Solution-based Knowledge Discovery for Multi-objective Optimization

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

Solution-based Knowledge Discovery for Multi-objective Optimization

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. In the combinatorial optimization field, Knowledge Discovery (KD) mechanisms (e.g., data mining, neural networks) have received increasing interest over the years. KD mechanisms are based upon two main procedures, being the extraction of knowledge from solutions, and the injection of such knowledge into solutions. However, in a multi-objective (MO) context, the simultaneous optimization of many conflicting objectives can lead to the learning of contradictory knowledge. We propose to develop a Solution-based KD (SKD) mechanism suited to MO optimization. It is integrated within two existing metaheuristics: the Iterated MO Local Search (IMOLS) and the MO Evolutionary Algorithm based on Decomposition (MOEA/D). As a case study, we consider a bi-objective Vehicle Routing Problem with Time Windows (bVRPTW), to define accordingly the problem-dependent knowledge of the SKD mechanism. Our experiments show that using the KD mechanism we propose increases the performance of both IMOLS and MOEA/D algorithms.

Keywords: Knowledge Discovery, Multi-objective Optimization, Combinatorial Optimization, Routing Problems

1 Introduction

Efficient exploration of the search space is a key element of solving discrete optimization problems. Indeed, the search space is a set of regions containing solutions of different quality, from which it may be more or less difficult to escape. In this paper, we assume that solutions in the same region share common characteristics and, by wisely combining them, it is possible to reach more interesting regions with better-performing solutions. This assumption was verified on Solomon's benchmark, where 40% (resp. 25%) of arcs are shared between close (high-quality) solutions in instances with tight (resp. wide) time windows. In addition, this assumption is used by PILS [1], in which the structure of local optima guides the exploration towards regions that are difficult to reach by simple local search (LS) algorithms.

Knowledge Discovery (KD) processes have already received various interests, in single-objective [1, 11, 23] and in multi-objective (MO) [18, 24, 32] optimization. In particular, the concept of *innovization* introduced by Deb et al. [10], focuses on the dependency between the decision variables of a solution to help an optimization algorithm to reach specific parts of the objective space. We investigate a different approach by using the representation of the solution (here, as a permutation) instead of directly using the decision variables. Our approach finds echoes in Estimation of Distribution Algorithms [22], and more recently in linkage learning for permutation problems [13], although our approach exploits only frequent common structures found instead of using bayesian networks to learn more precise dependencies between variables.

In this article, we further develop the notion of *Solution-based Knowledge Discovery* (SKD) metaheuristics, by extending the construction of knowledge groups developed in [18]. This work leads to a new model coherent with MO optimization algorithms. We instantiate the model with the Iterated MOLS (IMOLS) [5] and the MO Evolutionary Algorithm based on Decomposition (MOEA/D) [34].

Since the extraction and injection procedures are themselves dependent on the problem studied, we decided to base our study on a bi-objective Vehicle Routing Problem with Time Windows (bVRPTW) already presented in [19]. In this problem, we minimize the total traveling cost and waiting time of drivers, which are conflicting objectives [7]. Indeed, when a driver arrives too early a waiting time is incurred, increasing the duration of the route for the driver. Considering real-life situations (e.g. food delivery, medical transportation), this additional time may incur satisfaction issues. Moreover, in the classical version of the VRPTW, the first objective to optimize is the number of vehicles, which is a discrete objective function, and then the total traveled distance is minimized as a second objective. However, the use of two continuous objectives (the total cost and the total waiting time) together allows the generation of fronts that contain, in general, many more non-dominated solutions, and it is better suited to a MO context, especially when knowledge is extracted from solutions.

The article is structured as follows: Section 2 presents MO optimization concepts and the IMOLS and MOEA/D metaheuristics. Our contribution, the SKD metaheuristic, is presented in Section 3. The model is integrated into IMOLS and MOEA/D in Section 4. Section 5 presents the problem and defines the knowledge to extract and inject in this context. In Section 6 our experimental protocol is presented and the results obtained are discussed. We conclude in Section 7.

2 Context

2.1 Multi-Objective Optimization

A *Multi-objective Combinatorial Optimization Problem* (MoCOP) is commonly formalized as follows [8]:

$$(MoCOP) = \begin{cases} Optimize\ F(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ s.t.\ x \in \mathcal{D}, \end{cases} \quad (1)$$

where $n \geq 2$ objective functions f_i have to be optimized, x is a vector of decision variables, and $\mathcal{D}$ is the set of solutions. The *objective space* is the image of F.

