Adaptive local search for a pickup and delivery problem applied to large parcel distribution

Matthieu Fagot, Laure Brisoux Devendeville, Corinne Lucet

▶ **To cite this version:**

Matthieu Fagot, Laure Brisoux Devendeville, Corinne Lucet. Adaptive local search for a pickup and delivery problem applied to large parcel distribution. International Conference on Optimization and Learning (OLA2023), 2023, Malaga, Espagne, Spain. hal-04461557

HAL Id: hal-04461557
https://hal.science/hal-04461557

Submitted on 16 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Adaptative local search for a pickup and delivery problem applied to large parcel distribution*

Matthieu Fagot[1,2], Laure Brisoux Devendeville[1], and Corinne Lucet[1]

[1] Laboratoire MIS (UR 4290), Université de Picardie Jules Verne,
33 rue Saint-Leu 80039 Amiens Cedex 1, France
{m.fagot, laure.devendeville, corinne.lucet}@u-picardie.fr
[2] Smile Pickup, 80440 Boves, France
mfagot@smilepickup.com

1 Introduction

In 2020 alone, the number of parcels shipped in France has increased by 12.4% to reach 1.5 billion[3]. This phenomenon can be seen all over the world and has been reinforced by the recent sanitary crisis and successive lock downs. This growth has seen the explosion of a business: pickup points. More environmentally-friendly than home deliveries and often more practical, these pickup points are usually held by convenience stores close to customers. Although this works perfectly for small parcels, bigger ones such as furniture are usually not accepted either because of size, weight or space.

Smile Pickup is a young business which manages a group of pickup points dedicated to big parcels. With such an activity comes a logistical challenge for shipping parcels from stores to pickup points using local transport solutions. This sets up a vehicle routing problem with specific constraints that need to be taken into account. The objective is to give our partners customers the choice to be delivered in one of our pickup points accommodated for receiving oversized parcels. The store then packs the order for our fleet of vehicles to deliver to the pickup point chosen by the customer as soon as possible.

In this paper we will describe the particular vehicle routing problem faced by Smile Pickup which combines different well known vehicle routing problems: the *Smile Pickup Problem* (SPP). SPP is part of the class of paired vehicle routing problems with pick up and delivery [1]. As our problem is an extension of the classic VRP and PDP problem [2], SPP is also an NP-hard problem. The core of SPP is similar to the Pickup and Delivery Problem with Time Window (PDPTW) described by Li and Lim [3] with which they provide a benchmark. Regarding the time windows, Smile Pickup needs to be able to ensure great flexibility. For this purpose, we added the possibility of multiple time windows. This constraint was first proposed by DeJong et al. [4] to take into account customer brakes on home deliveries. More recently, Belhaiza [5] and Ferreira [6] have both proposed variable neighbourhood search heuristics to solve the vehicle

* CIFRE n^o 2021/0599 between Smile Pickup and MIS Laboratory

[3] https://www.data.gouv.fr/fr/datasets/observatoire-du-courrier-et-du-colis/

routing problem with multiple time windows. Additional constraints such as multiple depots and an heterogeneous fleet will also be considered in our problem as described in Salhi and al. [7]. If we look at pickup and delivery problems, we can refer to the multi-depot dial-a-ride problem with heterogeneous vehicles (M-DARP-HV) by Braekers [8] or more recently by Detti [9]. Dial a ride problems are close to PDP problems except that they transport people and not goods. Braekers [8] gives an exact method while Detti's article [9] gives a detailed integer linear program, a tabu search algorithm and multiple variable neighbourhood searches all tested on real life instances. The main difference between M-DARP-HV and our problem comes from the fact it takes into account the quality of the service delivered to the patients in the constraint deviates the problem from ours and makes the comparison difficult.

The main features of SPP which distinguish our problem from those encountered in the literature are: stores and pickup points can be loading places and unloading places for parcels at the same time - moreover, parcels can share their origins and destinations. This prevents us from using classical exploration methods of the solution space.

The remainder of this paper is organised as follows. Section 2 gives a detailed description of Smile Pickup's vehicle routing problem (SPP). Section 3 presents our $ALNS^{\epsilon}_{AS}$ algorithm. We detail movements and its two diversification process: an acceptance criterion based on simulated annealing and an epsilon greedy exploration strategy. Results follow in section 4 and conclusion in section 5.

2 Problem description

In this section, we give a detailed description of the problem faced by Smile Pickup. First, we will present the data of the problem. Afterwards, we will detail the representation of the solutions we chose and describe the objective. The example in Fig. 1 will be used through out this article to illustrate the problem.

