Improving MOEA/D with Knowledge Discovery. Application to a Bi-objective Routing Problem

Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, Marie-Eléonore Kessaci

Improving MOEA/D with Knowledge Discovery. Application to a Bi-Objective Routing Problem

Clément Legrand[0000−0002−4367−4676]1, Diego Cattaruzza[0000−0002−1814−2547]2, Laetitia Jourdan[0000−0002−4170−6830]1, and Marie-Eléonore Kessaci[0000−0002−4372−5162]1

[1] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
clement.legrand4.etu, laetitia.jourdan, marie-eleonore.kessaci@univ-lille.fr
[2] Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
diego.cattaruzza@centralelille.fr

Abstract. Knowledge Discovery (KD) mechanisms (e.g. data mining, neural networks) receive more and more interest over the years. A KD mechanism uses an extraction procedure, namely K_{ext}, to discover knowledge, and an injection procedure, namely K_{inj}, to exploit knowledge. However such mechanisms are not often applied to multi-objective combinatorial problems, due to the optimization of many objectives, which can lead to learning conflicting knowledge. The key is to know how the components of the KD mechanism should coexist and interact with the knowledge. In this article, we work with the MOEA/D algorithm, and existing K_{inj} and K_{ext} components. We propose different interactions between the components of the KD mechanism, by using different numbers of knowledge groups (dedicated to the storage of the knowledge) and different strategies for the injection component. The variants are evaluated through the bi-objective Vehicle Routing Problem with Time Windows (bVRPTW). Our results show, that using five knowledge groups and an intensification strategy for the injection procedure leads to better results.

Keywords: Multi-Objective Optimisation · MOEA/D · Knowledge Discovery · Routing Problem

1 Introduction

When solving a discrete optimization problem, large parts of the search space are explored during the execution of the algorithm. However, most of the solutions encountered are simply ignored, while they can bring interesting knowledge about the search space. Indeed, this knowledge can guide the algorithm towards more interesting solutions [1]. On the other hand, it can also help the algorithm to avoid getting stuck in local optima, as explicitly defined in Tabu Search methods [10].

In multi-objective problems, at least two conflicting functions are simultaneously optimized and the objective is to find the Pareto front of solutions. Over

the years, many metaheuristics based on local search techniques and using evolutionary algorithms [6] have been designed to tackle these problems. Among the most popular algorithms, there are MOEA/D [28], NSGA-II [8], and their variants.

Using knowledge from explored solutions is helpful to reduce the search space or to focus on interesting parts of the space, and can improve the performances of the algorithms mentioned above. However extracting knowledge from solutions and then using it to guide the search is a complex task, which has not been highly explored in the literature. Considering the papers on that subject leads to the following terminology for *Knowledge Discovery* (KD) processes. A KD process is built upon two main procedures called *Knowledge Extraction* (K_{ext}) and *Knowledge Injection* (K_{inj}). The K_{ext} procedure aims to extract problem-related knowledge from one or several solutions. Then the extracted knowledge can be used by the K_{inj} procedure to build new solutions taking into account past iterations. However, given extraction and injection procedures for a specific problem, there exist a plethora of ways to integrate them within a metaheuristic.

In this article, we investigate how a KD mechanism can be integrated into MOEA/D. To that purpose, we consider a bi-objective Vehicle Routing Problem with Time Windows (bVRPTW). In this problem, we minimize both the total traveling time and the total waiting time of drivers. With these two objectives, we obtain more diverse and bigger fronts (in terms of cardinality) than those obtained when minimizing the number of vehicles and the total traveling time, which are the original optimized objectives. Moreover, considering the waiting time can lead to different applications (e.g. food delivery, medical transportation). We propose a large number of hybridization variants that are evaluated, showing that one, in particular, is statistically better than the others.

The remaining of the paper is organized as follows: Section 2 focuses on KD mechanisms and their link with combinatorial optimization. We present new strategies for the components of a KD process in Section 3. The KD mechanism is integrated into MOEA/D in Section 4. The bVRPTW is described in Section 5. Our experimental setup is presented in Section 6, and our protocol in Section 7. We show and discuss our results in Section 8. Finally, we conclude in Section 9.