We say that a solution x dominates a solution y, noted $x \prec y$ in a minimization context, if and only if for all $i \in \{1 \dots n\}$, $f_i(x) \leq f_i(y)$ and there exists $j \in \{1 \dots n\}$ such that $f_j(x) < f_j(y)$. 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 to be *non-dominant*.

A *Pareto front* is defined as a set of non-dominated solutions. 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 $x \prec x^*$. We solve a MoCOP by finding 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 *true Pareto front*.

To compare Pareto fronts, and thus the algorithms providing them, many indicators have been developed [27]. In this paper, we consider the unary hypervolume (uHV) [35] metric. It is defined relatively to a reference point Z_{ref}, generally $(1.001, \dots, 1.001)$, and requires that the objectives of the solutions are normalized between 0 and 1. This indicator is to be maximized and allows a good evaluation of the front's accuracy, diversity, and cardinality. Geometrically, the uHV represents the volume of the objective space (bounded by Z_{ref}) covered by the members of a non-dominated set of solutions.

Many metaheuristics based on LS techniques, called MOLS [5], or using evolutionary algorithms, like Non-Domonated Sorting Genetic Algorithm (NSGA-II) [9], and MOEA/D [34], have been designed to solve MO problems. The following sections focus on iterated MOLS (Section 2.2) and MOEA/D (Section 2.3).

2.2 Iterated MOLS

A MOLS is an algorithm that iteratively explores solutions selected from a *current* population, by using LS procedures, accepts *candidates* during the search, and then updates an *external archive* of non-dominated solutions. We refer to the survey of Blot et al. [5], for a comprehensive overview of all possible mechanisms related to the conception of MOLS. A large part of MOLS algorithms is Pareto-based, meaning that they rely on the dominance criterion to accept neighbors during the search, contrarily to aggregation-based ones, which aggregate the different objectives to turn the problem into single-objective optimization. Among the Pareto-based algorithms, we find the Dominance-based MOLS (DMLS) algorithms [20] and the Pareto LS (PLS) [25].

In addition, it is possible to consider iterated MOLS (IMOLS), which mimic iterated LS, by using a *perturbation* procedure as a restart when a particular condition is reached (e.g., convergence of the MOLS).

2.3 MOEA/D

MOEA/D [34], is a genetic algorithm widely studied in the literature [33], approximating the Pareto front by decomposing the MO problem into several scalar

objective subproblems. There exist many ways to generate M scalar problems, but in every case, it requires a set of weight vectors $w^1, \ldots, w^M$. A weight vector $w = (w_1, \ldots, w_n)$ is 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. During the execution of MOEA/D, a population of solutions is maintained, where the i-th solution of the population is the best solution found for the i-th subproblem. Usually, a random permutation of the subproblems is defined in the beginning so that subproblems are always solved in the same order. Subproblems are iteratively solved, by applying a genetic step composed of crossover and mutation operators. When the subproblem i is optimized, two solutions from the population are selected for the crossover step. To perform that selection, two neighbor subproblems of subproblem i (included) are chosen, knowing that the neighborhood of a subproblem contains the m subproblems associated with the closest (for the Euclidean distance) weight vectors to weight vector w^i. The mutation is commonly replaced by a LS [6, 15, 17], to intensify the search in the regions identified with the crossover. If the final solution obtained is better than the initial solution considered for the subproblem, then it is replaced. The final solution is also tentatively added to an *external archive* storing the best non-dominated solutions found during the search and returned once the termination criterion of the algorithm is reached.

2.4 Unified View of IMOLS and MOEA/D

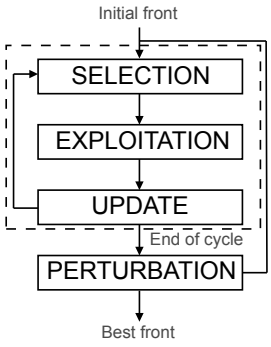

Fig. 1. The proposed unified view for IMOLS and MOEA/D metaheuristics.

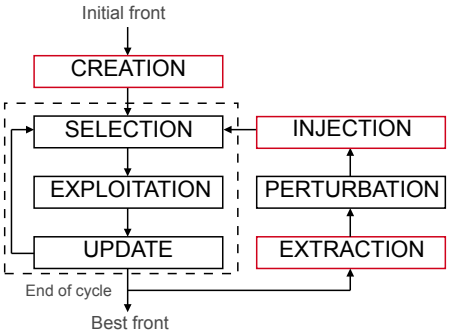

Fig. 2. The unified view integrating the three steps of the SKD.

