6.01	2.40	2.11
C205	100	586.4	1.63	1.07	0.49	0.18	0.12
C206	100	586.0	4.08	1.09	0.66	0.14	0.01
C207	100	585.8	3.26	1.65	0.43	0.05	0.01
C208	100	585.8	4.77	1.84	0.43	0.09	0.03

Appendix E

Enhanced Knowledge Groups

This appendix is related to Section 5.2, where an enhancement of the knowledge groups is proposed. In particular, with the new construction, the number of knowledge groups can be controlled and they are more independent from MOEA/D. We compared seven algorithms using different numbers of knowledge groups, and with different injection strategies: $\text{KD2}^{1}_{\text{MOEA/D}}$, $\text{KD2}^{3,s_I^i}_{\text{MOEA/D}}$, $\text{KD2}^{3,s_I^d}_{\text{MOEA/D}}$, $\text{KD2}^{5,s_I^i}_{\text{MOEA/D}}$, $\text{KD2}^{5,s_I^d}_{\text{MOEA/D}}$, $\text{KD2}^{M,s_I^i}_{\text{MOEA/D}}$, and $\text{KD2}^{M,s_I^d}_{\text{MOEA/D}}$. Summarized results are presented in Section 5.2.6.

Table E.1: Average hypervolume obtained on Solomon's R instances of size 100. Each variant is represented by a pair (number of groups, injection strategy).

Category	$(1,-)$	$(3,s_I^i)$	$(3,s_I^d)$	$(5,s_I^i)$	$(5,s_I^d)$	(M,s_I^i)	(M,s_I^d)
R101	0.892	0.878	0.878	0.893	0.879	0.884	0.884
R102	0.901	0.861	0.883	0.913	0.872	0.895	0.892
R103	0.898	0.854	0.858	0.915	0.854	0.895	0.885
R104	0.767	0.637	0.792	0.826	0.696	0.741	0.777
R105	0.882	0.757	0.801	0.883	0.750	0.818	0.830
R106	0.814	0.711	0.755	0.830	0.761	0.778	0.786
R107	0.715	0.656	0.696	0.807	0.663	0.677	0.750
R108	0.563	0.498	0.615	0.758	0.610	0.613	0.648
R109	0.771	0.496	0.711	0.777	0.570	0.648	0.564
R110	0.664	0.463	0.582	0.723	0.571	0.582	0.556
R111	0.663	0.548	0.657	0.745	0.600	0.620	0.668
R112	0.602	0.438	0.648	0.722	0.571	0.507	0.526
R1	0.761	0.650	0.740	0.816	0.700	0.721	0.730
R201	0.757	0.720	0.706	0.773	0.721	0.742	0.748
R202	0.761	0.713	0.699	0.770	0.708	0.721	0.734
R203	0.716	0.640	0.674	0.728	0.656	0.671	0.690
R204	0.646	0.541	0.592	0.663	0.591	0.611	0.654
R205	0.726	0.650	0.637	0.712	0.609	0.648	0.665
R206	0.734	0.654	0.741	0.772	0.704	0.664	0.718
R207	0.605	0.466	0.608	0.618	0.541	0.540	0.581
R208	0.629	0.496	0.663	0.654	0.591	0.564	0.648
R209	0.695	0.600	0.664	0.683	0.615	0.648	0.699
R210	0.755	0.640	0.676	0.728	0.666	0.666	0.741
R211	0.642	0.506	0.631	0.677	0.548	0.548	0.583
R2	0.697	0.602	0.663	0.707	0.632	0.638	0.678

Table E.2: Average hypervolume obtained on Solomon's RC instances of size 100. Each variant is represented by a pair (number of groups, injection strategy).

Category	$(1, -)$	$(3, s_I^i)$	$(3, s_I^d)$	$(5, s_I^i)$	$(5, s_I^d)$	(M, s_I^i)	(M, s_I^d)
RC101	0.823	0.741	0.789	0.864	0.776	0.835	0.795
RC102	0.826	0.645	0.778	0.861	0.692	0.783	0.794
RC103	0.768	0.626	0.691	0.857	0.690	0.723	0.733
RC104	0.538	0.444	0.611	0.726	0.561	0.665	0.581
RC105	0.750	0.611	0.727	0.832	0.689	0.752	0.719
RC106	0.689	0.459	0.664	0.779	0.613	0.643	0.654
RC107	0.618	0.436	0.618	0.795	0.518	0.653	0.542
RC108	0.594	0.351	0.630	0.748	0.528	0.632	0.568
RC1	0.701	0.539	0.689	0.808	0.633	0.711	0.673
RC201	0.812	0.751	0.759	0.810	0.769	0.784	0.788
RC202	0.815	0.724	0.721	0.803	0.742	0.745	0.788
RC203	0.713	0.589	0.637	0.702	0.665	0.640	0.691
RC204	0.759	0.630	0.719	0.741	0.699	0.723	0.768
RC205	0.801	0.703	0.729	0.773	0.736	0.745	0.770
RC206	0.811	0.644	0.742	0.760	0.720	0.730	0.758
RC207	0.744	0.569	0.644	0.706	0.651	0.651	0.691
RC208	0.742	0.517	0.655	0.733	0.597	0.698	0.638
RC2	0.775	0.641	0.701	0.754	0.697	0.714	0.736

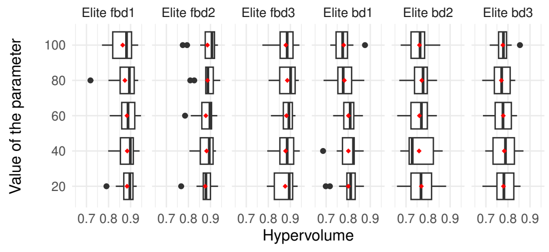
Table E.3: Average hypervolume obtained on Solomon's C instances of size 100. Each variant is represented by a pair (number of groups, injection strategy).