2.1 Data

The problem faced by Smile Pickup spans over a total of $H \in \mathbb{N}$ consecutive days $J = \{1, ..., H\}$ during which parcels C need to be transported between places Ω using vehicles V.

Places. The set of places Ω is divided in three subsets: depots D where vehicles start and end their day, stores E and pickup points P which exchange parcels. A travel distance d_{ij} and duration m_{ij} is associated to each arc between places $(i, j) \in \Omega^2$. For day $\tau \in J$, a set of time windows $\{[e^{\tau}_{ik}, f^{\tau}_{ik}] | k \in [1, k_{\max}]\}$ is assigned to each place $i \in \Omega$. $k_{\max}$ is the maximum number of time windows per place. Unused time windows are set to $[0, 0]$. When a vehicle visits a place, it must load and unload its parcels during one of the associated time windows. The vehicle can arrive early at a place even though it will have to wait until a time window opens before starting loading and unloading. For depots, the time windows model the opening hours during which vehicle may depart and return.

places	0, 1	2	3	4	5	6
time windows	[6,22] [0,0]	[8, 10] [14, 16]	[8, 13] [0, 0]	[10, 16] [0, 0]	[8, 13] [0, 0]	[10, 12] [14, 18]
types	depot	store	store	pickup point	pickup point	pickup point

parcels	c_0	c_1	c_2	c_3	c_4	c_5
s_r	1	1	1	1	1	1
o_r	2	3	2	3	2	6
d_r	4	5	5	6	6	3
f_r^c (min)	5	7	1	1	5	7
f_r^d (min)	5	4	1	1	5	4
dav_r	1	1	1	1	1	1

vehicles	Pu_v	K_v	o_v	τ
v_0	800	2	0	0
v_1	800	2	1	1

Legend for places: ▲ Depots ■ Stores ● Pickup Points

Fig. 1: Example of data

.

Parcels. A parcel $c_r \in C$ of length s_r is made available in a store or a pickup point $o_r \in E \cup P$ and needs to be delivered to it's destination $d_r \in E \cup P$. A parcel can be pickup up and delivered from day $dav_r \in J$ but can also be stored in o_r and serviced on a later day $\tau \geq dav_r$ for a penalty cost p_r^τ. Solutions do not need to deliver all parcels but each undelivered parcel will cost P_{NL}. Furthermore, the time needed to load (resp. unload) a parcel c_r is f_r^c (resp. f_r^d).

Vehicles. For each day $\tau \in J$, a set of vehicles V_τ is available ($V = \bigcup_{\tau \in J} V_\tau$). Each vehicle starts at a depot d_v and returns at the same depot at the end of the day. Vehicle $v \in V$ is given a usage cost Pu_v and a capacity K_v. The sum of the length of the parcels in a vehicle v must not exceed K_v at any time during the tour.

In Fig. 1, an example of data is given. The instance spans on a single day $J = 1$ using 2 vehicles $V = V_1 = \{v_0, v_1\}$, 6 parcels $C = \{c_0, c_1, ..., c_5\}$ and 7 places including depots $D = \{0, 1\}$, stores $E = \{2, 3\}$ and pickup points $P = \{4, 5, 6\}$. As for example, parcel c_5 of length $s_5 = 1$ needs to be picked up at point 6 and delivered at store 3. Loading will take $f_5^c = 7$ minutes while unloading only takes $f_5^d = 4$. If vehicle v_0 is used, it starts from depot 0 and cost $Pu_0 = 800$. The vehicle would need to load at point 6 either between time slots 10 and 12 or 14 and 18. Unloading in 3 must take place between 8 and 13.

2.2 Solution representation

A solution S is represented by a set of tours. A tour is assigned a single vehicle. We will consider a vehicle to be equivalent to a tour and use the same notation v. A tour $v \in V$ is an ordered list of triplets $< t_0^v, t_1^v, ..., t_k^v >$. Triplet i of tour v is such that $t_i^v = (p_i^v, C_i^{v,+}, C_i^{v,-})$ with $p_i^v \in \Omega$, $C_i^{v,+} \subseteq C$ the set of parcels loaded at p_i^v by v and $C_i^{v,-} \subseteq C$ the set of parcels unloaded at p_i^v by v. During servicing of a place, unloading will always be performed before loading. We notice that $\forall v \in S,\ p_0^v \in D,\ p_k^v \in D,\ p_0^v = p_k^v,\ C_0^{v,+} = C_0^{v,-} = \emptyset$ and $C_k^{v,+} = C_k^{v,-} = \emptyset$.

