New Neighborhood Strategies for the Bi-objective Vehicle Routing Problem with Time Windows

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

New Neighborhood Strategies for the Bi-Objective Vehicle Routing Problem with Time Windows

Clément Legrand[1], Diego Cattaruzza[2], Laetitia Jourdan[1], and Marie-Eléonore Kessaci[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. Local search (LS) algorithms are efficient metaheuristics to solve vehicle routing problems (VRP). They are often used either individually or integrated into evolutionary algorithms. For example, the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) can be enhanced with a local search replacing the mutation step based on a single move operator traditionally. LS are based on an efficient exploration of the neighborhoods of solutions. Many methods have been developed over the years to improve the efficiency of LS. In particular, the exploration strategy of the neighborhood and the pruning of irrelevant neighborhoods are important concepts that are frequently considered when designing a LS. In this paper, we focus on a bi-objective vehicle routing problem with time windows (bVRPTW) where the total traveling cost and the total waiting time have to be minimized. We propose two neighborhood strategies to improve an existing LS, efficient on the single-objective VRPTW. First, we propose a new strategy to explore the neighborhood of a solution. Second, we propose a new strategy for pruning the solution neighborhood that takes into account the second criterion of our bVRPTW namely the waiting time between customers. Experiments on Solomon's instances show that using LS with our neighborhood strategies in the MOEA/D gives better performance. Moreover, we can achieve some best-known solutions considering the traveling cost minimization only.

Keywords: VRP · Multi-Objective Optimization · MOEA/D · Local Search.

1 Introduction

Local search (LS) are known to be powerful algorithms used in evolutionary algorithms to improve their performance [7]. Indeed, LS are able to intensify the search by focusing on a specific region of the space. LS are based on neighborhood operators that link solution together and a neighborhood exploration

strategy define how the neighbors are explored and when the exploration is stopped. Here, we are mainly interested in the Vehicle Routing Problem with Time Windows (VRPTW). It is a routing problem where time is considered as an important resource and where customers must be served within a fixed time interval. Some LS have been developed for this problem and consequently many neighborhoods are available. For our study, we consider the same neighborhood as defined in [16]. The operators are: relocate, swap and 2-opt*. These operators are commonly used in routing problems, since they are simple operators and they are able to produce a large neighborhood. However the LS steps are time-consuming, that is why different strategies exist to speed-up the search and reduce the time allocated to the neighborhood exploration. First LS can be applied following a probability, that is a parameter of the final algorithm. Indeed, not applying the LS may have a positive impact since it brings more diversity to the solutions. Second, the exploration of the neighborhood can be done entirely with strategy *best*, or partially with strategy *first*. For the strategy *best*, all neighbors are considered and the best one is selected. For the strategy *first*, the neighbor are evaluated one by one and the exploration is stopped as soon as an improving neighbor is found and selected. Since routing problems produce large neighborhood pruning techniques have been designed to avoid irrelevant moves. The most common one is probably the granular search [18]. It is based on the idea that two distant clients have a low chance to produce a relevant arc.

In this paper, we study a Bi-objective VRPTW (bVRPTW), that is a Multi-objective Combinatorial Optimization Problems (MoCOPs) [5]. Such problems are frequent in the industry where decision-makers are interested in optimizing several conflicting objectives at the same time. The objectives considered are the total traveling cost (a classical objective in routing problems), and the total waiting time incurred when drivers arrive before the opening of the time window. Although this objective has not received much attention in the literature [4,25], it is relevant when considering the transportation of people or medical goods. Indeed, when a patient has a medical appointment, we do not want that he waits too much. Note that, here we only consider the minimum possible waiting time incurred by time windows. Moreover, in real problems, there is more than one way to link two customers considering the traveled distance, and the traveling time. However in the Solomon's instances, that are used for our experiments, the traveling time between two customers corresponds to the distance between them, which is a strong hypothesis.

To solve this problem, we use MOEA/D, a Multi-Objective Evolutionary Algorithm based on Decomposition [24] where the mutation step is replaced by a local search. The contribution of the paper is to present neighborhood strategies that are better adapted to the bVRPTW. First, we present a new strategy to explore the neighborhood of bVRPTW solutions inspired from state of the art for permutation flowshop. Second, we propose a pruning technique that considers not only the distance between the clients, but also their respective time window.