2 Scientific Context

2.1 Knowledge Discovery in Metaheuristics

Hybridizing machine learning methods and metaheuristics has become quite common to solve combinatorial problems. The survey of Talbi [21] reviews a large panel of hybridizations that are frequently used in the literature. These hybridizations are divided into three categories depending on where the integration is performed: at a problem level, at a low level, or at a high level. A problem-level integration takes into account the characteristics of the problem itself (e.g. data relative to the instance considered) to guide the algorithm. A low-level integration focuses on solutions produced by algorithms. A relevant

mechanism is able to analyze the structure of the solutions, learn from them, and then use this knowledge to improve the next steps. A high-level integration is interesting when several operators are available to solve a problem. A possible interest is to design automatically a problem-specific heuristic by selecting the most relevant operators to apply. In the following, we focus on low-level integration and learning from solutions, also called knowledge discovery (KD). KD can be realized either *online* or *offline* [7]. It is called online when it uses resources generated during the execution. Otherwise, it is called offline. Both of them have pros and cons. Online KD are often more adaptive and based on unsupervised methods, which may lead to a slow convergence rate. While offline methods are often supervised, and thus require huge amounts of data to be efficient.

Most KD processes are composed of an *extraction* mechanism (K_{ext}), where something is learned, and an *injection* mechanism (K_{inj}), which uses the extracted knowledge to find new promising solutions. A study of existing works in KD and its hybridization with metaheuristics [21] leads to four main questions: *What*/ *Where*/ *When*/ *How* is the knowledge extracted/injected?

The question *What* is problem-dependent, since each problem may have specific relevant knowledge. In the context of this article, this question is answered in Section 5, where the problem is presented. Questions *Where* and *When* are algorithm-dependent since the extraction and injection steps have to be integrated into the process of the algorithm. Both of these questions are not the subject of this article, and thus not discussed here at length. However, these questions are answered in Section 4 for the specific case of our study. The question *How* deals with overall strategies used during the KD mechanism (e.g. intensification or diversification). The answer to this question should be adapted according to the category of the problem studied (multi-objective in our case). Our contribution focuses on this question and is detailed in Sections 3 and 4.

2.2 Knowledge Integration in Multi-Objective Optimization

In the literature, KD processes have received various interests mainly in single-objective optimization contexts. Especially in routing problems [2,3,15,1]. However, using KD processes in multi-objective combinatorial optimization is quite new and has not been widely investigated. Among the first works on this subject, we cite the paper of Wattanapornprom et al. [26]. In order to solve a bi-objective TSP they learn probabilities of arcs belonging to good solutions by using a reward and punishment system based on the solutions visited during the execution. The authors show that their learning procedure improves the performance of NSGA-II. The survey of Bandaru [4] regroups different data mining methods that can be used in multi-objective optimization. Recently, Moradi et al. [16] and Legrand et al. [12] proposed algorithms enhanced with learning mechanisms to solve routing problems. The former presented the MODLEM algorithm which uses decision trees updated during the execution to guide the algorithm through the search space. The latter designed a MOEA/D using a KD mechanism, that extracts sequences of customers from generated solutions and injects the most frequent ones in solutions to improve them.

3 Knowledge Discovery for Multi-Objective Optimization

In this section, we propose an answer to the question *How* presented in Section 2.1. This question focuses on the interaction of the extraction and injection components with the knowledge itself. First, we present how knowledge groups are defined in Section 3.1. These groups allow the storage and the use of knowledge by the extraction and injection mechanisms. Section 3.2 is more focused on the possible strategies followed by both extraction and injection when interacting with the knowledge groups.

3.1 Definition of Knowledge Groups

One issue of KD mechanisms concerns the structure used to store the extracted knowledge to be injected. In multi-objective optimization, the fitness space is not in 1-Dimension and generally, the best solutions for one objective are not the same as for the other ones. We make the assumption that solutions sharing some similarities are more likely to be in the same region of the fitness space.

We propose to divide the fitness space into $k_{\mathcal{G}}$ regions each representing a *knowledge group*. The set of knowledge groups is denoted as $\mathcal{G}$. Therefore, a knowledge group is defined by a delimited region of the fitness space. The region can be either explicit (represented by equations) or implicit (represented by sets). If a solution belongs to the region of a knowledge group, then its associated knowledge is added to that group.

The number of knowledge groups and their construction within MOEA/D is discussed in Section 4.4.