We also introduce the following notations: the set of undelivered parcels C^u, the set of delivered parcels C^d, V^+ the set of used vehicles and the function

$d : C \rightarrow J$ indicating the day $d(c_r)$ parcel $c_r \in C^d$ is delivered. For simplicity, we also introduce $dist(v)$ the distance travelled by vehicle $v \in V^+$.

Furthermore, a solution S is feasible if capacity constraints are respected and if for every tour, there exists at least one schedule for the associated vehicle to be able to respect the time windows of every place visited by the tour.

Using this notation we can represent the solution in Fig. 2 as follows:

$$v_0 =< (0,\emptyset,\emptyset),(2,\{c_4\},\emptyset),(6,\emptyset,\{c_4\}),(0,\emptyset,\emptyset) >$$
$$v_1 =< (1,\emptyset,\emptyset),(3,\{c_1\},\emptyset),(5,\emptyset,\{c_1\}),(1,\emptyset,\emptyset) >$$

In the representation in Fig. 2, the sets $C_i^{v,+}$ and $C_i^{v,-}$ are listed under place p_i. If we look at Fig. 3 presenting a solution containing all parcels, the first vehicle loads c_5 when visiting pickup point 6 for the first time. It then goes to store 3 where c_5 is unloaded and c_3 is loaded. After having picked up parcel c_4 in store 3, both c_4 and c_3 are unloaded when visiting pickup point 6 once again. The vehicle finishes his journey by coming back to depot 0.

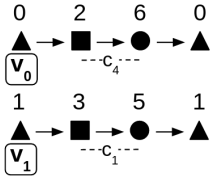

Fig. 2: An initial solution.

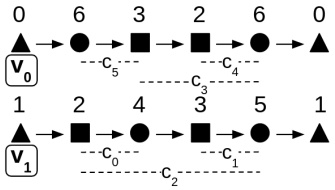

Fig. 3: An optimal solution.

2.3 Solution evaluation

The objective of the problem is to find the solution S which minimises the following criteria: the sum of the vehicles used cost, the number of undelivered parcels, the total distance travelled and the storage penalties. To normalise and prioritise the different criteria, we assign the respective weights $\alpha_{\text{veh}}, \alpha_{\text{nl}}, \alpha_{\text{dist}}, \alpha_{\text{pen}}$ to them. The fitness function is shown in equation 1.

$$f(S) = \alpha_{\text{veh}} \sum_{v \in V^+} PU_v + \alpha_{\text{nl}} \left|C^u\right| + \alpha_{\text{dist}} \sum_{v \in V^+} \text{dist}(v) + \alpha_{\text{pen}} \sum_{c_r \in C^d} p_r^{d(c_r)} \quad (1)$$

3 An Adaptative Large Neighborhood Search: $\text{ALNS}^{\epsilon}_{\text{SA}}$

In this section, we are going to describe the different steps in building the local search algorithm we propose and named $ALNS^{\epsilon}_{SA}$, to solve Smile Pickup problem (SPP). The principle of such a method consists in exploring the solution space by generating a set of neighbours from the current solution S and choosing one of them as the new current solution. Neighbourhoods are generated by movements dedicated to the specific problem considered. The process is

iterated and the best solution S_B found is returned. After testing different local search algorithms as the Variable Neighbourhood Search (VNS) and the Large Neighbourhood Search (LNS), we selected the Adaptative Large Neighbourhood Search (ALNS) [10], an extension of the LNS in which algorithms learn how to move in the search space more efficiently. The method we use, iterates over a deterioration phase and a reconstruction phase. These neighbours are generated from specific movements of two different types: deteriorating movements and constructive movements which we will describe in section 3.2.

3.1 General ALNS algorithm

Adaptative Large Neighbourhood Search algorithm dictates the general strategy used to decide which neighbouring solution to move to at each iteration. A general scheme is presented in algorithm 1. The algorithm starts by generating an initial solution using a greedy algorithm before iterating over the following four steps until the time limit is exceeded.

Algorithm 1 general scheme of ALNS algorithm

```
1: Nb_iterations ← 0
2: S ← greedy()
3: while stopping criteria do
4:     Nb_iterations++
5:     S_P ← S
6:     N_s ← choose a deteriorating movement using distribution W_S
7:     Apply N_s to S_P
8:     while stopping insertion criteria do
9:         N_s ← choose a constructive movement using distribution W_I
10:        Apply N_s to S_P
11:    end while
12:    Update current solution S according to S_P.
13:    Rewards (π_S^m, π_I^m) and counts (θ_S^i, θ_I^i) movements used in iteration.
14:    if Nb_iterations mod Δ = 0 then
15:        Update movement weights W_S^m and W_I^m according to rewards and movement counts.
16:    end if
17: end while
```

The first step is the deterioration phase (lines 5 to 7). One of the three deteriorating movements (see 3.2) is chosen and applied to the current solution S_P. Each deteriorating movement m is given a weight W_S^m used to choose one of them.