The remaining of the paper is structured as follows. After a brief presentation of multi-objective problems, the bVRPTW studied is described in Section 2, as

well as related works. Section 3 first focuses on the MOEA/D based framework used for this study, and then presents the different mechanisms proposed to improve the local search step. Section 4 describes the benchmark and how the algorithms were tuned. Then our experimental protocol is presented. Section 5 compares the results obtained for each combination of the mechanisms for the local search. Section 6 compares the results obtained with the best variant from Section 5, and the results obtained with state of the art algorithms for the VRPTW. Finally, Section 7 concludes and presents perspectives for this work.

2 Bi-Objective Routing Problem with Time Windows

2.1 Multi-Objective Optimization

In the following we formalize *Multi-objective Combinatorial Optimization Problems* (MoCOPs) [5]:

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

where n is the number of objectives ($n \geq 2$), x is the vector of decision variables, $\mathcal{D}$ is the (discrete) set of feasible solutions and each objective function $f_i(x)$ has to be optimized (i.e. minimized or maximized). In multi-objective optimization the objective function F defines a so-called objective space denoted by $\mathcal{Z}$. For each solution $x \in \mathcal{D}$ there exists a point in $\mathcal{Z}$ defined by $F(x)$.

A *dominance* criterion is defined to compare solutions together: a solution x dominates a solution 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 $f_j(x) < f_j(y)$. A partial order is defined on the solutions by $x < y$ if and only if x dominates y.

Then a set of non dominated solutions is called a *Pareto front*. 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 dominates x^*. Resolving a MoCOP involves finding all the Pareto optimal solutions which form the *Pareto optimal set*. The *true Pareto front* of the problem is the image of the Pareto optimal set by the objective function.

Over the years, many metaheuristics based on local search techniques or using evolutionary algorithms [3] have been designed to solve multi-objective problems. Moreover, many tools [14] have been developed to assess and compare the performance of multi-objective algorithms. In this paper, we use the unary hypervolume (HV) [26], which is a metric defined relatively to a reference point Z_{ref}. This indicator evaluates accuracy, diversity, and cardinality of the front, and it is the only indicator with this capability. Moreover, it can be used without knowing the true Pareto front of the problem. It reflects the volume covered by the members of a non dominated set of solutions. Thus, the larger the hypervolume, the better the set of solutions.

2.2 bVRPTW and Related Works

The bVRPTW [19] considered in this work 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 in 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, a time window $[a_i, b_i]$ during which service must occur, and a service time s_i estimating the required time to perform the delivery. Vehicles may arrive before a_i. In that case, the driver has to wait until a_i to accomplish service incurring in a waiting time. Arriving later than b_i is not allowed. It is assumed that all inputs are nonnegative integers. We recall that a *route* r is an elementary cycle on G that contains the depot (that is vertex 0) and can be expressed as a sequence of vertexes $r = (v_0, v_1, \ldots, v_R, v_{R+1})$ where $v_0 = v_{R+1} = 0$ and vertexes $v_1, \ldots, v_R$ are all different. The cost c_r of a route r is then given as the sum of traveling costs on arcs used to visit subsequent vertexes, that is $\sum_{i=0}^{R} c_{v_i, v_i+1}$. A solution x can be represented as a set of (possibly empty) K routes, that is $x = \{r_1, \ldots, r_K\}$, and its cost is expressed as:

$$f_1(x) = \sum_{k=1}^{K} c_{r_k} \tag{2}$$

The waiting time W_i at a customer i is given as the maximum between 0 and difference between the opening of the TW 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 $r = (v_0, v_1, \ldots, v_R, v_{R+1})$ can be associated with a feasible (i.e., consistent with traveling times and TWs) arrival time vector $T_r = (T_{v_0}, T_{v_1}, \ldots, T_{v_R}, T_{v_{R+1}})$ 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 $x = \{r_1, \ldots, r_K\}$ on a graph G, given a time arrival vector for each route in the solution, i.e. $T_x = (T_{r_1}, \ldots, T_{r_K})$, is given by the following formula:

$$f_2(x, T_x) = \sum_{k=1}^{K} W_{r_k}(T_{r_k}) \tag{3}$$

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. Note that a solution is represented as a permutation of the customers, and it is evaluated with the split algorithm detailed in [12].