Category	$(1, -)$	$(3, s_I^i)$	$(3, s_I^d)$	$(5, s_I^i)$	$(5, s_I^d)$	(M, s_I^i)	(M, s_I^d)
C101	0.964	1.002	1.002	1.002	1.000	1.002	1.002
C102	0.955	0.881	0.880	0.985	0.874	0.938	0.934
C103	0.833	0.612	0.748	0.894	0.646	0.774	0.783
C104	0.788	0.601	0.794	0.857	0.736	0.805	0.749
C105	0.954	0.962	0.972	0.994	0.951	0.984	0.974
C106	0.911	0.906	0.937	1.002	0.844	0.982	0.951
C107	0.911	0.814	0.792	0.993	0.815	1.002	0.979
C108	0.926	0.851	0.851	0.987	0.808	0.948	0.923
C109	0.884	0.768	0.872	0.975	0.861	0.938	0.930
C1	0.903	0.822	0.872	0.965	0.837	0.930	0.914
C201	0.993	0.993	0.938	1.002	0.956	1.002	0.975
C202	0.913	0.844	0.868	0.885	0.774	0.894	0.927
C203	0.938	0.885	0.884	0.957	0.891	0.945	0.924
C204	0.807	0.634	0.666	0.842	0.680	0.707	0.783
C205	0.861	0.957	0.882	0.989	0.849	0.985	0.994
C206	0.857	0.925	0.796	0.985	0.824	0.999	0.929
C207	0.807	0.882	0.768	0.973	0.805	0.987	0.912
C208	0.802	0.896	0.773	0.987	0.807	0.994	0.961
C2	0.872	0.877	0.822	0.952	0.823	0.939	0.926

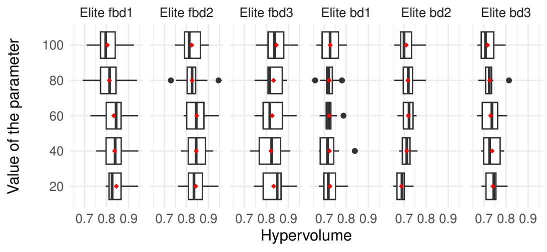
Appendix F

Parameters Analysis

This appendix is related to Section 5.3, where an analysis of the different parameters is performed. The parameters studied are the number of subproblems, the granularity, the crossover probability, the local search probability (see also Section 5.3.2), the exploration strategy, the granularity metric, the number of knowledge groups, the maximum size of the patterns extracted, the maximum number of patterns injected, and the injection probability. Six elite configurations returned by irace are used. There is one configuration per pair (exploration strategy, granularity metric). The analysis of these parameters led to a second tuning, used to generate the final tables. Summarized results are available in Section 5.3.3.3).

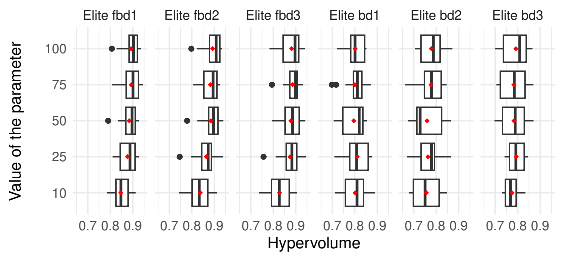


(a) Category C.

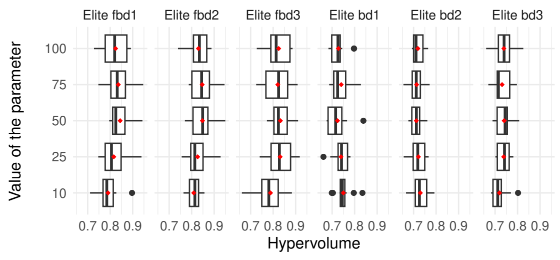


(b) Category R.

Figure F.1: Influence of the number of subproblems on instances of category C and R.

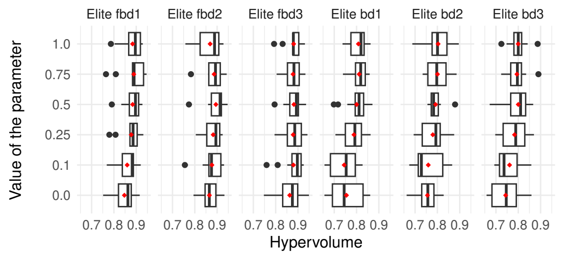


(a) Category C.

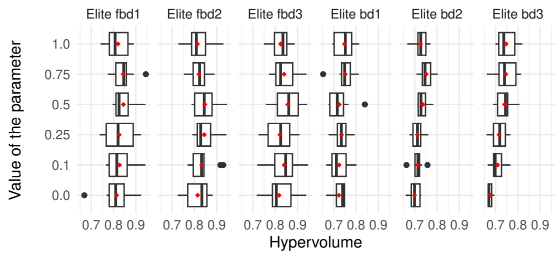


(b) Category R.

Figure F.2: Influence of the granularity on instances of category C and R.

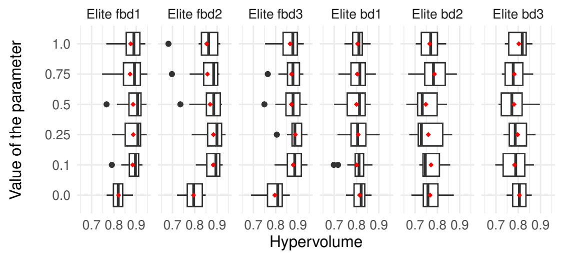


(a) Category C.

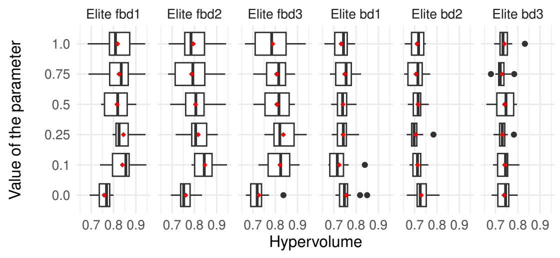


(b) Category R.

Figure F.3: Influence of the crossover probability on instances of category C and R.

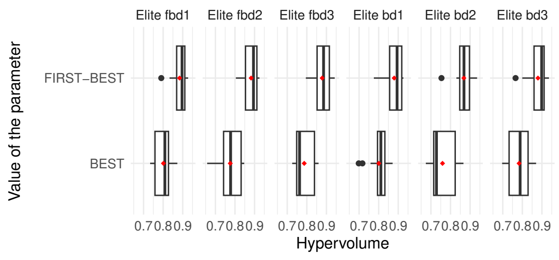


(a) Category C.