The second step is the constructive phase (lines 8 to 11) which rebuilds the solution by inserting unassigned parcels by performing constructive movements (see 3.2). These movements are selected and applied using their own weights W_I^m. While deteriorating movements are applied once, constructive movements

are applied iteratively. The process stops when E_{max} successive movements fail to produce a feasible solution.

In the deteriorating phase as in the constructive phase, two different strategies are tested. The first one is the classical roulette wheel selection which is performed following the distribution W_S (resp. W_I) to choose the movement to apply. The second one is the epsilon greedy strategy detailed in section 3.4.

The next step decides if this newly built solution is worthy enough to become the new current solution for the next iteration (line 12). It also classifies the performance of the iteration based on the solution produced for the weight adjustment step. This step is detailed on section 3.3.

The final step (lines 13 to 18) adapts the weights as follows: each movement applied successfully is rewarded based on the classification given in step 3. π_S^m (resp. π_I^m) counts the rewards earned by the deteriorating (resp. constructive) movement m while θ_S^m (resp. θ_I^m) counts the number of times it was successfully applied. Each Δ iterations, the weight distributions are corrected using the following formula: $W_S^m \leftarrow \rho \frac{\pi_S^m}{\theta_S^m P_S} + (1-\rho) W_S^m$ with P_S the sum of $\frac{\pi_S^m}{\theta_S^m}$ for the set of deteriorating movements. The weights W_I^m are updated in the same way as W_S^m.

3.2 Movements

In the following section, the six movements created to move from neighbour to neighbour in the solution space will be detailed.

Parcel insertion movement: *pim*

This movement *pim* first randomly selects a parcel $r \in C^u$ and tries to insert it in a tour. Candidate tours are classified in four sets ξ_j, $j = 1, \ldots, 4$. The first one, ξ_1 contains tours that visit both origin o_r and destination d_r of c_r in the right order. Next, ξ_2 contains tours visiting origin o_r and ξ_3 contains those that visit only the destination d_r. Finally ξ_4 contains tours that visit neither o_r nor d_r. Tours where d_r precedes o_r are included in ξ_2.

Let ξ_k be the first non-empty set, then operator *pim* will randomly pick out in ξ_k a tour v. Depending on ξ_k, the following processes are applied to v:

- case $\xi_k = \xi_1$. Choose a triplet $t_i^v \in v$ and $t_j^v \in v$ such that $p_i^v = o_r$, $p_j^v = d_r$ and $i < j$. Add c_r to $C_i^{v,+}$ and to $C_j^{v,-}$.
- case $\xi_k = \xi_2$. Choose a triplet $t_i^v \in v$ such that $p_i^v = o_r$ and add c_r to $C_i^{v,+}$. Choose t_j^v such that $j \geq i$. Insert the triplet $(d_r, \emptyset, \{r\})$ in tour v between t_j^v and t_{j+1}^v.
- case $\xi_k = \xi_3$. Choose a triplet $t_j^v \in v$ such that $p_j^v = d_r$ and add c_r to $C_j^{v,-}$. Choose t_i^v, such that $i < j$. Insert the triplet $(o_r, \{c_r\}, \emptyset)$ in tour v between t_i^v and t_{i+1}^v.
- case $\xi_k = \xi_4$. Choose a triplet t_i^v. Insert the triplet $(o_r, \{c_r\}, \emptyset)$ in tour v between t_i^v and t_{i+1}^v. Choose a second triplet t_j^v, $i \leq j$. Insert the triplet $(d_r, \emptyset, \{c_r\})$ in tour v between t_j^v and t_{j+1}^v.

When c_r is added to $C_i^{v,+}$, if capacity constraint is violated, then movement *pim* is reject. In a same manner, when triplets $t_i^v = (o_r, \{c_r\}, \emptyset)$ and/or $t_j^v = (d_r, \emptyset, \{c_r\})$ are inserted to tour v, if window constraints are violated, movement is rejected.