The VRPTW, where only the traveling cost is minimized, received many interests in the literature. Nowadays, all Solomon's instances (of size 100) can be optimally solved using an exact algorithm [11], however the computational cost grows exponentially with the size of the instances (e.g. it takes 64105 seconds to solve the instance R208). In practice meta-heuristic algorithms can obtain a

"good enough" solution in a short time and have the capacity to solve the large-scale complex problems, which is more suitable for applications. The NBD algorithm from Nagata et al. [10] is considered as a state of the art metaheuristic for the problem. Moreover, Schneider et al. [16] proposed different granular neighborhoods to improve an existing local search. Considering the multi-objective approaches the literature is more sparse. The second objective often minimized in the literature is the number of vehicles. Qi et al. [13] proposed a memetic algorithm based on MOEA/D to solve a bi-objective VRPTW. More recently, Moradi [9] integrated a learnable evolutionary model into a pareto evolutionary algorithm. The integration of learning mechanisms is known to be successful in both single-objective [1] and multi-objective algorithms [8]. In the following, we assume that the learning mechanism proposed is relevant for the studied problem, according to previous works [8].

3 Neighborhood Strategies

3.1 The Baseline MOEA/D

The MOEA/D [24] is a genetic algorithm that approximates the Pareto front by decomposing the multi-objective problem into several scalar objective subproblems, as illustrated in Figure 1. MOEA/D is a simple algorithm that has already been studied a lot in the literature [23], making it a good candidate for our study. More precisely, the objective function of the i-th subproblem is defined with a weight vector $w^i = (w_1^i, w_2^i)$, such that $w_1^i + w_2^i = 1$, and is expressed as: $g_i = w_1^i \cdot f_1 + w_2^i \cdot f_2$, with f_1 and f_2 being the two objectives defined in Section 2.2. In the following we consider a uniform distribution on the weight vectors, and we assume that is enough to obtain diverse subproblems. Moreover an external archive stores nondominated solutions found during the search. These solutions are returned once the termination criteria is reached.

However, we do not use the basic MOEA/D, but a variant where learning is integrated. We will refer to this algorithm as $\mathcal{A}$. This algorithm contains four major mechanisms. Two of them belong to the genetic aspect (crossover and mutation), while the two others belong to the learning aspect (injection and extraction).

The crossover is a Partially Mapped Crossover (PMX) [21], that occurs with probability p_{cro}. It is performed between two solutions taken from close subproblems. Among the two solutions produced only one solution is randomly selected to undergo the injection step, which is a costly step.

The mutation, replaced here by the LS, is applied following a probability p_{mut}. Three neighborhood operators are applied: Relocate, Swap and 2-opt*, generating the same neighborhood as described in [16]. The operators are shuffled before applying them, so that they are not always applied in the same order. Two possible strategies are considered to explore the neighborhoods and will be described in Section 3.2. To perform an efficient exploration of the neighbors, we use sequences as defined in [22]. Once a local optimum has been reached for an operator, the next one is applied and so on, until all have been applied.

In order to present the extraction and injection steps, we have to briefly present the integrated learning mechanism. We refer to [8] for a complete description of the mechanism. The learning mechanism uses learning groups, noted $\mathcal{G}_i$. The learning group $\mathcal{G}_i$ is associated to the subproblem with weight vector w^i. Each group gathers knowledge that is relevant for its associated subproblem.

The learning groups are updated when the extraction step occurs. However, to ensure that knowledge is extracted from local optima only, the extraction can occur only when the local search has been applied during the iteration. In addition to that, the extraction occurs with probability p_{ext}. The extraction step is quite similar to the one performed in PILS [1]. Given one solution $x = \{r_1, \ldots, r_K\}$, patterns can be extracted from routes $r_1, \ldots, r_K$. These patterns are sequences of consecutive customers (not including the depot). The patterns have a size between 2 and $MaxSize$, which is a parameter of the algorithm.