3.2 Intensification and Diversification 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 KD for the extraction and injection mechanisms. On the one hand, we propose an intensification strategy, where the procedure has access to a small number of groups. The objective of the procedure is to focus on the same region of the fitness space, by exploring close regions. 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 objective of the procedure is to explore different regions of the fitness space, by bringing diversity to the solutions. In that case, the knowledge can travel through the groups. The definition of these strategies for the integration of the KD into MOEA/D is discussed in Section 4.4.

4 MOEA/D Enhanced with Knowledge Discovery

4.1 MOEA/D

MOEA/D [28] is a genetic algorithm that approximates the Pareto front by decomposing the multi-objective problem into M several scalar objective sub-

problems. The scalarization is obtained by weighting each of the n objectives f_k with a weight $w_k \in [0,1]$. Thus the fitness of a solution x for the subproblem i is the following quantity: $f(x|w^i) = \sum_{k=1}^{n} w_k^i \cdot f_k(x)$.

During an iteration, MOEA/D minimizes the i-th subproblem by using the solutions of its closest neighbors. The *neighborhood*, of size m, of a weight vector w^i is defined as the set of its m closest (for the euclidean distance) weight vectors among $\{w^1, \ldots, w^M\}$. Then the neighborhood $\mathcal{N}_m(i)$ of the i-th subproblem consists of the m subproblems defined with a weight vector belonging to the neighborhood of w^i. Note that each subproblem is associated with its best solution found during the execution.

At the start of MOEA/D, M weight vectors are given, then it works as follows. Initially, a random population (of size M) is generated and evaluated. The neighborhood (of size m) of each subproblem is also computed. When optimizing subproblem i, a random pair of solutions is selected from its neighborhood. The Partially Mapped crossover (PMX) is applied with probability p_{cro}, and only one solution is randomly kept. Then a Local Search (LS), described in Section 5, is applied with probability p_{mut}. Indeed the mutation is frequently replaced by an LS [11] in genetic algorithms. Finally, the resulting solution is added to the set S of solutions generated during the iteration, and a few neighbors of the subproblem i are updated. When all subproblems have been seen, S is merged with the archive A. If the termination criterion is reached, the nondominated solutions of A are returned, otherwise, a new iteration is started.

4.2 Construction of the Knowledge Groups and Strategies

As explained in Section 3.1, we use knowledge groups to store the extracted knowledge. Since we work with MOEA/D, we use the underlying subproblems to delimit the $k_{\mathcal{G}}$ groups. Note that regions are defined implicitly. We propose to characterize each knowledge group $\mathcal{G}_k \in \mathcal{G}$ by a vector $g^k = (g_1^k, \ldots, g_n^k) \in [0,1]^n$ satisfying $g_1^k + \ldots + g_n^k = 1$. Since the weight vectors of subproblems are chosen uniformly in that hyperplane, we also choose $k_{\mathcal{G}}$ uniformly distributed vectors in the same hyperplane, so that groups are balanced. In the following, we assume that we work in a bi-dimensional case. If $k_{\mathcal{G}} = 1$, then the group is associated with all the subproblems, thus the vector characterizing the group does not matter, and we set $g^1 = (0.5, 0.5)$. In the general case, when $k_{\mathcal{G}} \geq 2$, for $k \in \{1, \ldots, k_{\mathcal{G}}\}$, we characterize $\mathcal{G}_k$ with $g^k = (\frac{k-1}{k_{\mathcal{G}}-1}, 1 - \frac{k-1}{k_{\mathcal{G}}-1})$.

The definition of the regions of the groups is linked to the strategy followed by the extraction. We consider the M subproblems and their associated weight vectors defined in MOEA/D. Given a subproblem i of weight vector w^i, we can compute the set $\mathcal{N}_{\mathcal{G}}(i) = \{d(w^i, g^k) | 1 \leq k \leq k_{\mathcal{G}}\}$, where $d(w^i, g^k)$ represents the Euclidean distance between the i-th subproblem and the group $\mathcal{G}_k$. With this set, we can know how far each group is from the i-th subproblem. We propose to associate each subproblem with its $m_{\mathcal{G}}^{ext}$ closest groups. Therefore, the region of a group is the set of subproblems that are associated with that group. The smaller the value of $m_{\mathcal{G}}^{ext}$, the more intensive the extraction. We decide to keep only the most intensive strategy ($m_{\mathcal{G}}^{ext} = 1$) for the extraction. More precisely, if

x is a solution obtained while optimizing the subproblem i, then only the closest group to i (regarding $\mathcal{N}_{\mathcal{G}}(i)$) receives the knowledge extracted from x.

