A fast re-optimization approach for dynamic vehicle routing

Victor Pillac, Christelle Guéret, Andrés Medaglia

A fast re-optimization approach for dynamic vehicle routing

Victor Pillac[1,2], Christelle Guéret*[,1], and Andrés L. Medaglia[2]

[1]*LUNAM Université, École des Mines de Nantes, IRCCyN UMR 6597, Nantes, France*
[2]*Universidad de Los Andes, COPA & CEIBA, Bogotá, Colombia*

October 9, 2012

Technical Report 12/6/AUTO
École des Mines de Nantes
France

Abstract

The present work deals with dynamic vehicle routing problems in which new customers appear during the design or execution of the routing. We propose a parallel Adaptive Large Neighborhood Search (pALNS) that produces high quality routes in a limited computational time. Then, we introduce the notion of driver inconvenience and define a bi-objective optimization problem that minimizes the cost of routing while maintaining its consistency throughout the day. We consider a problem setting in which vehicles have an initial routing plan at the beginning of the day, that is periodically updated by a decision maker. We introduce a measure of the driver inconvenience resulting from each update and propose a bi-objective approach based on pALNS that is able to produce a set of non-dominated solutions in reasonable computational time. These solutions offer different tradeoffs between cost efficiency and consistency, and can be used by the decision maker to update the routing of the vehicles introducing a controlled number of changes.

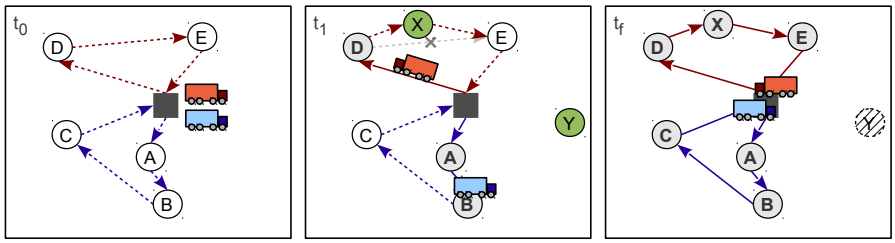
Keywords: Dynamic vehicle routing, route consistency, bi-objective optimization

*Corresponding author: gueret@mines-nantes.fr

1 Introduction

The problem of operating a fleet of vehicles arises in many contexts, from pickup and delivery of goods to the transportation of patients in hospitals. More specifically, Vehicle Routing Problems (VRP) deal with the design of a set of minimal-cost vehicle routes that serve the demand for goods or services of a group of geographically spread customers, satisfying operational constraints. From an information perspective, such problems generally include two dimensions: *evolution* and *quality* of information [30]. Information evolution relates to the fact that the data available to the planner may change during the execution of the routes, for example with the arrival of new customer requests. Information quality reflects possible uncertainty on the available data, for instance, when the demand of a customer is only known as a range estimate of its real demand, or when the geographical distribution of customers can be forecasted. Based on these dimensions, Pillac et al. [24] classify vehicle routing problems in four categories depending on whether the problem is static/dynamic and deterministic/stochastic. Dynamism in routing can emerge from different aspects of the problem. The most common source of dynamism is the arrival of new customers with a demand for goods or services. Other sources of dynamic include dynamically revealed demands for a set of known customers, dynamic travel times, and vehicle availability.

The present work focuses on *dynamic and deterministic* routing, also referred to as *online* routing, in which part or all of the input is unknown and revealed dynamically and unpredictably during the execution of the routes. Vehicle routes are redefined in an ongoing fashion, requiring technological support for real time communication between the vehicles and the decision maker (e.g., mobile phones and global positioning systems). More specifically, we study the Dynamic Vehicle Routing Problem with Time Windows (D-VRPTW), in which a limited fleet of identical capacitated vehicles must deliver a product to a set of customers over a single day horizon. Each customer has a geographic position, requires a known quantity of product, and must be served within a given time frame. While a set of (*static*) customers is known beforehand, new (*dynamic*) customers may appear during the day.



a. Initial routing b. New customers c. Final routing

Figure 1: Illustration of a typical dynamic vehicle routing problem.