Finally the injection step, following a probability p_{inj}, uses the knowledge stored in the groups to diversify the solutions. More precisely, any solution that undergoes the injection step will receive at most $N_{Injected}$ patterns from one learning group randomly chosen. A pattern is kept only if it improves the solution. Each pattern is selected as follows. First the size of the pattern is randomly chosen, and then it is selected among the $N_{Frequent}$ most frequent patterns of the same size in the corresponding group. Figure 2 illustrates how the injection is performed. First the pattern is formed by deleting adjacent vertices, and then the pieces of route created are put together to form the best possible solution.

Algorithm 1 presents the framework of $\mathcal{A}$. Initially the external archive is empty as well as the learning groups. The initial population is randomly generated, and undergo the LS (still with its own probability). Then, until the termination criteria is reached, subproblems are solved one at a time. The crossover is the first operator applied, followed by the injection and the LS. The extraction is performed only if the LS occurred. Then neighboring subproblems have their solutions updated if necessary, as well as the archive.

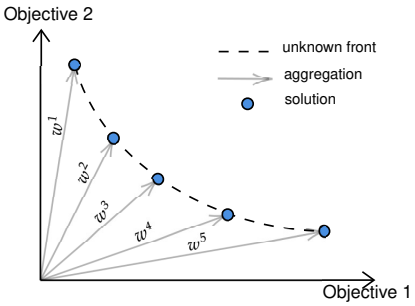

Fig. 1. A bi-objective problem decomposed into five scalar problems with MOEA/D.

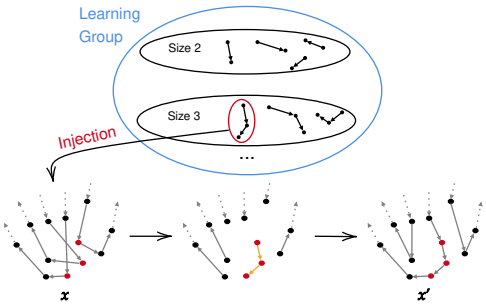

Fig. 2. Injection of a frequent pattern of size 3 in a VRPTW solution.

Algorithm 1: The $\mathcal{A}$ framework.

Input: M weight vectors $w^1, \ldots, w^M$ and the size m of each neighborhood.
Output: The external archive A

```
/* Initialisation                                                    */
```
$A \leftarrow \emptyset$
$P \leftarrow$ random initial population (x^i for the i-th subproblem)
for $i \in \{1, \ldots, M\}$ **do**
  $\mathcal{N}(i) \leftarrow$ indexes of the m closest weight vectors to w^i
  $x^i \leftarrow LS(x^i)$
  $Obj^i \leftarrow F(x^i)$
  $\mathcal{G}_i \leftarrow \emptyset$

```
/* Core of the algorithm                                             */
```
while *not stopping criteria satisfied* **do**
  for $i \in \{1, \ldots, M\}$ **do**
    $(k, l) \leftarrow$ select randomly two indexes from $\mathcal{N}(i)$
    $x_c \leftarrow Crossover(x^k, x^l)$
    $x_{inj} \leftarrow Injection(s, x_c)$
    $x' \leftarrow LS(x_{inj})$
    if *LS applied* **then**
      $\mathcal{K} \leftarrow Extraction(x')$
      $\mathcal{G}_1, \ldots, \mathcal{G}_M \leftarrow$ update with $\mathcal{K}$
    for $j \in \mathcal{N}(i)$ **do**
      if $g_j(x') \leq g_j(x^j)$ **then**
        $x^j \leftarrow x'$
        $Obj^j \leftarrow F(x')$
    $A \leftarrow Update(A, x')$
return A

3.2 Strategy of Exploration

In this section we give more details about the two exploration strategies considered in the local search. In routing problems, the most commonly used neighborhood exploration strategy is the classical *best* strategy, where the best move found by the operator is applied. That is why, we consider this strategy as the reference. Although this exploration allows a fast convergence towards a local optimum it requires an entire exploration of the neighborhoods before applying a single move, that is time consuming.

Here we propose a *first-best* strategy, which is inspired from [15]. This method is commonly used to solve flowshop problems. Algorithm 2 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 LS. For the given operator we try to insert each customer to its best location, considering the possible moves allowed by the operator. These moves are given through the

procedure *generate_moves* (l.7 of Algorithm 2). We repeat the process until no more improving moves are found for any customer.