This section shows the structural similarities between IMOLS and MOEA/D through a unified view. Our motivation behind this unification is to show how our knowledge discovery mechanism (SKD), presented in Section 3, can be integrated into algorithms sharing the same structural properties.

The IMOLS and MOEA/D frameworks can be abstracted with the following four main steps: `Selection`, `Exploitation`, `Update`, and `Perturbation`. The Figure 1 shows how these steps interact together. The `Exploitation` 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`, `Exploitation`, and `Update`). When a cycle ends, a `Perturbation` occurs, if a specific criterion is met, to update the current population before the next `Selection`, and so forth, until a termination criterion is reached (generally based on time or number of iterations). An external archive is maintained to track the best non-dominated solutions found and is finally returned. The steps are discussed below with details about their instantiation in IMOLS and MOEA/D.

The `Selection` step chooses one or several solutions to explore in the current population, 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 IMOLS, the selection is directly performed from the current population. In MOEA/D, each subproblem is sequentially selected, and consequently, the associated solution is explored.

`Exploitation` is the intensification step of the algorithm where the search focuses on specific regions of the search space. During this step, the neighborhood of the selected solutions is exploited, until a criterion is reached, to generate new (better) candidate solutions. In IMOLS, the exploitation consists of accepting either non-dominated or dominating neighbors of the selected solutions, considering a reference set. Consequently, many iterations are needed to reach out to a Pareto local optima. In MOEA/D the exploitation consists of applying a single-objective LS [14], for the selected subproblem, until a local optimum is reached.

When new solutions are found after the exploitation, the `Update` step tentatively integrates them into the external archive and the current population. While the external archive generally relies on bounded mechanisms, it is possible to adopt different strategies to update the current population (e.g., replacement of the solution explored, keeping non-dominated solutions in priority).

In neighborhood-based algorithms and evolutionary ones, 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. 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 IMOLS, the perturbation relies on LS mechanisms. In MOEA/D, it corresponds to a crossover.

In the next section, we present the SKD mechanism. Its integration in IMOLS and MOEA/D is presented in Section 4.

3 Solution-based Knowledge Discovery

3.1 Global Overview and Main Issues

In [18], the concept of *knowledge groups* is introduced. The idea is to divide the objective space into regions each representing a *knowledge group*. A knowledge

group gathers structural elements of the solutions of the same region. If the approach was promising, many obstacles remain and have to be tackled to ensure a model, that can be easily integrated into various MO algorithms. The first issue concerns the creation of knowledge groups. The creation proposed in [18] was dependent on the aggregations used in MOEA/D, which highly restricted its range of applications. Our proposition, detailed in Section 3.2 overcomes this limit, allowing many more integration possibilities. Then, the interaction between the extraction (resp. injection) procedure and the groups newly created, is presented in Section 3.3 (resp. Section 3.4). Although the interactions remain similar to those described in [18], we present them in a more flexible way to allow a better integration in metaheuristics.

3.2 Creation of Knowledge Groups

The problem is associating each knowledge group with a region of the objective space. We consider that each group is related to a representative, inducing the region of the group. In the following, we consider, for simplicity purposes, a bi-objective space. We propose two strategies to create the k_G representatives of the groups. The first one, represented in Figure 3, with $k_G = 5$ representatives named g^i, generates k_G uniformly spread weight vectors. Then, to evaluate the proximity of a solution to a group we aggregate the objectives of the solution by using the weight vector associated with the group. This strategy is a simple variant of [18] allowing to use it in other algorithms than MOEA/D. The second strategy, represented in Figure 4, links the extreme points of the current front with a straight line. 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. With this second strategy, it is possible (and recommended) to dynamically update the representatives of each group, before the extraction, if the extreme points vary. In both figures, each point of the Pareto front is linked to its closest representative, which leads to different distributions for each construction.

3.3 Extraction and Knowledge Groups

The extraction procedure is presented in Algorithm 1. It is possible to deactivate the extraction until a certain execution time is reached, to balance low-quality and high-quality solutions. In the following, we activate the procedure with no delay, at the beginning of the execution, since an initial front is provided.

For the extraction procedure, a *learning set* L of solutions generated during the execution of the algorithm is provided. However, MO algorithms explore plenty of solutions during their execution (e.g., MOLS), and learning from all of them would scramble the knowledge added to the groups. Consequently, a subset of L that contains only the solutions that undergo the extraction procedure is considered. Here, we suggest to keep only the non-dominated solutions of L. In particular, it allows the learning to focus on the most interesting solutions. Please

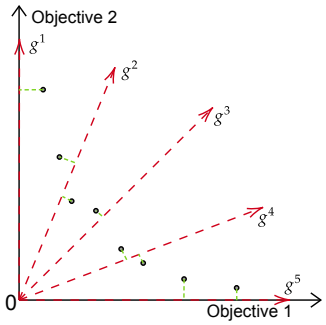

Fig. 3. Creation of groups based on weight vectors (denoted WG).