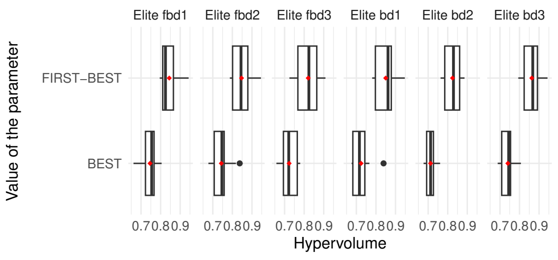


(b) Category R.

Figure F.4: Influence of the local search probability on instances of category C and R.

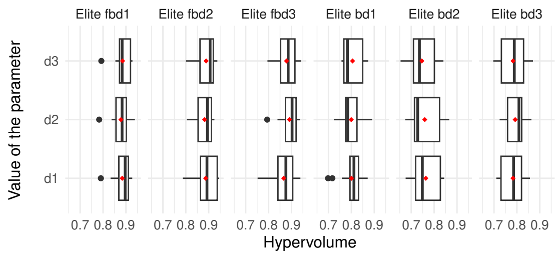


(a) Category C.

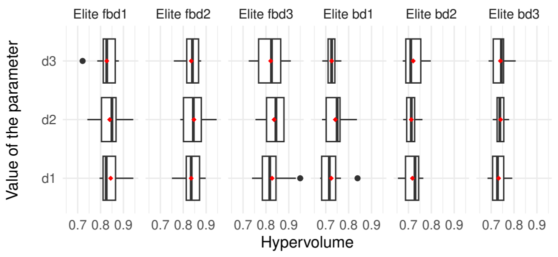


(b) Category R.

Figure F.5: Influence of the exploration strategy on instances of category C and R.

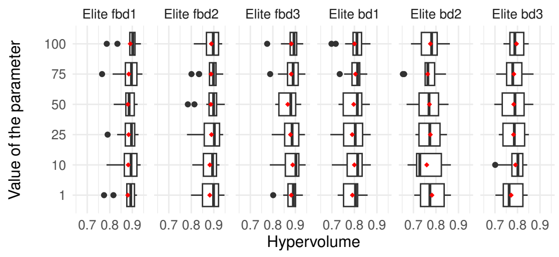


(a) Category C.

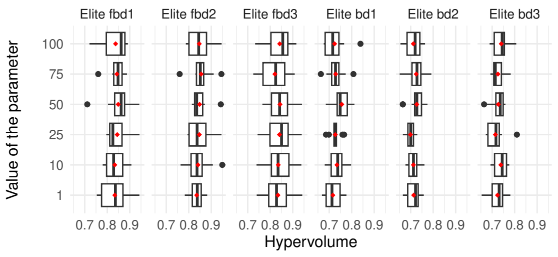


(b) Category R.

Figure F.6: Influence of the granularity metric on instances of category C and R.

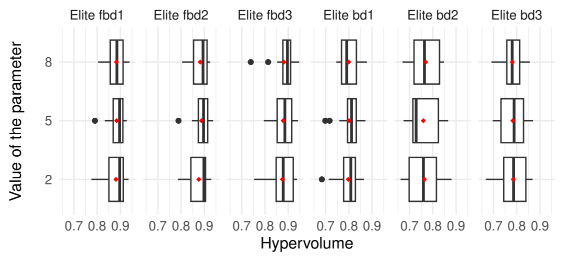


(a) Category C.

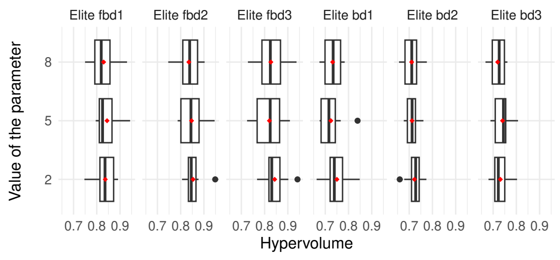


(b) Category R.

Figure F.7: Influence of the number of knowledge groups on instances of category C and R.

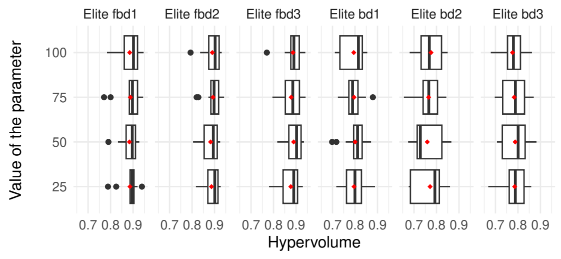


(a) Category C.

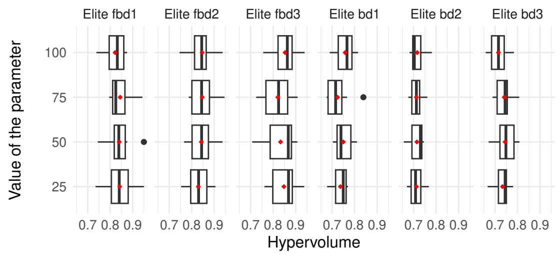


(b) Category R.

Figure F.8: Influence of the maximum size of extracted patterns on instances of category C and R.

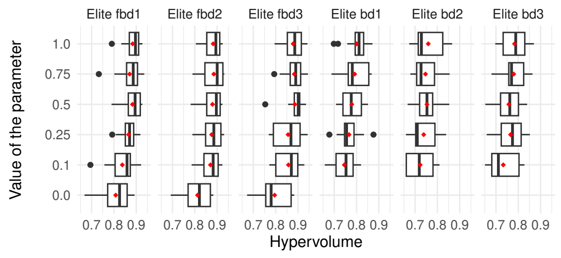


(a) Category C.

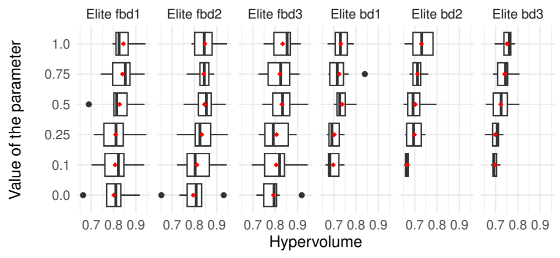


(b) Category R.

Figure F.9: Influence of the number of patterns injected on instances of category C and R.



(a) Category C.



(b) Category R.

Figure F.10: Influence of the injection probability on instances of category C and R.