Consider now the initial solution in Fig. 2 and $c_0 \in C^u$ the parcel to insert ($o_0 = 2$ and $d_0 = 4$). Then the four sets are: $\xi_1 = \emptyset$, $\xi_2 = \{v_1\}$, $\xi_3 = \emptyset$ and $\xi_4 = \{v_2\}$. We add c_0 to $C_2^{v_1,+}$ and insert the triplet $t_4^{v_1} = (d_{c_0}, \emptyset, \{c_0\})$ in tour v_1. The resulting tour is illustrated in Fig. 4(b).

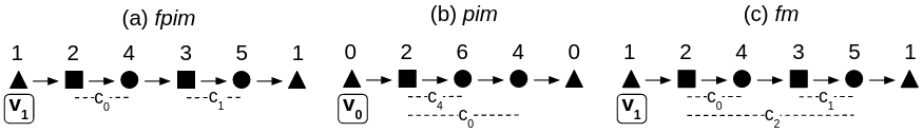

Fig. 4: Examples of solutions after the application of constructive movements.

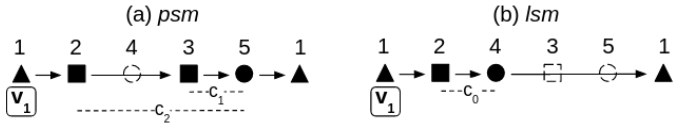

Fig. 5: Examples of solutions after the application of deteriorating movements.

Forced parcel insertion movement: *fpim*
Movement *fpim* starts by randomly selecting a parcel $c_r \in C^u$ and a tour $v \in V$. Then two triplets t_i^v and t_j^v ($0 \leq i \leq j < k$) are randomly selected. Next, triplets $(o_r, \{c_r\}, \emptyset)$ and $(d_r, \emptyset, \{c_r\})$ are respectively inserted after t_i^v and t_j^v in the right order. Capacity and time window constraints are then checked on v. If unsuccessful, *fpim* is rejected. Fig. 4(a) presents a possible outcome of the application of *fpim* on the solution of Fig. 2. c_0 is added to v_1 by inserting both $(2, \{c_0\}, \emptyset)$ and $(4, \emptyset, \{c_0\})$ after $t_0^{v_1}$ in the right order.

Fill movement: *fm*
Movement *fm* randomly selects a vehicle $v =< t_0^v, ..., t_k^v >$ and one of its triplets $t_i^v = (p_i^v, C_i^{v,+}, C_i^{v,-})$ with $0 < i < k$. For every triplet $t_j^v = (p_j^v, C_j^{v,+}, C_j^{v,-})$ such that $0 < j < i$, let's define $C_{fm}^u = \{c_r \in C^u \mid o_r = p_j^v \text{ and } d_r = p_i^v\}$. Parcels of C_{fm}^u are added in a random order to both $C_j^{v,+}$ and $C_i^{v,-}$ as long as capacity and time window constraints are not broken. Ditto with triplets t_j^v where $i < j < k$, parcels of $C_{fm}^u = \{c_r \in C^u \mid o_r = p_i^v \text{ and } d_r = p_j^v\}$ are inserted in $C_i^{v,+}$ and $C_j^{v,-}$. Fig. 4(c) gives an example with the application of *fm* on the solution in Fig. 4(a) where $C^u = \{c_2, c_3, c_5\}$. v_1 is selected as well as $t_1^{v_1}$. The only triplet $t_j^{v_1}$ ($1 < j \leq 4$) where $C_{fm}^u \neq \emptyset$ is $t_4^{v_1}$: $C_{fm}^u = \{c_2\}$. Hence, c_2 is added in $C_1^{v_1,+}$ and $C_4^{v_1,-}$. No more parcel can be added because the tour is full with capacity $K_{v_1} = 2$.

Parcel suppression movement: *psm*
Movement *psm* starts by randomly selecting a parcel $c_r \in C^d$. Let be $t_i^v =$

$(p_i^v, C_i^{v,+}, C_i^{v,-})$ and $t_j^v = (p_j^v, C_j^{v,+}, C_j^{v,-})$ such that $c_r \in C_i^{v,+}$ and $c_r \in C_j^{v,-}$. Movement *psm* modifies both triplets by removing c_r from $C_i^{v,+}$ and $C_j^{v,-}$. c_r is added to C^u. *psm* finishes by removing unused places ($C_i^{v,-} = C_i^{v,+} = \emptyset$) and unused tours. Fig. 5(a) shows the result of the application of *psm* on parcel c_0 of solution in Fig. 4(c). c_0 is added to C^u and removed from $t_1^{v_1}$ and $t_2^{v_1}$. Because $C_2^{v_1,-} = C_2^{v_1,+}$ are empty, $t_2^{v_1}$ is removed from v_1.