Concerning the injection, we introduce similarly a parameter $m_{\mathcal{G}}^{inj}$. It represents the number of groups that can provide the knowledge to be injected. The diversity increases along with the value of $m_{\mathcal{G}}^{inj}$. For the study, we keep only the two extreme values being 1 (for intensification) and M (for diversification). More precisely, when $m_{\mathcal{G}}^{inj} = 1$, only the closest group to the subproblem can provide the knowledge, and when $m_{\mathcal{G}}^{inj} = M$, it can be any group (chosen at random).

4.3 MOEA/D with Knowledge Discovery

In this section, we combine the elements described in the former section to obtain the framework shown in Algorithm 1. If lines 3, 11, and 16 are removed, then the algorithm becomes the variant of MOEA/D described in Section 4.1. At line 3, the procedure `createGroups` is called to create the vector of each group, as explained in Section 4.2. At line 11, the injection procedure $\mathrm{K}_{\mathrm{inj}}$ is applied to the current solution x, using either an intensification ($m_{\mathcal{G}}^{inj} = 1$) or a diversification ($m_{\mathcal{G}}^{inj} = k_{\mathcal{G}}$) strategy as explained in Section 4.2. At line 16, the extraction procedure $\mathrm{K}_{\mathrm{ext}}$ is used to extract the knowledge from the set of solutions generated during the iteration. Then it updates the closest group ($m_{\mathcal{G}}^{ext} = 1$) of the subproblem being optimized as explained in Section 4.2. In the following section, we instantiate the Algorithm 1 with different values of $k_{\mathcal{G}}$.

4.4 Experimental Variants

In this section, we present and discuss the different values of $k_{\mathcal{G}}$ retained for the study. Since the extraction is performed in an intensive manner, only the strategies for the injection are considered.

First of all, we consider the simplest case, where there is only one group ($k_{\mathcal{G}} = 1$). In that case, the intensification is equivalent to the diversification, leading to only one variant, the so-called *Base* algorithm.

It is known that solutions in the middle of the front (i.e. solutions that have an equivalent trade-off between the objectives) are the most difficult to obtain. Therefore we need to create at least $k_{\mathcal{G}} = 3$ to obtain a relevant decomposition. In this article, we limit the investigation to the case where the groups are uniformly spread along the front. Thus, two groups are focused on a specific objective, and an *intermediate* group gathers trade-off solutions. Hence there are two variants using three groups: A^3_{int} (resp. A^3_{div}), which uses an intensification (resp. diversification) strategy for the injection. Then we can refine the process to obtain $k_{\mathcal{G}} = 5$ (uniformly spread) groups in the decomposition, leading to two other variants: A^5_{int} and A^5_{div}. Moreover, we keep the extreme case where $k_{\mathcal{G}} = M$, creating as many groups as subproblems since it has been studied in [12]. In this case, each group is dedicated to one specific aggregation. More precisely, for $k \in \{1, \ldots, k_{\mathcal{G}}\}$, $g^k = w^k$. However, it may lead to a waste of resources since a lot of redundant knowledge between groups may exist. The last two variants are: A^M_{int} and A^M_{div}.

Algorithm 1: Knowledge Discovery MOEA/D Framework.

Input: M weight vectors $w^1, \ldots, w^M$. The number $k_{\mathcal{G}}$ of knowledge groups and the strategy $m_{\mathcal{G}}^{inj}$ (resp. $m_{\mathcal{G}}^{ext}$) for $\mathrm{K_{inj}}$ (resp. $\mathrm{K_{ext}}$).
Output: The external archive A

`/* Initialisation */`
$A \leftarrow \emptyset$; $S \leftarrow \emptyset$
$P \leftarrow$ random initial population (x^i for the i-th subproblem)
$\mathcal{G} \leftarrow \mathrm{createGroups}(k_{\mathcal{G}})$
for $i \in \{1, \ldots, M\}$ **do**
  $\mathcal{N}(i) \leftarrow$ indexes of the m closest weight vectors to w^i
  $Obj^i \leftarrow \{f_j(x^i) \mid 1 \leq j \leq n\}$