The two strategies considered, *best* and *first-best*, lead to two variants of the algorithm $\mathcal{A}$, that are respectively $\mathcal{A}^{best}$ and $\mathcal{A}^{first-best}$

Algorithm 2: The $First - Best$ procedure.

```
Input: A solution x and a neighborhood operator 𝒩
Output: A local optimum
1  improve ← True
2  while improve do
3      improve ← False
4      indexes ← shuffle([1 ... N])
5      for customer ∈ indexes do
6          x' ← remove customer from x
7          moves ← generate_moves(customer, 𝒩)
8          x' ← best solution obtained by applying a move from moves
9          if g(x') < g(x) then
10             x ← x'
11             improve ← True
12 return x
```

3.3 Granularity and Pruning of Neighborhoods

In routing problems, many moves of a neighborhood operator can be a priori classified as irrelevant, and thus should not be considered during the neighborhood exploration. Most of the time these moves consider customers that are "far" distant. Having a method that restricts the neighborhood to relevant moves is interesting to spare time and resources during the LS. However, such a method requires a way to quantify the closeness between customers. In [18], the closeness between two customers is evaluated according to the distance between them. If it is enough for single-objective problems, it might not be adapted for multi-objective problems. In particular for our bi-objective VRPTW, close customers can incur a big waiting time if they are visited in the same route. Once a metric between customers is defined, a natural way to prune the neighborhood is to consider moves including the δ nearest customers for the metric defined.

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 both objectives. More precisely, each subproblem generated in MOEA/D, with weight vector $w = (w_1, w_2)$, has its own metric that is defined as: $d_2^w(u, v) = w_1 \cdot distance(u, v) + w_2 \cdot WT(u, v)$. The value $WT(u, v)$ is the waiting time incurred by going to v from u. If $[a_u, b_u]$

(resp. $[a_v,\ b_v]$) is the time window of customer u (resp. v), s_u the service time of customer u and t_{uv} the traveling time from u to v, then WT is expressed as follows: $WT(u,v) = max(0,\ a_v - (a_u + s_u + t_{uv}))$.

The strategies presented in this section lead to four variants of $\mathcal{A}$ following the exploration strategy and the distance metrics used by the neighborhood operators: $\mathcal{A}_{d_1}^{best}$, $\mathcal{A}_{d_1}^{first-best}$, $\mathcal{A}_{d_2}^{best}$, and $\mathcal{A}_{d_2}^{first-best}$.

4 Experimental Setup

4.1 The Solomon's Benchmark

We use the Solomon's instances [17] to evaluate the performance of the four variants presented in Section 3. 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 randomly places customers in the grid, while the generation C tends to create clusters of customers. The generation RC 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 $1XX$ have wider time windows than instances of class $2XX$, meaning that instances $2XX$ are more constrained. All 56 instances exist in three sizes: 25, 50 and 100. However, instances of size 25 and 50 are restrictions of instances of size 100. For our experiments instances of size 25 are discarded, since they are too small. Although this set was originally created to evaluate single-objective algorithms, it is used in the literature to evaluate the performance of multi-objective algorithms [6,13,9].

4.2 Setup and Tuning

We recall that the four variants compared are: $\mathcal{A}_{d_1}^{best}$, $\mathcal{A}_{d_1}^{first-best}$, $\mathcal{A}_{d_2}^{best}$ and $\mathcal{A}_{d_2}^{first-best}$. Note that the algorithm $\mathcal{A}_{d_1}^{best}$ will be our referent algorithm during the experiments, since it uses state of the art mechanisms.

Each algorithm is tuned to find a good setting of the parameters. To perform the tuning, we generated 96 new instances of sizes 50 and 100, by using the method described by Uchoa et al. [20] to mimic the Solomon's instances.

Each variant uses 10 parameters: M, the number of subproblems considered and m the size of the neighborhood of each subproblem. The four probabilities associated to each mechanism: p_{cro}, p_{inj}, p_{mut}, p_{ext}. The granularity parameter δ used to prune the neighborhood during LS. The maximal size $MaxSize$ of the patterns extracted, and the number $N_{Injected}$ of patterns injected, chosen among the $N_{Frequent}$ most frequent patterns. The parameters obtained after tuning are reported in Table 1.

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 [2].