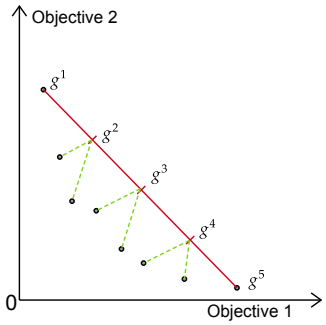

Fig. 4. Creation of groups based on extrema points (denoted EG).

note that other possibilities may be taken into account as a random sample or a mix of dominated and 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 1) of x, following the evaluation of the proximity between a solution and a group provided in Section 3.2. The parameter d_e allows the control of the diversification of the mechanism: smaller values correspond with fewer groups being updated. Then, the elements of knowledge are added to the corresponding groups, and a score (e.g., the frequency of appearance) reflecting the relevance of each element is updated. However, 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 1 are presented in Section 4 since they are algorithm-dependent. The functions `Extract` (l.3) and `Update` (l.5), being problem-dependent, are presented in Section 5.2.

Algorithm 1: 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 $\quad \mathcal{K} \leftarrow$ `Extract`(x)
4 $\quad G = \{G_1, \ldots, G_{d_e}\} \leftarrow$ `SelectGroups`$(\mathcal{G}, d_e, x)$
5 $\quad$ `Update`$(G, \mathcal{K})$
6 $L \leftarrow \emptyset$
7 **return** $\mathcal{G}$

3.4 Injection and Knowledge Groups

At that point, all the solutions from the current population undergo the injection procedure presented in Algorithm 2.

First, the knowledge to inject has to be selected. Likewise for the extraction (see the `SelectGroups` function), a subset of d_i groups containing the closest groups to x is created. Again, this parameter controls the diversification of the mechanism. The function `SelectOne` applied to the d_i candidate groups selects the final group, that will produce the knowledge to inject in the solution x. It 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 selection of the knowledge should use 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 non-dominating x') during the injection of k are added to a set S'. 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.

`SelectOne` (l.2), `SelectKnowledge` (l.3), and `SelectNextSolution` (l.7) used in Algorithm 2 are defined in Section 4 since they are algorithm-dependent. The problem-dependent function `Inject` (l.6) is defined in Section 5.2.

Algorithm 2: Injection procedure.

Input: $\mathcal{G}$ the knowledge groups, x the current solution, and d_i the number of candidate groups.
Output: Accepted solutions.

```
1  G = {G_1, ..., G_{d_i}} ← SelectGroups(𝒢, d_i, x)
2  G' ← SelectOne(G)
3  𝒦 ← SelectKnowledge(G')
4  S ← ∅
5  x' ← x
6  for k ∈ 𝒦 do
7  |  S' ← Inject(k, x')
8  |  x' ← SelectNextSolution(S', x')
9  |  S ← S ∪ S'
10 return S
```

3.5 Integration of SKD into the Unified View

The Solution-based Knowledge Discovery (SKD) uses knowledge groups and the procedures of extraction and injection suited to MO algorithms. The Unified

View presented in Section 2.4 contains successive steps of intensification and diversification. The intensification is usually the core of the MO 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 MO algorithms using our unified view (see Figure 1). We aim to improve the diversification phase, by exploring larger regions of the search space with the knowledge stored in the groups.

At the beginning of the execution, given the initial front provided, the knowledge groups are created following one strategy presented in Section 3.2.

Applying the extraction procedure at every iteration would result in a lot of noise for the knowledge groups. 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.

Any solution can undergo the injection but, like the `Extraction`, applying it to all the explored solutions would waste computational resources. 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 or after the `Extraction` otherwise. After the injection, a new cycle (i.e., an intensification step) is started by updating the archive and the current population. These remarks lead to the conception of the model presented in Figure 2.