Place suppression movement: *lsm*
Movement *lsm* starts by randomly selecting a tour $v \in V^+$ and one of its triplet $t_i^v = (p_i^v, C_i^{v,+}, C_i^{v,-})$. All parcels from $C_i^{v,+}$ and $C_i^{v,-}$ are removed from tour v. *psm* finishes by removing unused places ($C_i^{v,-} = C_i^{v,+} = \emptyset$) and unused tours. Fig. 5(b) shows the outcome of applying *lsm* to tour v_1 of the solution in Fig. 4(c). $t_4^{v_1} = (5, \emptyset, \{c_1, c_2\})$ is removed and $\{c_1, c_2\}$ are added to C^u. Triplet $t_3^{v_1}$ is now empty and hence removed from v.

Tour suppression movement: *tsm*
This movement is the most disruptive movement. A tour $v \in V^+$ is chosen at random. All parcels serviced by v are added to C^u while v is removed.

A short term tabu memory was added to avoid cycling over a set of solutions. Parcels are set tabu when they are removed from a tour. Afterwards, the parcel can not be re-inserted in the same tour for a certain number of iterations called tenure δ.

3.3 Solution updating and classification

After a neighbour S_P of S has been chosen by the first two steps of an iteration, we update the current solution S. Four cases are possible: (1) $f(S_P) < f(S_B)$ where S_B is the best solution visited so far, (2) $f(S_P) < f(S)$, (3) S_P satisfies the acceptance criterion and (4) S_P does not satisfy the acceptance criterion. When cases (1), (2) or (3) occur, S is replaced by S_P. Furthermore, when case (1) is met, S_B is updated with S_P. These cases are also used to choose the reward σ_i ($i \in \{1, \ldots, 4\}$) according to the quality of the solution in order to update the weights of the movements W_S and W_I [10].

The acceptance criterion is used to accept some solutions (case (3)) which have decreased the fitness function and hence avoid getting stuck in local optima. Simulated annealing was chosen to manage the acceptance criterion (see section 3.4).

3.4 Exploitation, exploration and learning strategies

In this subsection, strategies to improve the performance of the ALNS are discussed including a simulated annealing strategy to accept degraded solutions and an epsilon greedy movement selection strategy.

Simulated annealing (SA). Simulated annealing is used as an acceptance criterion to allow degradation of the solutions. Indeed, in case (3) (see section 3.3) the solution S_P is accepted with probability $p(S_P) = e^{-\frac{f(S_P)-f(S)}{T}}$ where T is the

temperature and S the current solution. The temperature controls the range of solutions to be accepted with high probability. When T is high, worse solutions have higher chance of passing while when T is lower, only solutions with close fitness scores have a real chance of going through.

We tested two scenarios. The first one is the classical SA process where the temperature decreases progressively using a multiplicative coefficient γ_{SA}. The second one proceeds with restarts when the solution is not improved for $R^{\text{SA}}_{\text{step}}$ iterations. In this case the weight learning is conserved.

Epsilon greedy. The epsilon greedy strategy is used to balance between exploration and exploitation during movement selection. This new strategy takes in consideration the weight distributions W_I and W_S. The movement with the heavier weight is chosen with probability $(1 - \epsilon)$ while with probability ϵ the roulette wheel is used with weight distributions W_I and W_S to select a movement. During the execution of the algorithm, ϵ is slowly decreased by a constant multiplicative coefficient γ_ϵ and reinitialised if, for $R^{\epsilon}_{\text{step}}$ iterations, there is no improvement since the last restart.

The version of our algorithm, including SA with restart and epsilon-greedy strategy is named $ALNS^{\epsilon}_{SA}$.

4 Computational experiments

In this section we will describe the experiments we made to test our $ALNS^{\epsilon}_{SA}$ algorithm. First we will describe the instances used before talking about the method we used to tune our parameters. A comparison of VNS, LNS and ALNS, and for each of them their combination with the SA acceptance criterion is presented. Then two movement selection strategies were put in competition: roulette wheel and epsilon greedy. Finally, we tested our $\text{ALNS}^{\epsilon}_{\text{SA}}$ on Li & Lim benchmark [3].

4.1 Instances

Our algorithms were tested, tuned and compared on two different sets of instances: Li & Lim benchmark instances[4] and our own instances. Li and Lim's instances are dedicated to the PDPTW. This benchmark was the closest we found to our problem. A simplification of our problem is necessary to be able to compare. The horizon is set to $H = 1$ and places have exactly one time window. Parcels do not share their place of origin and destination meaning the number of stores and pickup points equal the number of parcels. Finally, we adjust the weights α_i of the criteria in the fitness function in equation 1 in order to deliver all parcels and prioritise the number of vehicles used before minimising the travelled distance.