Table F.1: Detailed results for algorithms fbd_2, bd_2, and $\mathrm{R}_{\mathrm{MOEA/D}}$ on Gehring and Homberger's C instances of size 200. From left to right: the average size of the front, the average uHV, and the average time to reach 80% of the reference uHV. A uHV of 0.000 means that the solutions found are too far from the nadir of the current reference front. A uHV strictly higher than 1.002 means that better non-dominated solutions were found (i.e., dominating the current ideal point).

	fbd_2			bd_2			$\mathrm{R}_{\mathrm{MOEA/D}}$		
Inst.	$\|F\|$	uHV	Time (s)	$\|F\|$	uHV	Time (s)	$\|F\|$	uHV	Time (s)
C1_2_1	2.0	0.990	815.3	2.1	0.875	2359.1	5.7	0.024	2882.3
C1_2_2	5.0	1.122	1235.0	6.1	0.298	2865.2	12.4	0.000	2883.2
C1_2_3	4.5	1.081	1569.7	5.2	0.108	2892.9	17.0	0.003	2883.1
C1_2_4	2.5	1.017	1849.2	2.1	0.037	2896.6	11.7	0.000	2882.4
C1_2_5	1.0	0.977	1047.0	2.3	0.680	2706.3	3.8	0.002	2882.2
C1_2_6	3.4	0.926	1605.4	4.2	0.450	2866.2	9.5	0.000	2882.5
C1_2_7	1.6	0.930	1610.4	2.3	0.393	2875.3	3.3	0.000	2882.1
C1_2_8	1.1	0.936	1333.2	1.6	0.111	2892.3	3.5	0.000	2882.1
C1_2_9	1.1	0.950	1469.7	1.4	0.062	2905.4	2.2	0.000	2881.4
C1_2_10	1.1	1.165	1118.6	1.0	0.064	2903.8	1.8	0.000	2881.7
C2_2_1	1.0	1.002	379.2	1.1	0.464	2474.6	3.2	0.625	2577.7
C2_2_2	7.0	0.929	984.0	3.7	0.165	2876.9	6.1	0.450	2882.9
C2_2_3	11.8	1.065	1167.2	4.0	0.090	2924.0	12.4	0.536	2883.0
C2_2_4	10.6	1.034	1473.4	3.5	0.064	2956.9	10.9	0.617	2804.4
C2_2_5	1.2	0.979	621.3	3.0	0.257	2822.8	6.2	0.421	2882.6
C2_2_6	3.1	0.971	869.9	2.8	0.117	2887.9	8.4	0.408	2837.0
C2_2_7	2.4	0.974	841.0	3.6	0.085	2915.6	9.7	0.379	2843.8
C2_2_8	1.2	0.955	802.4	2.8	0.087	2909.6	3.9	0.384	2881.8
C2_2_9	2.8	0.962	1160.0	2.9	0.089	2934.9	13.0	0.348	2882.1
C2_2_10	2.3	0.954	1006.9	1.8	0.017	2949.1	4.6	0.301	2882.1

Table F.2: Detailed results for algorithms fbd_2, bd_2, and $\mathrm{R}_{\mathrm{MOEA/D}}$ on Gehring and Homberger's R instances of size 200. From left to right: the average size of the front, the average uHV, and the average time to reach 80% of the reference uHV. A uHV strictly higher than 1.002 means that better non-dominated solutions were found (dominating the current ideal point).

	fbd_2			bd_2			$\mathrm{R}_{\mathrm{MOEA/D}}$		
Inst.	$\|F\|$	uHV	Time (s)	$\|F\|$	uHV	Time (s)	$\|F\|$	uHV	Time (s)
R1_2_1	88.3	0.883	833.1	67.5	0.855	2212.0	48.9	0.776	2886.3
R1_2_2	67.1	0.882	1269.8	43.7	0.820	2584.8	45.4	0.746	2888.6
R1_2_3	50.2	0.904	1380.1	29.7	0.781	2762.4	44.5	0.688	2888.7
R1_2_4	22.1	0.965	1552.7	13.0	0.731	2810.8	35.5	0.552	2887.5
R1_2_5	56.0	0.879	1354.0	42.5	0.845	2326.8	32.6	0.680	2888.1
R1_2_6	48.7	0.894	1423.0	32.1	0.808	2683.0	43.4	0.663	2887.0
R1_2_7	34.2	0.918	1594.1	19.4	0.802	2732.1	45.1	0.600	2885.5
R1_2_8	14.4	1.038	1643.4	8.3	0.695	2803.3	23.6	0.338	2884.8
R1_2_9	40.9	0.857	1916.2	30.9	0.828	2562.9	30.3	0.598	2888.0
R1_2_10	23.5	0.975	1438.9	14.1	0.801	2658.0	23.1	0.438	2887.0
R2_2_1	87.7	0.894	631.3	44.4	0.824	2414.6	44.4	0.864	897.6
R2_2_1	87.7	0.894	631.3	44.4	0.824	2414.6	44.4	0.864	897.6
R2_2_2	68.0	0.865	1230.0	24.5	0.727	2835.5	32.0	0.813	2256.7
R2_2_3	54.0	0.853	1595.2	18.0	0.629	2891.5	22.4	0.795	2714.0
R2_2_4	29.9	0.895	1766.0	7.4	0.453	2975.2	14.6	0.730	2870.8
R2_2_5	56.2	0.892	1154.7	24.1	0.747	2725.9	29.2	0.814	2261.6
R2_2_6	50.2	0.885	1284.2	14.1	0.612	2897.9	21.3	0.793	2654.1
R2_2_7	36.1	0.829	2258.9	12.2	0.524	2913.8	15.9	0.704	2889.3
R2_2_8	20.2	0.898	1974.6	6.7	0.456	2897.9	8.7	0.632	2877.7
R2_2_9	45.4	0.893	1160.1	21.9	0.714	2775.6	24.8	0.794	2553.1
R2_2_10	30.2	0.888	1693.8	12.1	0.594	2891.6	16.1	0.718	2836.8

Bibliography

Luca Accorsi and Daniele Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, 55(4):832–856, 2021.

Florian Arnold and Kenneth Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, 105:32–46, 2019.

Florian Arnold, Ítalo Santana, Kenneth Sörensen, and Thibaut Vidal. PILS: Exploring high-order neighborhoods by pattern mining and injection. *Pattern Recognition*, 2021.

Sunith Bandaru and Kalyanmoy Deb. A dimensionally-aware genetic programming architecture for automated innovization. In *Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013, Sheffield, UK, March 19-22, 2013. Proceedings 7*, pages 513–527. Springer, 2013.