4 SKD for IMOLS and MOEA/D

4.1 SKD for IMOLS

We follow the DMLS model originally introduced by Liefooghe et al. [20]. The problem's representation, the solution evaluation, and the neighborhood structure are defined in Section 5.1 with the problem. The algorithm starts from an initial front given by the user, integrated into a bounded archive, $\mathcal{A}$, of size U_a, representing the current population. The archive is bounded by using the crowding distance [9]. Then, U_c randomly selected solutions from the archive (among the not entirely explored ones) form the set to explore. The DMLS algorithm iteratively explores the selected solutions. During the LS, the neighborhood of a solution x is explored until a non-dominated solution, considering all solutions of $\mathcal{A}$, is found [4]. If no solution is found, x is tagged as explored and is no longer selected during the current cycle, moreover tagged solutions cannot be selected during the LS. If any, the accepted solution is tentatively added to $\mathcal{A}$.

In the iterated variant, we manage a second (unbounded) archive, $\mathcal{A}^*$, containing the best non-dominated solutions found during the execution. After l_c iterations (denoting the length of a cycle), the uHV of $\mathcal{A}$ is evaluated, and the solutions of $\mathcal{A}$ are integrated into $\mathcal{A}^*$. Before starting a new cycle, if all solutions of $\mathcal{A}$ 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 tags are removed from solutions, all elements from $\mathcal{A}$ are tagged as explored and a new archive $\mathcal{A}$ is created by perturbing solutions from $\mathcal{A}^*$. To perturb a solution

x, we apply three moves of the LS, with the following acceptance criterion: a solution y is accepted when $\forall i \in \{1, \ldots, n\}, f_i(y) \leq (1+\epsilon_p) \cdot f_i(x)$, with $\epsilon_p \in \mathbb{R}^+$, allowing a slight relaxation of the objectives of x to test the dominance relation. This version of IMOLS is called $\mathrm{R_{IMOLS}}$.

Following the steps presented in Section 3.5, the extraction and injection procedures are added to $\mathrm{R_{IMOLS}}$. The variant using the weights (resp. the extrema) to create the groups is called $\mathrm{WG_{IMOLS}}$ (resp. $\mathrm{EG_{IMOLS}}$). 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 $\mathcal{A}$ after the exploration step should be added to the learning set, since it may produce interesting knowledge to exploit. We only keep non-dominated solutions to filter the solutions of the learning set. Concerning the injection procedure, it is sequentially applied to all the solutions from $\mathcal{A}$. The `SelectOne`, `SelectKnowledge`, and `SelectNextSolution` functions from Algorithm 2 are defined hereafter. The `SelectOne` function chooses the group that gives the knowledge to inject. Here, we choose the group randomly. For the `SelectKnowledge` function, we rely on the scores 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 [1]. More details are given in Section 5.2 in the context of the problem. Finally, for the `SelectNextSolution` function, the initial solution is returned (x in Algorithm 2). 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.

4.2 SKD for MOEA/D

Now we provide an instantiation of MOEA/D, called $\mathrm{R_{MOEAD}}$, following the framework described in Section 2.3. We consider scalar problems obtained with a weighted sum of the objectives. Contrary to Tchebycheff decomposition, it does not require a reference point. Given a weight vector w, the fitness of a solution is defined as the following quantity: $g(x|w) = \sum_{i=1}^{n} w_i \cdot f_i(x)$. However, all the solutions of the true Pareto front can not be obtained with such aggregations. In the following, we generate M weight vectors uniformly distributed, assuming that is enough to obtain diverse subproblems. A Partially Mapped Crossover (PMX) [16] is applied with probability p_{pmx}. Among the two generated solutions, only one is randomly chosen to keep the population's size constant. When the crossover is not applied, the solution associated with the i-th subproblem is kept. The mutation is a LS detailed in Section 5.1, and applied with probability p_{ls}.

Following the steps presented in Section 3.5, the extraction and injection procedures are added to $\mathrm{R_{MOEAD}}$. The variant using the weights (resp. the extrema) to create the groups is called $\mathrm{WG_{MOEAD}}$ (resp. $\mathrm{EG_{MOEAD}}$). For the extraction procedure, we keep the idea exposed in Section 4.1. Each solution tentatively added to the external archive (`Update` step of Figure 1), is added to the learning set. Then, the knowledge is extracted from non-dominated solutions of the learning set. The injection procedure is applied to all the solutions

of the current population (i.e., the best solution of each subproblem). The functions `SelectOne` and `SelectKnowledge` are the same as presented in Section 4.1, but `SelectNextSolution` differs. The next solution is the best (considering the aggregation of the associated subproblem) accepted during the injection.

5 Case Study: bi-objective VRPTW

5.1 Presentation

See [19] for a detailed formalization of the bi-objective VRPTW (bVRPTW) considered. The bVRPTW calls for the determination of routes such that the traveling cost (i.e. the sum of the Euclidean distance between consecutive customers) and the total waiting time (i.e. the sum of the waiting times induced by an early arrival to deliver a customer) are simultaneously minimized. Moreover, each solution of the bVRPTW needs to satisfy the following constraints: each route starts and ends at a specific location (called depot), each customer is visited by exactly one route, the sum of the demands of the customers in any route does not exceed the capacity of the vehicles, and time windows are respected (late arrivals are not allowed).