`/* Core of the algorithm */`
while *not stopping criterion satisfied* **do**
  for $i \in \{1, \ldots, M\}$ **do**
    $(i_1, i_2) \leftarrow \mathrm{Select}(\mathcal{N}(i))$
    $x \leftarrow \mathrm{PMX}(x^{i_1}, x^{i_2})$
    $x \leftarrow \mathrm{K_{inj}}(x, \mathcal{G}, i, m_{\mathcal{G}}^{inj})$
    $x \leftarrow \mathrm{LS}(x)$
    $S \leftarrow S \cup \{x\}$
    $\mathrm{updateNeighbors}(P, \mathcal{N}(i), x)$
  $A \leftarrow \mathrm{updateArchive}(A, S)$
  $\mathcal{G} \leftarrow \mathrm{K_{ext}}(\mathcal{G}, S, m_{\mathcal{G}}^{ext})$
  $S \leftarrow \emptyset$
return A

5 Bi-Objective Vehicle Routing Problem with Time Windows (bVRPTW)

5.1 Problem Description

The bVRPTW [23] is defined on a graph $G = (V, E)$, where $V = \{0, 1, \ldots, N\}$ is the set of vertices and $E = \{(i, j) \mid i, j \in V\}$ is the set of arcs. It is possible to travel from i to j, incurring a travel cost c_{ij} and a travel time t_{ij}. Vertex 0 represents the depot where a fleet of K identical vehicles with limited capacity $\mathcal{Q}$ is based. Vertices $1, \ldots, N$ represent the customers to be served, each one having a demand q_i and a time window $[a_i, b_i]$ during which service must occur. Vehicles may arrive before a_i. In that case, the driver has to wait until a_i to accomplish service incurring a waiting time. Arriving later than b_i is not allowed. It is assumed that all inputs are nonnegative integers. The bVRPTW calls for the determination of at most K routes such that the traveling cost and waiting time are simultaneously minimized and the following conditions are satisfied: (a) each route starts and ends at the depot, (b) each customer is visited by exactly one route, (c) the sum of the demands of the customers in any route does not exceed $\mathcal{Q}$, (d) time windows are respected.

5.2 Related Works

The original VRPTW aims to minimize the number of vehicles and the total traveling cost. In the literature, we find many lexicographic approaches that minimize the number of vehicles first and then the traveling cost. Nowadays, all Solomon's instances [20] can be optimally solved using an exact algorithm [17], however, the computational cost grows exponentially with the size of the instances. In practice, meta-heuristic algorithms can obtain a "good enough" solution in a short time and have the capacity to solve large-scale complex problems, which is more suitable for applications. Schneider et al. [19] proposed different granular neighborhoods to improve the local search performed. More recently Zhang et al. [27] designed a new Evolutionary Scatter Search with Particle Swarm Optimization, the so-called ESS-PSO, able to reach very good results on Solomon's instances in a small amount of time. Considering the multi-objective approaches, the literature is more sparse. Qi et al. [18] proposed a memetic algorithm based on MOEA/D to solve a bi-objective VRPTW. More recently, Moradi [16] integrated a learnable evolutionary model into a Pareto evolutionary algorithm.

5.3 Local Search and Knowledge Operators

The LS performed in Algorithm 1 is the same as described in [13]. Briefly, three neighborhood operators are used: swap, relocate, and 2-opt*. Initially, we shuffle the list of operators, so that they are not always applied in the same order. Then, for a given operator, we try to insert each customer to its best location, considering the possible moves allowed by the operator. If a better location is found for the customer, the process is repeated with another customer. When no more improving moves are found for all customers, the search stops and the next operator is picked up.