Sunith Bandaru, Amos HC Ng, and Kalyanmoy Deb. Data mining methods for knowledge discovery in multi-objective optimization: Part a-survey. *Expert Systems with Applications*, 70:139–159, 2017.

RaúL Baños, Julio Ortega, ConsolacióN Gil, Antonio L MáRquez, and Francisco De Toro. A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. *Computers & Industrial Engineering*, 65(2):286–296, 2013.

Benjamín Barán and Matilde Schaerer. A multiobjective ant colony system for vehicle routing problem with time windows. In *Applied informatics*, pages 97–102, 2003.

Matthieu Basseur, Rong-Qiang Zeng, and Jin-Kao Hao. Hypervolume-based multi-objective local search. *Neural Computing and Applications*, 21:1917–1929, 2012.

Antonio Benitez-Hidalgo, Antonio J Nebro, Jose Garcia-Nieto, Izaskun Oregi, and Javier Del Ser. jMetalPy: A python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, 51:100598, 2019.

Nicola Beume, Carlos M Fonseca, Manuel Lopez-Ibanez, Luis Paquete, and Jan Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082, 2009.

Aymeric Blot, Laetitia Jourdan, and Marie-Éléonore Kessaci. Automatic design of multi-objective local search algorithms: case study on a bi-objective permutation flowshop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 227–234, 2017.

Aymeric Blot, Manuel López-Ibáñez, Marie-Éléonore Kessaci, and Laetitia Jourdan. Archive-aware scalarisation-based multi-objective local search for a bi-objective permutation flowshop problem. In *Parallel Problem Solving from Nature-PPSN XV*, 2018a.

Aymeric Blot, Marie-Eléonore Marmion, and Laetitia Jourdan. Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation. *J. Heuristics*, 24(6):853–877, 2018b. doi: 10.1007/s10732-018-9381-1. URL https://doi.org/10.1007/s10732-018-9381-1.

Jakob Bossek, Christian Grimme, Stephan Meisel, Günter Rudolph, and Heike Trautmann. Local search effects in bi-objective orienteering. In *Proceedings of the genetic and evolutionary computation conference*, pages 585–592, 2018.

Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

Jürgen Branke, Kalyanmoy Deb, Henning Dierolf, and Matthias Osswald. Finding knees in multi-objective optimization. In *International conference on parallel problem solving from nature*, pages 722–731. Springer, 2004.

Xinye Cai, Zhiwei Mei, Zhun Fan, and Qingfu Zhang. A constrained decomposition approach with grids for evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 22(4):564–577, 2017.

Juan Castro-Gutiérrez, Dario Landa-Silva, and José Moreno-Pérez. Dynamic lexicographic approach for heuristic multi-objective optimization. In *Proceedings of the Workshop on Intelligent Metaheuristics for Logistic Planning (CAEPIA-TTIA 2009)(Seville (Spain))*, pages 153–163, 2009.

Juan Castro-Gutierrez, Dario Landa-Silva, and José Moreno Pérez. Nature of real-world multi-objective vehicle routing with evolutionary algorithms. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 257–264. IEEE, 2011.

Vira Chankong and Yacov Y Haimes. *Multiobjective decision making: theory and methodology*. Courier Dover Publications, 2008.

Tsung-Che Chiang and Wei-Huai Hsu. A knowledge-based evolutionary algorithm for the multiobjective vehicle routing problem with time windows. *Computers & Operations Research*, 45:25–37, 2014.

Tsung-Che Chiang and Yung-Pin Lai. MOEA/D-AMS: Improving MOEA/D by an adaptive mating selection mechanism. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 1473–1480. IEEE, 2011.

Carlos A Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. Multi-objective combinatorial optimization: Problematic and context. In *Advances in multi-objective nature inspired computing*, pages 1–21. Springer, 2010.

David Corne, Clarisse Dhaenens, and Laetitia Jourdan. Synergies between operations research and data mining: The emerging use of multi-objective approaches. *European Journal of Operational Research*, 221(3):469–479, 2012.

Piotr Czyzżak and Adrezej Jaszkiewicz. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of multi-criteria decision analysis*, 7(1): 34–47, 1998.

Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

Kalyanmoy Deb and Aravind Srinivasan. Innovization: Innovating design principles through optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1629–1636, 2006.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. Multi-objective optimization. In *Decision sciences*, pages 161–200. CRC Press, 2016.

Jacques Desrosiers and Marco E Lübbecke. Branch-price-and-cut algorithms. *Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Chichester*, pages 109–131, 2011.

Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

Mădălina M Drugan and Dirk Thierens. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18:727–766, 2012.

Jonas K Falkner and Lars Schmidt-Thieme. Learning to solve vehicle routing problems with time windows through joint attention. *arXiv preprint arXiv:2006.09100*, 2020.

Abel Garcia-Najera and John A Bullinaria. An improved multi-objective evolutionary algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 38(1): 287–300, 2011.

Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

Hermann Gehring and Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64. Springer Berlin, 1999.

Martin Geiger. *A Genetic Algorithm applied to multiple objective vehicle routing problems.* Citeseer, 2001.

Martin Geiger. A computational study of genetic crossover operators for multi-objective vehicle routing problem with soft time windows. In *Multi-Criteria-und Fuzzy-Systeme in Theorie und Praxis*, pages 191–207. Springer, 2003.

Keivan Ghoseiri and Seyed Farid Ghannadpour. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing*, 10(4):1096–1107, 2010.

Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

David E Goldberg. Genetic algorithms in search. *Optimization, and MachineLearning*, 1989.

Arthur Guijt, Ngoc Hoang Luong, Peter AN Bosman, and Mathijs de Weerdt. On the impact of linkage learning, gene-pool optimal mixing, and non-redundant encoding on permutation optimization. *Swarm and Evolutionary Computation*, 70:101044, 2022.

Yi-Nan Guo, Jian Cheng, Sha Luo, Dunwei Gong, and Yu Xue. Robust dynamic multi-objective vehicle routing optimization method. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(6):1891–1903, 2017.

Michael Pilegaard Hansen et al. Tabu search for multiobjective optimization: MOTS. In *Proceedings of the 13th international conference on multiple criteria decision making*, pages 574–586. Citeseer, 1997.

Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A Moreno Pérez. *Variable neighborhood search*. Springer, 2019.

Kei Harada, Satoru Hiwa, and Tomoyuki Hiroyasu. Adaptive weight vector assignment method for MOEA/D. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–9. IEEE, 2017.