A solution to the problem is encoded as a customer permutation and evaluated with the split algorithm provided by [26], providing a feasible solution. For this study, we consider the operators `Relocate`, `Swap`, and `2-opt`*. These simple operators are largely used in LS algorithms for routing problems [28] since they can produce a large neighborhood, and allow an easy incremental evaluation. The `Relocate` operator moves one customer from its current position to another location. The `Swap` operator exchanges in the solution the position of two customers. The `2-opt`* operator generalizes the 2-opt, by involving different routes. In $\mathrm{R}_{\mathrm{MOEAD}}$ a Randomized Variable Neighborhood Descent is applied for exploitation [19, 30], where the order of the operators is kept during descent (until a local optimum is reached) but shuffled each time the LS is applied. In $\mathrm{R}_{\mathrm{IMOLS}}$, the order of the operators is randomized too, but the search stops at the first accepting neighbor. Only feasible solutions are considered.

5.2 Knowledge Related to a Solution

In this section, the remaining `Extract` and `Update` (resp. `SelectKnowledge` and `Inject`) functions from the Algorithm 1 (resp. Algorithm 2), are defined to suit the bVRPTW context. These functions are inspired by the Pattern Injection LS (PILS) method [1]. It is an optimization method relying on frequent patterns from high-quality solutions to explore vast neighborhoods. PILS has already been integrated into the Hybrid Genetic Search [31] and the Guided LS [2] to solve the Capacitated Vehicle Routing Problem (CVRP).

In routing problems, patterns are defined as sequences of consecutive customers on a route without the depot. Those with a size between 2 and s_p are extracted from generated solutions by `Extract`. In particular, a route $r =$

$(0, v_1, \ldots, v_{|r|}, 0)$, contains $max(|r|-k+1, 0)$ patterns of size k. Once the patterns are extracted, `Update` adds them to corresponding groups. If the pattern already belongs to the group, its frequency is incremented. Otherwise, it is added with a frequency of one. Different groups may have different frequencies for the same pattern. A pattern becomes *frequent* when its frequency exceeds a threshold l_f

For the injection, `SelectKnowledge` randomly selects a pattern size among $\{2, \ldots, s_p\}$ to not bias the selection towards smaller, more numerous, patterns. Then, N_i patterns are randomly chosen among the N_f most frequent patterns of the corresponding size (without repetition). Only patterns tagged *frequent* can be selected. Given a pattern and a solution x, the `Inject` function creates a solution from x containing the pattern provided, as explained in [1] (except that reversed fragments are discarded due to time windows). First, arcs connecting the pattern are removed, thus creating partial routes, which are reconnected (with an exhaustive search) to form feasible solutions. Several solutions may be accepted during the reconstruction step. In IMOLS, all non-dominated solutions are accepted, while solutions with better fitness are accepted in MOEA/D.

6 Experimental Study

6.1 Choice of Parameters Value

The tuning of the parameters for the $\mathrm{R_{MOEAD}}$ variant comes from previous tuning with irace [21] and we refer to [19] for a detailed analysis of the parameters. $M = 40$ 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 $p_{pmx} = 1.00$, and the LS with probability $p_{ls} = 0.10$.

For $\mathrm{R_{IMOLS}}$, the parameters are chosen to be fair with $\mathrm{R_{MOEAD}}$. The archive limit is set to $U_a = 40$. Each iteration, $U_c = 1$ solution is explored. The perturbation occurs when the uHV does not increase by at least $e_{uHV} = 10^{-2}$, and during the perturbation, $\epsilon_p = 0.02$. A cycle performs $l_c = 100$ iterations.

The parameters value of $\mathrm{EG_{MOEAD}}$ and $\mathrm{WG_{MOEAD}}$ (resp. $\mathrm{EG_{IMOLS}}$ and $\mathrm{WG_{IMOLS}}$) are similar and their values follow the recommendation made in [19]. p_{pmx} is set to 0.50. There are $k_G = 20$ knowledge groups. The maximum size s_p of extracted patterns is set to 8 (resp. 5) for instances of class 2 (resp. class 1) since large (resp. short) routes are designed. The knowledge is added to $d_e = 1$ group, and the knowledge to inject is provided by at most $d_i = 1$ group. $N_i = 100$ 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 $l_f = 2$.