	$\mathcal{A}_{d_1}^{best}$		$\mathcal{A}_{d_1}^{first-best}$		$\mathcal{A}_{d_2}^{best}$		$\mathcal{A}_{d_2}^{first-best}$	
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
δ	21	51	25	75	16	19	36	31
p_{cro}	0.94	0.30	0.88	0.86	0.93	0.35	0.94	0.67
p_{mut}	0.06	0.05	0.42	0.55	0.05	0.06	0.11	0.21
p_{ext}	0.50	0.96	0.48	0.60	0.55	0.90	0.86	0.83
$MaxSize$	2	3	5	5	2	4	2	5
p_{inj}	0.70	0.88	0.83	0.93	0.89	0.59	0.86	0.70
$N_{Frequent}$	73	165	74	135	52	175	66	115
$N_{Injected}$	33	80	17	74	10	63	18	31

Table 1. The configurations returned by irace for each variant and for both sizes of instance.

4.3 Experimental Protocol

In our experimentation, we investigate the efficiency of the mechanisms proposed, and their impact on the quality of the solutions returned.

To that aim, all the variants use a 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 the 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, since we do not know the true Pareto fronts of the instances. Note that, for the experiments we use the same values to normalize the objectives of the solutions returned by all variants. These values are simply the best and worst values for each objective, obtained among all the executions.

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 single-objective algorithms: the TS_{tw} from Schneider et al. [16] and NBD from Nagata et al. [10], but also to competitive multi-objective algorithms: the M-MOEAD from Qi et al. [13] and the MODLEM from Moradi [9].

5 Analysis of Neighborhood Strategies

The Table 5 regroups the average hypervolume obtained on all classes of instance for all the variants. One can see that two variants stand out from the others: $\mathcal{A}_{d_1}^{first-best}$ and $\mathcal{A}_{d_2}^{first-best}$. 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 $\mathcal{A}_{d_1}^{first-best}$ returns slightly higher hypervolumes

than $\mathcal{A}_{d_2}^{first-best}$ on most instances, and clearly outperforms $\mathcal{A}_{d_2}^{first-best}$ on few instances (e.g. RC1 of size 50 and C1 of size 100). Indeed, the subproblems which mainly focus on the waiting time will "forget" the distance between customers. That can worsen the hypervolume since we would like to obtain the minimal cost with the minimal possible waiting time in our Pareto front. Knowing that, the d_2 metric can be improved.

Now we analyze the gaps between the best solutions returned for the total cost objective and the best-knowns. The gaps obtained on instances of size 100 are reported in Table 3 (R instances), Table 4 (RC instances) and Table 5 (C instances). 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. One can notice that the variants $\mathcal{A}_{d_1}^{first-best}$ and $\mathcal{A}_{d_2}^{first-best}$ still outperform the two other variants. However, this is the variant $\mathcal{A}_{d_2}^{first-best}$ which returns the best results on most instances.

Class	Size	$\mathcal{A}_{d_1}^{best}$	$\mathcal{A}_{d_1}^{first-best}$	$\mathcal{A}_{d_2}^{best}$	$\mathcal{A}_{d_2}^{first-best}$
$R1$	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

Table 2. Average hypervolume obtained with the four variants on all classes of instance of both sizes.

6 Comparison with State of the Art Algorithms

Considering the results obtained in the former section, we decide to compare the variant $\mathcal{A}_{d_2}^{first-best}$ to the other state-of-the-art algorithms. Table 6 compares the average value of the best traveling cost obtained by different algorithms on each class of instance of size 100. We recall that, there are two single-objective algorithms: TS_{tw} [16] and NBD [10], and two multi-objective algorithms: M-MOEA/D [13] and MODLEM [9]. Note that MODLEM integrates a learning mechanism, which is a learnable evolution model based on decision trees. Moreover, the algorithms that solve the VRPTW in a single-objective context, first minimize the number of vehicles and then the traveled distance. To be fair,

Instance	Size	Reference	$\mathcal{A}_{d_1}^{best}$		$\mathcal{A}_{d_1}^{first-best}$		$\mathcal{A}_{d_2}^{best}$		$\mathcal{A}_{d_2}^{first-best}$	
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 3. Gaps (%) obtained for the total cost objective, relatively to the best-known on instances of class 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.

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 much higher than the one found by other algorithms. However, our algorithm is able to reach competitive results on C instances.