John H Holand. Adaptation in natural and artificial systems. *Ann Arbor: The University of Michigan Press*, page 32, 1975.

Sung-Chul Hong and Yang-Byung Park. A heuristic for bi-objective vehicle routing with time window constraints. *International Journal of Production Economics*, 62(3):249–258, 1999.

Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. Metaheuristic research: a comprehensive survey. *Artificial intelligence review*, 52:2191–2233, 2019.

Hisao Ishibuchi and Tadahiko Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 28(3):392–403, 1998.

Hisao Ishibuchi, Tsutomu Doi, and Yusuke Nojima. Effects of using two neighborhood structures in cellular genetic algorithms for function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 949–958. Springer, 2006.

Hisao Ishibuchi, Yuji Sakane, Noritaka Tsukamoto, and Yusuke Nojima. Adaptation of scalarizing functions in MOEA/D: An adaptive scalarizing function-based multiobjective evolutionary algorithm. In *Evolutionary Multi-Criterion Optimization: 5th International Conference, EMO 2009, Nantes, France, April 7-10, 2009. Proceedings 5*, pages 438–452. Springer, 2009.

Hisao Ishibuchi, Yuji Sakane, Noritaka Tsukamoto, and Yusuke Nojima. Simultaneous use of different scalarizing functions in MOEA/D. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 519–526, 2010.

Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified distance calculation in generational distance and inverted generational distance. In *Evolutionary Multi-Criterion Optimization: 8th International Conference, EMO 2015, Guimarães, Portugal, March 29–April 1, 2015. Proceedings, Part II 8*, pages 110–125. Springer, 2015.

Min Jiang, Zhongqiang Huang, Liming Qiu, Wenzhen Huang, and Gary G Yen. Transfer learning-based dynamic multiobjective optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 22(4):501–514, 2017a.

Shouyong Jiang, Shengxiang Yang, Yong Wang, and Xiaobin Liu. Scalarizing functions in decomposition-based multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 22(2):296–313, 2017b.

Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how? In *Proceedings of the genetic and evolutionary computation conference*, pages 1042–1049, 2001.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. Multi-objective vehicle routing problems. *European journal of operational research*, 189(2):293–309, 2008.

Liangjun Ke, Qingfu Zhang, and Roberto Battiti. MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and antcolony. *IEEE transactions on cybernetics*, 43(6):1845–1859, 2013.

James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 98–105. IEEE, 1999.

Joshua Knowles and David Corne. Properties of an adaptive archiving algorithm for storing non-dominated vectors. *IEEE Transactions on Evolutionary Computation*, 7(2):100–116, 2003.

Joshua D Knowles. *Local-search and hybrid evolutionary algorithms for Pareto optimization*. PhD thesis, University of Reading, Reading, 2002.

Joshua D Knowles, David W Corne, and Mark Fleischer. Bounded archiving using the Lebesgue measure. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 4, pages 2490–2497. IEEE, 2003.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

Mark William Shannon Land. *Evolutionary algorithms with local search for combinatorial optimization*. University of California, San Diego, 1998.

Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282, 2002.

Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, 2006.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, and Marie-Eléonore Kessaci. Enhancing MOEA/D with learning: application to routing problems with time windows. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 495–498, 2022a.

Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, and Marie-Eléonore Kessaci. New neighborhood strategies for the bi-objective vehicle routing problem with time windows. In *Metaheuristics International Conference*, pages 45–60. Springer, 2022b.

Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, and Marie-Eléonore Kessaci. Improving neighborhood exploration into MOEA/D framework to solve a bi-objective routing problem. *International Transactions in Operational Research*, 2023a.

Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, and Marie-Eléonore Kessaci. Improving MOEA/D with knowledge discovery. application to a bi-objective routing problem. In *EMO: 12th International Conference, 2023, Proceedings*, pages 462–475. Springer, 2023b.

Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, and Marie-Eléonore Kessaci. Investigation of the benefit of extracting patterns from local optima to solve a bi-objective VRPTW. In *Metaheuristics International Conference*, pages 62–77. Springer, 2024.

Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. *IEEE transactions on evolutionary computation*, 13(2):284–302, 2008.

Hui Li, Min Ding, Jingda Deng, and Qingfu Zhang. On the use of random weights in MOEA/D. In *2015 IEEE congress on evolutionary computation (CEC)*, pages 978–985. IEEE, 2015.

Ke Li, Alvaro Fialho, Sam Kwong, and Qingfu Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130, 2013a.

Ke Li, Qingfu Zhang, Sam Kwong, Miqing Li, and Ran Wang. Stable matching-based selection in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 18 (6):909–923, 2013b.

Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, and El-Ghazali Talbi. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 18:317–352, 2012.

Hai-Lin Liu, Fangqing Gu, and Qingfu Zhang. Decomposition of a multiobjective optimization problem into a number of simple multiobjective subproblems. *IEEE transactions on evolutionary computation*, 18(3):450–455, 2013.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

Jose A Lozano. *Towards a new evolutionary computation: advances on estimation of distribution algorithms*, volume 192. Springer Science & Business Media, 2006.

Flavien Lucas, Romain Billot, Marc Sevaux, and Kenneth Sörensen. Reducing space search in combinatorial optimization using machine learning tools. In *International Conference on Learning and Intelligent Optimization*, pages 143–150. Springer, 2020.

Xiaoliang Ma, Fang Liu, Yutao Qi, Maoguo Gong, Minglei Yin, Lingling Li, Licheng Jiao, and Jianshe Wu. MOEA/D with opposition-based learning for multiobjective optimization problem. *Neurocomputing*, 146:48–64, 2014a.

Xiaoliang Ma, Fang Liu, Yutao Qi, Lingling Li, Licheng Jiao, Meiyun Liu, and Jianshe Wu. MOEA/D with Baldwinian learning inspired by the regularity property of continuous multi-objective problem. *Neurocomputing*, 145:336–352, 2014b.

Xiaoliang Ma, Fang Liu, Yutao Qi, Lingling Li, Licheng Jiao, Xiaozheng Deng, Xiaodong Wang, Bei Dong, Zhanting Hou, Yongxiao Zhang, et al. MOEA/D with biased weight adjustment inspired by user preference and its application on multi-objective reservoir flood control problem. *Soft Computing*, 20:4999–5023, 2016.