6.2 Experimental Protocol

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 [3]. The source code and our results are available on a Git[3].

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

The Solomon [29] and the Gehring and Homberger [12] benchmarks are commonly used to evaluate the performance of MO algorithms. Solomon's benchmark contains instances with up to 100 customers. Customers can be randomly located (R), clustered (C), or mixed (RC). Each category is divided into two classes. Instances of class 1 have tighter time windows than instances of class 2, which are less constrained. Gehring and Homberger's benchmark uses a similar instance generation but considers a larger number of customers.

To fairly compare the algorithms, they are all initialized with the same fronts. Hence, we generate 30 initial fronts (available on Git) for each instance. In IMOLS, the initial front is directly used as the initial population, however, in MOEA/D, each subproblem is initialized with the best solution of the front.

The six algorithms are then executed over 30 seeds on each instance. 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.

6.3 Results

Table 1. Average uHV ($\times 10^3$) of the algorithms on the different categories of instances. R_{MOEAD} and R_{IMOLS} are the reference algorithms. EG_{MOEAD}, WG_{MOEAD}, EG_{IMOLS}, and WG_{IMOLS} are the learning variants. Gray cells are statistically better comparing all algorithms, i.e., the six rows. Bold values represent the best-performing algorithms when MOEA/D (resp. IMOLS) variants are compared together (i.e., three rows each).

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

Table 1 summarises the results obtained. Detailed results per instance are available on the Git provided. First, R_{IMOLS} returns better results than R_{MOEAD} 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 bigger 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_{IMOLS} in all instances. The same conclusion holds

for R_{MOEAD} 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. 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 (see Figure 5 for comparison).

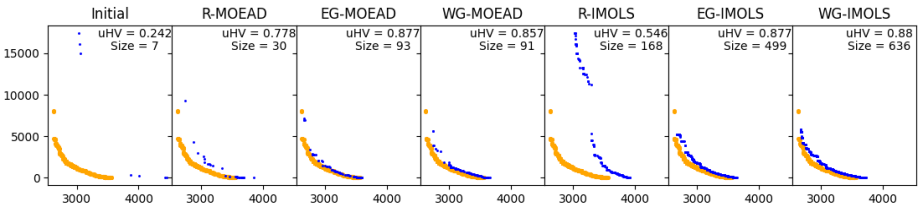

Fig. 5. Results of the execution on instance RC2_2_6 (run 6), from the Gehring and Homberger set. The associated hypervolume and size of the final fronts (blue dots) are shown, as well as the reference front (orange dots).

7 Conclusion

In this paper, we proposed to extend the mechanism of [18] to develop a solution-based KD mechanism, called SKD, which extracts knowledge from solutions and injects knowledge to explore new regions of the solution space. The mechanism is mainly based on the creation of knowledge groups, dividing the objective space. Here, two creation strategies for the groups are developed and compared. Any MO algorithm that can be an instantiation of the unified algorithm presented in Figure 1, can be extended by integrating SKD as shown in Figure 2. Then, we integrated SKD into two MO algorithms (IMOLS and MOEA/D) to solve a bi-objective routing problem, and we defined accordingly the algorithm-dependent components and the problem-dependent knowledge. Experiments were performed over instances with different characteristics of size and structure. In most cases, using SKD increases the performance of the original algorithm, showing an interest in our developed mechanism. Moreover, creating the groups with the weight vector strategy seems more profitable.

Future works should investigate the impact of the initialization on SKD. More precisely, the time to start the learning may impact the resolution and further analysis may be beneficial. Finally, it could be interesting to investigate the possibility of transferring the knowledge learned from one instance to another without starting from scratch again. This may be done by detecting similarities in the instances.

Acknowledgements. This work has been supported by the French National Research Agency through the AI_PhD@Lille program (ANR-20-THIA-0014). This support is gratefully acknowledged.