Now we define the $\mathrm{K_{inj}}$ and $\mathrm{K_{ext}}$ mechanisms related to the bVRPTW. Both mechanisms are based on the work of Arnold et al. [1]. They introduced PILS, an optimization strategy that uses frequent patterns from high-quality solutions, to explore high-order local-search neighborhoods. PILS has been hybridized with the Hybrid Genetic Search (HGS) of Vidal et al. [25] and the Guided Local Search (GLS) of Arnold and Sörensen [2] to solve the Capacitated Vehicle Routing Problem (CVRP) with good results. Given a solution x of the problem, $\mathrm{K_{ext}}$ extracts all patterns of x with a size between 2 and $size_p$, a user-defined parameter. The depot is not considered inside patterns. Patterns are sequences of consecutive customers in a route. For instance, a route $r = (0, v_1, \ldots, v_{|r|}, 0)$, contains $max(|r| - k + 1, 0)$ patterns of size k. Then for each extracted pattern, its frequency inside the groups updated is incremented. $\mathrm{K_{inj}}$ tentatively injects N_{Inj} patterns in the current solution x. Only improving patterns are kept in the solution, leading to a kind of elitism selection for patterns. To select a pattern we proceed as follows. First, the size of the pattern is randomly chosen among $\{2, \ldots, size_p\}$. It allows to not bias the selection towards smaller, more numerous, patterns. Then the pattern is randomly chosen among the $N_{Frequent}$ most frequent patterns of the same size. Here $N_{Frequent}$ is also a parameter of

the algorithm. When all the N_{Inj} patterns have been selected, they are injected one by one according to the following steps. Firstly arcs incident to a node of the pattern are removed and the nodes of the pattern are connected. This step creates several pieces of routes, that are reconnected to form a feasible solution. The reconnection is optimal, in the sense that all possibilities are tested. Because of time windows, we do not consider reversed patterns in our mechanism.

6 Experimental Setup

6.1 Solomon's Benchmark

We use Solomon's instances [20], of size 100, to evaluate the performance of all the seven variants presented in Section 4. This set is frequently used in the literature to evaluate the performance of multi-objective algorithms [9,18,16]. The set contains 56 instances divided into three categories according to the type of generation used, either R (random), C (clustered), or RC (random-clustered). The generation R (23 instances) randomly places customers in the grid, while the generation C (17 instances) tends to create clusters of customers. The generation RC (16 instances) mixes both generations. Each category is itself divided into two classes, either $1XX$ or $2XX$, according to the width of time windows. Instances of class $2XX$ have wider time windows than instances of class $1XX$, meaning that instances $1XX$ are more constrained.

6.2 Termination Criterion and Performance Assessment

The termination criterion of all the variants is set to 720 seconds. It allows us to obtain accurate and robust results. The quality of the fronts is evaluated with the unary hypervolume [29] (uHV), which measures the volume of the area dominated by the solutions of the front. Indeed, true Pareto fronts of the problem are not known, thus we can not use metrics that rely on them. For each instance, the two extreme points used to normalize the objectives of the solutions, are obtained through our experiments and are automatically updated when a new point is found. To compute the uHV we use the point $(1.001, 1.001)$ as a reference. The experiments are run on two computers "Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz", with 24 cores each. The variants have been implemented using the jMetalPy framework [5]. The code is available at https://github.com/Clegrandlixon/kdmoopy.

6.3 Tuning

Each algorithm is tuned with irace [14] to find a good setting of the parameters. To perform the tuning, we generated 96 new instances of size 100, by using the method described by Uchoa et al. [24] to mimic Solomon's instances. Each variant 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_{cro} for the crossover, p_{inj} for the injection, and p_{mut} for the LS. The granularity parameter δ [22] is used to prune the neighborhood during LS. The maximal size $size_p$ of the patterns extracted, and the number N_{Inj} of patterns injected, chosen among the $N_{Frequent}$ most frequent patterns. According to a preliminary study and existing works, we set $m = 1/4 \times M$ and $N_{Frequent} = 100$. We do not consider the number of groups $k_{\mathcal{G}}$ in the tuning, because we want to highlight its influence on the algorithm. We propose a different range of values for the seven remaining parameters (cf. Table 1), to define the configuration space in irace. We granted a budget of 2000 configurations over 8 iterations to irace. Each configuration is evaluated with the uHV metric. The best configurations are presented in Table 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 in [1]. Moreover, the probability of applying the LS seems low, but the LS is the most time-consuming step of the algorithm, mainly in the beginning when solutions are not optimized. With $p_{mut} = 0.10$ the LS represents already 50% of the running time. However, it represents only 60% when $p_{mut} = 0.25$. 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 LS.

Table 1. Parameter's space given to irace. The space contains 77175 configurations.