7 Conclusion

LS are commonly used in evolutionary algorithms to improve their performance. In this paper, we considered an LS from [16], adapted to the VRPTW, that uses a *best* strategy for exploration. That strategy has been compared to a *first-best* strategy inspired by ones used for other combinatorial problems like flow shops. Through our experiments, conducted on Solomon's instances, we showed that the adapted *first-best* strategy performs better than the *best* strategy, on the

Instance	Size	Reference	$\mathcal{A}_{d_1}^{best}$		$\mathcal{A}_{d_1}^{first-best}$		$\mathcal{A}_{d_2}^{best}$		$\mathcal{A}_{d_2}^{first-best}$	
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 4. Gaps (%) obtained for the total cost objective, relatively to the best-known on instances of class RC. 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	Reference	$\mathcal{A}_{d_1}^{best}$		$\mathcal{A}_{d_1}^{first-best}$		$\mathcal{A}_{d_2}^{best}$		$\mathcal{A}_{d_2}^{first-best}$	
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

Table 5. Gaps (%) obtained for the total cost objective, relatively to the best-known on instances of class C. 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.

Class	NBD [10]	TS_{tw} [16]	M-MOEA/D [13]	MODLEM [9]	$\mathcal{A}_{d_2}^{first-best}$
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)

Table 6. Comparison of the average of the best-traveling cost obtained on instances of size 100 between four state-of-the-art algorithms and our algorithm $\mathcal{A}_{d_2}^{first-best}$. The corresponding average number of vehicles used is given in brackets.

bVRPTW. We also investigated a new method for pruning the solution neighborhood taking into account the second criterion of our bVRPTW, being the waiting times. Our pruning method is able to reach similar results to the original one, but with smaller neighborhoods. The experimental results show also the benefit of our pruning method to reach better solutions when considering the first criterion only. The performance compared to state-of-the-art algorithms for both single- and bi-objective VRPTW shows the interest of our new neighborhood strategies. Future works will investigate the neighborhood exploration strategy for other variants of routing problems. Moreover, we will analyze the impact of the weights of our pruning method on the Pareto front.

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. 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.
3. 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.
4. Castro-Gutierrez, J., Landa-Silva, D., and Pérez, J. M. Nature of real-world multi-objective vehicle routing with evolutionary algorithms. In *2011 IEEE International Conference on Systems, Man, and Cybernetics* (2011), IEEE.
5. Coello, C. A. C., Dhaenens, C., and Jourdan, L. Multi-objective combinatorial optimization: Problematic and context. In *Advances in multi-objective nature inspired computing*. Springer, 2010, pp. 1–21.
6. 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.
7. Knowles, J. D. *Local-search and hybrid evolutionary algorithms for Pareto optimization*. PhD thesis, University of Reading Reading, 2002.
8. 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).

9. 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.
10. Nagata, Y., Bräysy, O., and Dullaert, W. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & operations research 37*, 4 (2010), 724–737.
11. 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.
12. Prins, C. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & operations research 31*, 12 (2004), 1985–2002.
13. 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.
14. Riquelme, N., Von Lücken, C., and Baran, B. Performance metrics in multi-objective optimization. In *2015 Latin American computing conference (CLEI)* (2015), IEEE, pp. 1–11.
15. Ruiz, R., and Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European journal of operational research 177*, 3 (2007), 2033–2049.
16. 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.
17. Solomon, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research 35*, 2 (1987), 254–265.
18. 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.
19. Toth, P., and Vigo, D. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.
20. 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.
21. Varun Kumar, S., and Panneerselvam, R. A study of crossover operators for genetic algorithms to solve vrp and its variants and new sinusoidal motion crossover operator. *International Journal of Computational Intelligence Research 13*, 7 (2017), 1717–1733.
22. Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research 40*, 1 (2013).
23. Xu, Q., Xu, Z., and Ma, T. A survey of multiobjective evolutionary algorithms based on decomposition: variants, challenges and future directions. *IEEE Access 8* (2020), 41588–41614.
24. Zhang, Q., and Li, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation 11*, 6 (2007).
25. Zhou, Y., and Wang, J. A local search-based multiobjective optimization algorithm for multiobjective vehicle routing problem with time windows. *IEEE Systems Journal 9*, 3 (2014), 1100–1113.
26. 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.