References

1. Arnold, F., Santana, Í., Sörensen, K., Vidal, T.: PILS: Exploring high-order neighborhoods by pattern mining and injection. Pattern Recognition (2021)
2. Arnold, F., Sörensen, K.: Knowledge-guided local search for the vehicle routing problem. Computers & Operations Research **105**, 32–46 (2019)
3. Benitez-Hidalgo, A., Nebro, A.J., Garcia-Nieto, J., Oregi, I., Del Ser, J.: jMetalPy: A python framework for multi-objective optimization with metaheuristics. Swarm and Evolutionary Computation **51**, 100598 (2019)
4. Blot, A., Jourdan, L., Kessaci, M.É.: 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. pp. 227–234 (2017)
5. Blot, A., Marmion, M., Jourdan, L.: Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation. J. Heuristics **24**(6), 853–877 (2018). https://doi.org/10.1007/s10732-018-9381-1, `https://doi.org/10.1007/s10732-018-9381-1`
6. Bossek, J., Grimme, C., Meisel, S., Rudolph, G., Trautmann, H.: Local search effects in bi-objective orienteering. In: Proceedings of the genetic and evolutionary computation conference. pp. 585–592 (2018)
7. Castro-Gutierrez, J., Landa-Silva, D., Pérez, J.M.: Nature of real-world multi-objective vehicle routing with evolutionary algorithms. In: 2011 IEEE International Conference on Systems, Man, and Cybernetics. pp. 257–264. IEEE (2011)
8. Coello, C.A.C., Dhaenens, C., Jourdan, L.: Multi-objective combinatorial optimization: Problematic and context. In: Advances in multi-objective nature inspired computing, pp. 1–21. Springer (2010)
9. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
10. Deb, K., Srinivasan, A.: Innovization: Innovating design principles through optimization. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation. pp. 1629–1636 (2006)
11. Fitzpatrick, J., Ajwani, D., Carroll, P.: Learning to prune electric vehicle routing problems. In: International Conference on Learning and Intelligent Optimization. pp. 378–392. Springer (2023)
12. Gehring, H., Homberger, J.: A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Proceedings of EUROGEN99. vol. 2, pp. 57–64. Springer Berlin (1999)
13. Guijt, A., Luong, N.H., Bosman, P.A., de Weerdt, M.: On the impact of linkage learning, gene-pool optimal mixing, and non-redundant encoding on permutation optimization. Swarm and Evolutionary Computation **70**, 101044 (2022)
14. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations and applications. Elsevier (2004)
15. Knowles, J.D.: Local-search and hybrid evolutionary algorithms for Pareto optimization. Ph.D. thesis, University of Reading, Reading (2002)

16. Kora, P., Yadlapalli, P.: Crossover operators in genetic algorithms: A review. International Journal of Computer Applications **162**(10) (2017)
17. Land, M.W.S.: Evolutionary algorithms with local search for combinatorial optimization. University of California, San Diego (1998)
18. Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, M.E.: Improving MOEA/D with knowledge discovery. application to a bi-objective routing problem. In: EMO: 12th International Conference, 2023, Proceedings. pp. 462–475. Springer (2023)
19. Legrand, C., Cattaruzza, D., Jourdan, L., Kessaci, M.E.: Improving neighborhood exploration into MOEA/D framework to solve a bi-objective routing problem. International Transactions in Operational Research (2023)
20. Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., Talbi, E.G.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics **18**, 317–352 (2012)
21. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
22. Lozano, J.A.: Towards a new evolutionary computation: advances on estimation of distribution algorithms, vol. 192. Springer Science & Business Media (2006)
23. Lucas, F., Billot, R., Sevaux, M., Sörensen, K.: Reducing space search in combinatorial optimization using machine learning tools. In: International Conference on Learning and Intelligent Optimization. pp. 143–150. Springer (2020)
24. 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), 6741–6769 (2020)
25. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) Metaheuristics for Multiobjective Optimisation. pp. 177–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
26. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & operations research **31**(12), 1985–2002 (2004)
27. Riquelme, N., Von Lücken, C., Baran, B.: Performance metrics in multi-objective optimization. In: 2015 Latin American Computing Conference (CLEI). pp. 1–11. IEEE (2015)
28. Schneider, M., Schwahn, F., Vigo, D.: Designing granular solution methods for routing problems with time windows. European Journal of Operational Research **263**(2), 493–509 (2017)
29. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)
30. Subramanian, A., Uchoa, E., Ochi, L.S.: A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research **40**(10), 2519–2531 (2013)
31. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research (2014)
32. Wattanapornprom, W., Olanviwitchai, P., Chutima, P., Chongstitvatana, P.: Multi-objective combinatorial optimisation with coincidence algorithm. In: 2009 IEEE Congress on Evolutionary Computation. pp. 1675–1682. IEEE (2009)
33. Xu, Q., Xu, Z., Ma, T.: A survey of multiobjective evolutionary algorithms based on decomposition: variants, challenges and future directions. IEEE Access **8**, 41588–41614 (2020)

34. Zhang, Q., Li, H.: MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on evolutionary computation **11**(6), 712–731 (2007)
35. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Da Fonseca, V.G.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on evolutionary computation **7**(2), 117–132 (2003)