Name	Range
Population size: M	(20, 40, 60, 80, 100)
Granularity: δ	(10, 25, 50, 75, 100)
Probability of crossover: p_{cro}	(0.00, 0.10, 0.25, 0.50, 0.75, 0.90, 1.00)
Probability of mutation: p_{mut}	(0.00, 0.10, 0.25, 0.50, 0.75, 0.90, 1.00)
Maximum size of pattern: $size_p$	(5, 7, 10)
Number of patterns injected: N_{Inj}	(20, 40, 60)
Probability of injection: p_{inj}	(0.00, 0.10, 0.25, 0.50, 0.75, 0.90, 1.00)

7 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.

To that aim, 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. We recall that the termination criterion is set to 720 seconds for all variants.

Table 2. Best elite configurations returned by irace for each variant.

Params.	Base	A^3_{int}	A^3_{div}	A^5_{int}	A^5_{div}	A^M_{int}	A^M_{div}
M	60	60	40	40	20	40	20
m	15	15	10	10	5	10	5
δ	50	25	25	25	25	25	25
p_{cro}	0.50	0.50	0.90	0.50	0.90	0.50	0.75
p_{mut}	0.10	0.10	0.10	0.25	0.10	0.10	0.25
$size_p$	5	5	7	7	5	5	5
N_{Inj}	60	20	40	60	60	40	40
p_{inj}	0.75	0.75	1.00	1.00	0.90	1.00	0.90

For each category of instance (either R, RC, or C), we compute the average uHV obtained over the 30 runs. Then 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 it is not the case, we apply a pairwise Wilcoxon test with the *Bonferroni* correction to know which algorithms are statistically better. Finally, we define a fourth category *All*, containing all the instances, and we compute similarly the average ranks of each variant in that case.

8 Experimental Results and Discussion

In previous studies [12,13], we compared different instantiations of the framework with the original MOEA/D (i.e. without using the knowledge groups). It shows that using the knowledge discovery framework is beneficial. Table 3 (resp. Table 4) shows the average rank (resp. uHV) of each variant on each category of instance. The variant A^5_{int} always leads to the best average rank (1.46) and average uHV (0.828). Moreover, this variant returns statistically better results than the other variants. Hence it is interesting to use more than one group in a multi-objective context.

Using the diversification strategy with five groups worsened a lot the returned results. Indeed, A^5_{div} ranks 5.73 on average, which is the second highest rank. Only A^3_{int} has a higher rank. The other variant A^3_{div} has also a high rank, meaning that using three groups is a wrong choice in that context.

The variants A^M_{int} and A^M_{div} provide average uHV that are close in value. It is 0.767 for A^M_{int} and 0.770 for A^M_{div}. The conclusion is similar if we look at each category separately. Hence, when many groups are used, there is not a significant difference between intensification and diversification strategies for the injection.

Surprisingly, the Base variant returns good results, except on clustered instances. Hence 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, but further studies should investigate the behavior

of the procedure with different numbers of groups or consider the possibility of adapting the number of groups during execution.

Table 3. Average ranks of the variants according to their average uHV over the different categories of instance. Bold results are statistically significant.

Category	Base	A^3_{int}	A^3_{div}	A^5_{int}	A^5_{div}	A^M_{int}	A^M_{div}
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 4. Average uHV of the variants according to their average uHV over the different categories of instance. Bold results are statistically significant.

Category	Base	A^3_{int}	A^3_{div}	A^5_{int}	A^5_{div}	A^M_{int}	A^M_{div}
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

9 Conclusion

Integrating a knowledge discovery mechanism into a metaheuristic requires taking into account a lot of design aspects, summarized by the questions: *What*, *Where*, *When*, and *How* should the knowledge be extracted and injected. In this article, we mainly focused on the question *How*, while we considered existing works to answer the other questions. In particular, to answer the *How* question we have to consider how should interact the extraction and injection components of the KD mechanism, to be as efficient as possible.

As a contribution, we defined the notion of knowledge groups, studied in the literature, by giving a construction for 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. 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 near future, our framework will be compared to different state-of-the-art algorithms (e.g. NSGA-II, MODLEM), and different problems will also be investigated (e.g. bTSP). The tuning phase performed by irace provided similar configurations for each of the variants. Hence it will be interesting to investigate, whether with the same parameter configuration for all variants similar conclusions can be reached. Moreover, the number of groups will be considered as a parameter to be tuned in future works, to see if irace achieves similar conclusions. We would also like to investigate more deeply the impact of the strategies presented for injection and extraction. Finally, we aim to create an adaptive algorithm, which automatically adapts the number of groups and the strategies followed by the operators.