We also generated a benchmark PickOptBench to fully test our problem. These 135 instances where generated to be as close as possible to the reality

[4] Benchmark available on `http://www.sintef.no/pdptw`

faced by Smile Pickup. This was achieved by analysing the distribution of parcels across working days J and places Ω. Time windows and vehicles data where chosen based on existing ones. Here are the main characteristics for our set of instances: $H = 3$, $|D| \in \{1, 2\}$, $|E| \in \{1, 2, 4\}$, $|P| \in \{5, 10, 20, 40\}$, $k_{\max} = 3$, $|C| \in \{60, 120, 240, 480, 960\}$, $|V| \in [2, 46]$ and $K_v \in [20, 30]$ (see section 2).

The experimentations were conducted on an *intel core i7-10875H* for a maximum execution time of 10 minutes each.

4.2 Parameter tuning

To tune our algorithms and the different strategies implemented, we used the Irace software by López-Ibáñez and al. [11]. The tuning was done using instances from PickOptBench with 1000 experiments per run. We proceeded step by step and started by tuning the ALNS with simulated annealing without restart. After having fixed those parameters, we added other strategies one by one and tuned them separately. The results are presented in table 1 with coefficient $\alpha_{\text{veh}} = 1000$, $\alpha_{\text{nl}} = 500$, $\alpha_{\text{dist}} = 1$, $\alpha_{\text{pen}} = 1$ for the fitness function (see section 1).

algorithm	**parameters tuned**	**configuration**
ALNS SA	$< \rho, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \gamma_{\text{SA}}, \delta >$	$< 3, 29, 10, 1, 1, 895, 7428 >$
restart	$< T_{\text{restart}}, R^{\text{sa}}_{\text{step}} >$	$< 6164, 9598 >$
epsilon greedy	$< \gamma_\epsilon, R^{\epsilon}_{\text{step}} >$	$< 4.5 \cdot 10^5, 10^5 >$

Table 1: Best configurations after tuning with irace.

4.3 ALNS and Simulated Annealing contributions

We first compare the following algorithms: greedy, VNS, VNS_{SA}, LNS, LNS_{SA}, ALNS and ALNS_{SA} on the PickOptBench and Li & Lim Benchmark. VNS_{SA} (resp. LNS_{SA}, ALNS_{SA}) is the VNS (resp. LNS, ALNS) algorithms improved with a simulated annealing acceptance criterion. Each instance was ran 5 times on each algorithm. Maximum execution time is 3 minutes.

PickOptBench							
algorithms	Greedy	VNS	VNS_{SA}	LNS	LNS_{SA}	ALNS	ALNS_{SA}
best	1739	1236	1041	1138	1027	1081	**1011**
average all	1959	1300	1077	1156	1056	1151	**1027**
worst	2230	1363	1107	1172	1090	1230	**1047**
Li & Lim Benchmark							
algorithms	Greedy	VNS	VNS_{SA}	LNS	LNS_{SA}	ALNS	ALNS_{SA}
best	1651	1017	1083	1692	1017	1622	**865**
average all	1760	1068	1124	1728	1061	1654	**889**
worst	1877	1118	1165	1764	1104	1686	**911**

Table 2: Algorithm comparison on both benchmarks.

Table 2 gives the average fitness score over all runs and the average of the best (resp. worst) scores obtained over the 5 runs. We can deduce multiple informations from results in Table 2: first, as expected, all the local search algorithms improve over the greedy algorithm by at least 50%. We also notice simulated annealing improves the results for LNS, VNS and ALNS with a gain of 10% up to 20% on average.

4.4 Combining epsilon greedy and simulated annealing restart strategies

Here we measure the impact of the strategy used in the choice of a movement by comparing roulette wheel to epsilon greedy. We also evaluate the interest of integrating a restart of the SA, when updating the current solution. Experimentation results presented in Table 3 were obtained on PickOptBench 35 biggest instances. The smaller ones are not discriminating enough since every strategies find the same results. 10 runs of 10 minutes were made for each instance. The column *best* (resp. *worst*) presents the average over all instances of the fitness (see equation 1) of the best (resp. worst) solution returned during the 10 runs of each instance. Likewise, the column *average* is the average of fitness of all the solutions.

We clearly see the benefit of combining in $ALNS^{\epsilon}_{SA}$ epsilon greedy strategy with restart for SA.