Xiaoliang Ma, Qingfu Zhang, Guangdong Tian, Junshan Yang, and Zexuan Zhu. On Tchebycheff decomposition approaches for multiobjective evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 22(2):226–244, 2017.

Kent McClymont and Edward C Keedwell. Markov chain hyper-heuristic (MCHH) an online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 2003–2010, 2011.

Belén Melián-Batista, Alondra De Santiago, Francisco AngelBello, and Ada Alvarez. A bi-objective vehicle routing problem with time windows: A real case in Tenerife. *Applied Soft Computing*, 17: 140–152, 2014.

Adriana Menchaca-Mendez and Carlos A Coello Coello. GD-MOEA: A new multi-objective evolutionary algorithm based on the generational distance indicator. In *International conference on evolutionary multi-criterion optimization*, pages 156–170. Springer, 2015.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

Behzad Moradi. The new optimization algorithm for the vehicle routing problem with time windows using multi-objective discrete learnable evolution model. *Soft Computing*, 24(9):6741–6769, 2020.

Yuichi Nagata, Olli Bräysy, and Wout Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & operations research*, 37(4): 724–737, 2010a.

Yuichi Nagata, Olli Bräysy, and Wout Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4): 724–737, 2010b. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2009.06.022. URL https://www.sciencedirect.com/science/article/pii/S0305054809001762.

Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, 2006.

Marek Ostaszewski, Pascal Bouvry, and Franciszek Seredynski. Multiobjective classification with moGEP: an application in the network traffic domain. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 635–642, 2009.

Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

Luis Paquete and Thomas Stützle. A two-phase local search for the biobjective traveling salesman problem. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 479–493. Springer, 2003.

Luis Paquete, Marco Chiarandini, and Thomas Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, pages 177–199, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-642-17144-4.

Martin Pilat and Roman Neruda. Incorporating user preferences in MOEA/D through the coevolution of weights. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 727–734, 2015.

Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & operations research*, 31(12):1985–2002, 2004.

Yutao Qi, Xiaoliang Ma, Fang Liu, Licheng Jiao, Jianyong Sun, and Jianshe Wu. MOEA/D with adaptive weight adjustment. *Evolutionary computation*, 22(2):231–264, 2014.

Yutao Qi, Zhanting Hou, He Li, Jianbin Huang, and Xiaodong Li. A decomposition based memetic algorithm for multi-objective vehicle routing problem with time windows. *Computers & Operations Research*, 62:61–77, 2015.

Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2015.

Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European journal of operational research*, 177(3):2033–2049, 2007.

J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

Michael Schneider, Fabian Schwahn, and Daniele Vigo. Designing granular solution methods for routing problems with time windows. *European Journal of Operational Research*, 263(2):493–509, 2017.

Paolo Serafini. Simulated annealing for multi objective optimization problems. In *Multiple Criteria Decision Making: Proceedings of the Tenth International Conference: Expand and Enrich the Domains of Thinking and Application*, pages 283–292. Springer, 1994.

Jiang Siwei, Cai Zhihua, Zhang Jie, and Ong Yew-Soon. Multiobjective optimization by decomposition with pareto-adaptive weight vectors. In *2011 Seventh international conference on natural computation*, volume 3, pages 1260–1264. IEEE, 2011.

Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531, 2013.

Éric D Taillard. *Design of heuristic algorithms for hard optimization: with python codes for the travelling salesman problem*. Springer Nature, 2023.

El-Ghazali Talbi. Machine learning into metaheuristics: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(6):1–32, 2021.

Kay Chen Tan, YH Chew, and Loo Hay Lee. A hybrid multi-objective evolutionary algorithm for solving truck and trailer vehicle routing problems. *European Journal of Operational Research*, 172(3):855–885, 2006.

Yan-Yan Tan, Yong-Chang Jiao, and Xin-Kuan Wang. MOEA/D with uniform design for the multiobjective 0/1 knapsack problem. In *2011 Seventh International Conference on Computational Intelligence and Security*, pages 96–100. IEEE, 2011.

Sara Tari, Matthieu Basseur, and Adrien Goëffon. Partial neighborhood local searches. *International Transactions in Operational Research*, 29(5):2761–2788, 2022.

Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4):333–346, 2003.

Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications.* SIAM, 2014.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

Tamara Ulrich, Johannes Bader, and Eckart Zitzler. Integrating decision space diversity into hypervolume-based multiobjective search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 455–462, 2010.

Ekunda L Ulungu, Jacques Teghem, and Philippe Fortemps. Heuristics for multi-objective combinatorial optimization by simulated annealing. In *Multiple Criteria Decision Making: Theory and Applications. Proceedings of the 6th National Conference on Multiple Criteria Decision Making*, pages 228–238, 1995.

Thibaut Vidal. Split algorithm in O(n) for the capacitated vehicle routing problem. *Computers & Operations Research*, 69:40–47, 2016.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489, 2013.

Luping Wang, Qingfu Zhang, Aimin Zhou, Maoguo Gong, and Licheng Jiao. Constrained subproblems in a decomposition-based multiobjective evolutionary algorithm. *IEEE Transactions on Evolutionary computation*, 20(3):475–480, 2015.

Rui Wang, Qingfu Zhang, and Tao Zhang. Decomposition-based algorithms using pareto adaptive scalarizing methods. *IEEE Transactions on Evolutionary Computation*, 20(6):821–837, 2016.

Zhenkun Wang, Qingfu Zhang, Maoguo Gong, and Aimin Zhou. A replacement strategy for balancing convergence and diversity in MOEA/D. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 2132–2139. IEEE, 2014.

Mengyuan Wu, Ke Li, Sam Kwong, Qingfu Zhang, and Jun Zhang. Learning to decompose: A paradigm for decomposition-based multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 23(3):376–390, 2018.

Zhihao Xing and Shikui Tu. A graph neural network assisted Monte Carlo tree search approach to traveling salesman problem. *IEEE Access*, 8:108418–108428, 2020.

Hang Xu, Wenhua Zeng, Defu Zhang, and Xiangxiang Zeng. MOEA/HD: A multiobjective evolutionary algorithm based on hierarchical decomposition. *IEEE Transactions on cybernetics*, 49 (2):517–526, 2017.

Qian Xu, Zhanqi Xu, and Tao Ma. A survey of multiobjective evolutionary algorithms based on decomposition: variants, challenges and future directions. *IEEE Access*, 8:41588–41614, 2020.

Xin-She Yang. *Nature-inspired optimization algorithms.* Academic Press, 2020.