References

1. Arnold, F., Santana, Í., Sörensen, K., and Vidal, T. Pils: Exploring high-order neighborhoods by pattern mining and injection. *Pattern Recognition* (2021).
2. Arnold, F., and Sörensen, K. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research 105* (2019), 32–46.
3. Arnold, F., and Sörensen, K. What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Computers & Operations Research 106* (2019), 280–288.
4. Bandaru, S., Ng, A. H., and Deb, K. Data mining methods for knowledge discovery in multi-objective optimization: Part a-survey. *Expert Systems with Applications 70* (2017), 139–159.
5. Benitez-Hidalgo, A., Nebro, A. J., Garcia-Nieto, J., Oregi, I., and Del Ser, J. jmetalpy: A python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation 51* (2019), 100598.
6. Blot, A., Marmion, M., and Jourdan, L. Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation. *J. Heuristics 24*, 6 (2018), 853–877.
7. Corne, D., Dhaenens, C., and Jourdan, L. Synergies between operations research and data mining: The emerging use of multi-objective approaches. *European Journal of Operational Research 221*, 3 (2012), 469–479.
8. Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*, 2 (2002), 182–197.
9. Ghoseiri, K., and Ghannadpour, S. F. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing 10*, 4 (2010), 1096–1107.
10. Glover, F., and Laguna, M. Tabu search. In *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.
11. Knowles, J. D. *Local-search and hybrid evolutionary algorithms for Pareto optimization*. PhD thesis, University of Reading Reading, 2002.
12. Legrand, C., Cattaruzza, D., Jourdan, L., and Kessaci, M.-E. Enhancing moea/d with learning: Application to routing problems with time windows. In *Proceedings of the GECCO companion* (2022).
13. Legrand, C., Cattaruzza, D., Jourdan, L., and Kessaci, M.-E. New neighborhood strategies for the bi-objective vehicle routing problem with time windows. In *Proceedings of MIC'22* (2022).

14. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives 3* (2016), 43–58.
15. Lucas, F., Billot, R., Sevaux, M., and Sörensen, K. Reducing space search in combinatorial optimization using machine learning tools. In *International Conference on Learning and Intelligent Optimization* (2020), Springer, pp. 143–150.
16. Moradi, B. The new optimization algorithm for the vehicle routing problem with time windows using multi-objective discrete learnable evolution model. *Soft Computing 24*, 9 (2020), 6741–6769.
17. Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing 29*, 3 (2017), 489–502.
18. Qi, Y., Hou, Z., Li, H., Huang, J., and Li, X. A decomposition based memetic algorithm for multi-objective vehicle routing problem with time windows. *Computers & Operations Research 62* (2015), 61–77.
19. Schneider, M., Schwahn, F., and Vigo, D. Designing granular solution methods for routing problems with time windows. *European Journal of Operational Research 263*, 2 (2017), 493–509.
20. Solomon, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research 35*, 2 (1987), 254–265.
21. Talbi, E.-G. Machine learning into metaheuristics: A survey and taxonomy. *ACM Computing Surveys (CSUR) 54*, 6 (2021), 1–32.
22. Toth, P., and Vigo, D. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing 15*, 4 (2003), 333–346.
23. Toth, P., and Vigo, D. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.
24. Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research 257*, 3 (2017), 845–858.
25. Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research* (2014).
26. Wattanapornprom, W., Olanviwitchai, P., Chutima, P., and Chongstitvatana, P. Multi-objective combinatorial optimisation with coincidence algorithm. In *2009 IEEE Congress on Evolutionary Computation* (2009), IEEE, pp. 1675–1682.
27. Zhang, J., Yang, F., and Weng, X. An evolutionary scatter search particle swarm optimization algorithm for the vehicle routing problem with time windows. *IEEE Access 6* (2018), 63468–63485.
28. Zhang, Q., and Li, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation 11*, 6 (2007).
29. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Da Fonseca, V. G. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation 7*, 2 (2003), 117–132.