Algorithms		roulette wheel			epsilon greedy		
		best	average	worst	best	average	worst
SA	no restart	2565	2602	2637	2578	2604	2630
	restart	2582	2607	2636	**2554**	**2591**	**2613**

Table 3: Comparison of the different strategies on PickOptBench.

4.5 Comparison on Li & Lim benchmark

The comparison of our algorithm $ALNS^{\epsilon}_{SA}$ with the best known results found for Li & Lim benchmark [3] are presented in this section. This benchmark is composed of 56 instances organised in six classes LC1, LC2, LR1, LR2 LRC1 and LRC2, for wich results are a pair of values: the number of used vehicles and the total distance travelled. $ALNS^{\epsilon}_{SA}$ finds the best known solutions on more than 70% for classes LC1, LC2, LR1 and LRC1. Nevertheless, only 10% are reached for classes LR2 and LRC2. Table 4 illustrates a small part of these results, for classes LC1 and LRC2.

LC1	best known		$\text{ALNS}^{\epsilon}_{\text{SA}}$		LRC2	best known		$\text{ALNS}^{\epsilon}_{\text{SA}}$	
	$\lvert V^+\rvert$	distance	$\lvert V^+\rvert$	distance		$\lvert V^+\rvert$	distance	$\lvert V^+\rvert$	distance
lc101	10	824.94	10	824.94	lrc201	4	1406.94	5	1497.47
lc102	10	824.94	10	824.94	lrc202	3	1374.27	5	1544.84
lc103	9	1035.35	10	826.44	lrc203	3	1089.07	4	1092.13
lc104	9	860.01	9	860.01	**lrc204**	3	818.66	3	818.66
lc105	10	824.94	10	824.94	lrc205	3	1302.2	5	1363.63
lc106	10	824.94	10	824.94	lrc206	3	1159.03	4	1210.00
lc107	10	824.94	10	824.94	lrc207	3	1062.05	4	1138.55
lc108	10	826.44	10	826.44	lrc208	3	852.76	4	937.57

Table 4: Comparison with best solutions on classes LC1 and LRC2.

5 Conclusion

This paper presented our $ALNS^{\epsilon}_{SA}$ algorithm for a pickup and delivery problem applied to a real life case for Smile Pickup business (SPP). Additional constraints considered are multiple time windows, heterogeneous fleet and multiple depots. $ALNS^{\epsilon}_{SA}$ is an Adaptive Learning Neighbourhood Search algorithm, combining two diversification processes. The first one is based on a simulated annealing technique dedicated to updating the current solution. The second one is an epsilon greedy strategy used to balance between exploration and exploitation during the generation of neighbourhoods. $ALNS^{\epsilon}_{SA}$ was tested on PickOptBench and Li&Lim benchmarks. Experimentation results show that such an approach is very promising for solving SPP. In addition, many levers exist to improve the performance of $ALNS^{\epsilon}_{SA}$. For example, we plan to improve suppression movements by integrating more relevant selection criteria than the random selection of deleted items.

References

1. Parragh, S.N., Doerner, K.F., Hartl, R.F.: A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations. Journal für Betriebswirtschaft **58** (2006) 81–117
2. Berbeglia, G., Cordeau, J.F., Gribkovskaia, I., Laporte, G.: Static pickup and delivery problems: a classification scheme and survey. Top **15** (2007) 1–31
3. Li, H., Lim, A.: A metaheuristic for the pickup and delivery problem with time windows. In: Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001. (2001) 160–167
4. de Jong, C., Kant, G., Van Vlient, A.: On finding minimal route duration in the vehicle routing problem with multiple time windows. Manuscript, Department of Computer Science, Utrecht University, Holland (1996)
5. Belhaiza, S., Hansen, P., Laporte, G.: A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. Comput. Oper. Res. **52** (2014) 269–281
6. Ferreira, H.S., Bogue, E.T., Noronha, T.F., Belhaiza, S., Prins, C.: Variable neighborhood search for vehicle routing problem with multiple time windows. Electron. Notes Discret. Math. **66** (2018) 207–214
7. Salhi, S., Imran, A., Wassan, N.A.: The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. Comput. Oper. Res. **52** (2014) 315–325
8. Braekers, K., Caris, A., Janssens, G.K.: Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transportation Research Part B: Methodological **67** (2014) 166–186
9. Detti, P., Papalini, F., de Lara, G.Z.M.: A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. Omega **70** (2017) 1–14
10. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40** (2006) 455–472
11. 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** (2016) 43–58