William B Yates and Edward C Keedwell. An analysis of heuristic subsequences for offline hyper-heuristic learning. *Journal of Heuristics*, 25(3):399–430, 2019.

Stanisław K Zaremba. Good lattice points, discrepancy, and numerical integration. *Annali di matematica pura ed applicata*, 73:293–317, 1966.

Kaikai Zhang, Yiqiao Cai, Shunkai Fu, and Huizhen Zhang. Multiobjective memetic algorithm based on adaptive local search chains for vehicle routing problem with time windows. *Evolutionary Intelligence*, pages 1–12, 2019.

Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

Ying Zhang, Rennong Yang, Jialiang Zuo, and Xiaoning Jing. Enhancing MOEA/D with uniform population initialization, weight vector design and adjustment using uniform design. *Journal of Systems Engineering and Electronics*, 26(5):1010–1022, 2015.

Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes. *IEEE Transactions on Evolutionary Computation*, 16(3):442–446, 2012.

Wei Zheng, Yanyan Tan, Lili Meng, and Huaxiang Zhang. An improved MOEA/D design for many-objective optimization problems. *Applied Intelligence*, 48:3839–3861, 2018.

Wei Zhou, Tingxin Song, Fei He, and Xi Liu. Multiobjective vehicle routing problem with route balance based on genetic algorithm. *discrete Dynamics in Nature and Society*, 2013, 2013.

Ying Zhou and Jiahai Wang. A local search-based multiobjective optimization algorithm for multiobjective vehicle routing problem with time windows. *IEEE Systems Journal*, 9(3):1100–1113, 2014.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003.

Konstantinos G Zografos and Konstantinos N Androutsopoulos. A heuristic algorithm for solving hazardous materials distribution problems. *European Journal of Operational Research*, 152(2): 507–519, 2004.

Titre : Extraction de connaissance pour les métaheuristiques à partir des solutions appliquée aux problèmes de tournées de véhicules bi-objectif

Mots-Clés : optimisation combinatoire, optimisation multi-objectif, metaheuristiques, apprentissage machine, extraction de connaissance, tournées de véhicule.

Résumé : Cette thèse intitulée « Extraction de connaissance pour les métaheuristiques à partir des solutions appliquée aux problèmes de tournées de véhicules bi-objectif » s'intéresse à l'intégration de mécanismes d'apprentissage au sein d'algorithmes multi-objectifs existants. En effet, l'utilisation d'apprentissage machine pour résoudre des problèmes d'optimisation combinatoire a permis d'améliorer de manière significative (à la fois en termes de performance et de temps d'exécution) des métaheuristiques existantes. Nous nous sommes concentrés sur un problème de tournées de véhicules bi-objectif avec fenêtres de temps qui est un problème de logistique où l'on cherche à optimiser la création de tournées pour livrer chaque client à une période précise, symbolisée par une fenêtre de temps. La résolution de ce type de problèmes est un enjeux pour de nombreuses entreprises. Les deux objectifs minimisés sont le coût total de transport et le temps total d'attente des livreurs, provoqué par l'arrivée du livreur avant le début de la période de livraison. Pour résoudre ce problème, nous proposons d'exploiter les séquences de clients livrés consécutivement au sein d'une tournée. Ces séquences sont extraites lors de l'exécution de l'algorithme à partir de solutions générées. Les séquences les plus prometteuses sont ensuite intégrées dans d'autres solutions pour les améliorer. Si l'apprentissage de séquences pour résoudre ce type de problèmes s'est révélé efficace en mono-objectif, cela reste un challenge de les exploiter dans un cadre multi-objectif, puisque certaines séquences intéressantes pour un objectif peuvent se révéler inutiles pour l'autre objectif. Plus précisément, dans cette thèse, nous nous interrogeons sur les manières d'exploiter au mieux les séquences disponibles dans les solutions. En particulier cela nous a conduits à nous poser les questions suivantes : comment gérer ces séquences dans un cadre multi-objectif ? De quelles solutions devons-nous extraire les séquences et dans quelles solutions les injecter ? A quel moment de l'exécution, les étapes d'injection et d'extraction doivent-elles être effectuées ? Les réponses à ces questions nous ont menés à l'élaboration d'un modèle d'apprentissage exploitant les séquences des solutions dans un cadre multi-objectif, où des groupes de connaissance sont créés pour stocker les séquences relatives à une partie de l'espace de recherche. Ce modèle a ensuite été intégré dans deux algorithmes populaires : MOEA/D et MOLS, montrant l'efficacité du modèle proposé.

Title. Solution-based Knowledge Discovery in Metaheuristics for Bi-Objective Vehicle Routing Problems

Keywords. combinatorial optimization, multi-objective optimization, metaheuristics, machine learning, knowledge discovery, vehicle routing

Abstract This thesis entitled "Solution-based Knowledge Discovery in Metaheuristics for Bi-Objective Vehicle Routing Problems" focuses on the integration of learning mechanisms within existing multi-objective algorithms. Indeed, the use of machine learning to solve combinatorial optimization problems has led to significant improvements (both in terms of performance and execution time) in existing metaheuristics. We have focused on a bi-objective vehicle routing problem with time windows, which is a logistics problem where the aim is to optimize the creation of routes to deliver to each customer at a precise period, symbolized by a time window. Solving this type of problem is a challenge for many companies. The two objectives to be minimized are the total cost of transport and the total waiting time for the delivery driver, caused by the driver arriving before the start of the delivery period. To solve this problem, we propose to exploit the sequences of customers delivered consecutively within a tour. These sequences are extracted from generated solutions during algorithm execution. The most promising sequences are then integrated into other solutions to improve them. While learning sequences to solve this type of problem has proved effective in single-objective settings, it remains a challenge to exploit them in a multi-objective context. Indeed, some sequences that are interesting for one objective may prove useless for the other. More specifically, in this thesis, we are looking at ways of making the most of the sequences available in solutions. In particular, this led us to ask the following questions: how can we manage these sequences in a multi-objective framework? From which solutions should we extract sequences, and into which solutions should we inject them? At what point in the runtime should the injection and extraction steps be carried out? The answers to these questions led us to develop a learning model exploiting solution sequences in a multi-objective context, where knowledge groups are created to store sequences relating to a part of the search space. This model was then integrated into two popular algorithms: MOEA/D and MOLS, demonstrating the effectiveness of the proposed model.