Figure 1 illustrates the routing of two vehicles in a dynamic context. Before the vehicles leave the depot (1a.), two routes are designed to visit the currently known customers: (A, B, C) and (D, E). While the vehicles execute their route, two new customers (X and Y) appear at time t_1 (1b.). At this stage, the dispatcher must decide whether or not it should *accept* or *reject* the new requests. In this case, Y is far from the current routes and vehicles, therefore its service may not be feasible

or may be too costly. Customer Y is thus rejected and a penalty is paid. On the other hand, X is accepted and inserted in the second route. Finally, at time t_f the executed route are (A, B, C) and (D, X, E) (1c.). This example reveals how dynamic routing inherently adjusts routes in an ongoing fashion, which requires real-time communication between vehicles and the dispatching center. In this context, the problem is first to design an initial set of routes, visiting all the static customers. Then, each time a new customer appears, the problem is to decide whether it can be served or not, and eventually, to reoptimize the vehicle routes. We assume that by rejecting customers we incur a penalty that can be interpreted as an outsourcing cost.

Dynamic routing problems introduce new challenges as they require to react quickly to changes in the available data. According to Ichoua et al. [17], the level of dynamism of a problem can be characterized according to two dimensions: the *frequency* of changes and the *urgency* of customer requests. The former is the rate at which new information becomes available, while the latter is the period of time between the disclosure of a new customer and its expected service time. From this observation different metrics have been proposed to measure the dynamism of a problem (or instance). Lund et al. [22] defined the *degree of dynamism* δ as the ratio between the number of dynamic customers n_d and the total number of customers n_{tot}: $\delta = \frac{n_d}{n_{tot}}$. Larsen [20] extended the degree of dynamism to take into account the disclosure date and the time windows of the dynamic customers.

To the best of our knowledge, the first application of an optimization technique to dynamic routing is due to Psaraftis [30] with the development of a dynamic programming approach. His research focuses on the Dial A Ride Problem (DARP) and consists in finding the optimal route each time a new customer is known. The main drawback of dynamic programming is the well-known *curse of dimensionality* [28, Chap. 1], which often prevents its application to large instances. Few research was conducted on dynamic routing between Psaraftis [30] and the late 1990s. However, the last decade has seen a renewed interest in dynamic routing, with numerous approaches tackling a variety of problems. This section classifies the major contributions in this field in two categories: 1) *periodic reoptimization* and 2) *continuous reoptimization*. The reader is referred to the reviews, books, and special issues by Gendreau and Potvin [11], Ghiani et al. [12], Ichoua et al. [17], Larsen et al. [21], Pillac et al. [24], and Zeimpekis et al. [40], to complement our review.

Figure 1 presents an overview of periodic reoptimization approaches: the algorithm starts at the beginning of the day and a first optimization produces an initial solution S_0. Then, the procedure waits for an update in the available data, or for a fixed period of time, followed by a new optimization trigger that leads to an updated solution S_{t+1}. The advantage of periodic reoptimization approaches is that they can be based on algorithms developed for static routing, for which extensive research has been conducted. Their main drawback is that all the optimization has to be performed before updating the solution, which can increase the delays for the dispatcher, while the computational power is unused during waiting times.

Periodic reoptimization approaches were used in different contexts to tackle dynamic routing problems. Chen and Xu [7] designed a dynamic column generation algorithm (DYCOL) for the D-VRPTW. The authors use the concept of *decision epochs* over the planning horizon, which are the dates when the optimization process runs. It is worth noting that a new customer is not handled until the next decision epoch, hence, the optimization is run statically and independently at each decision epoch. The main advantage of this time partition is that similar computational effort is allowed for each time slice. The novelty of their approach relies on dynamically gen-

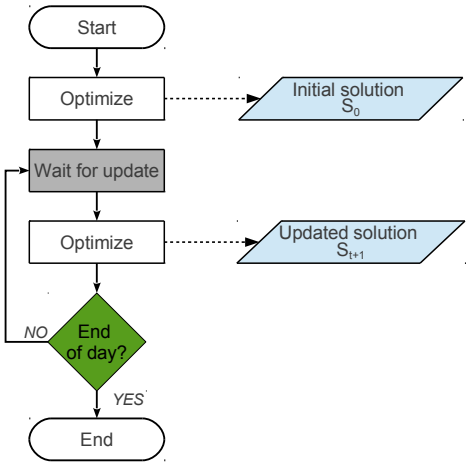


Figure 2: Overview of periodic reoptimization approaches

erating columns for a set-partitioning model, using columns from the previous decision epoch. The authors compared DYCOL to a traditional column generation with no time limit (COL). Computational results based on the Solomon benchmark [35] demonstrate that DYCOL yields comparable results in terms of objective function, but with running times limited to 10 seconds, opposed to the various hours consumed by COL. Using a notion similar to decision epochs, Montemanni et al. [23] developed an Ant Colony System (ACS) to solve the D-VRP. An interesting feature of their approach is the use of the pheromone trace to transfer characteristics of a good solution to the next time slice. ACS was also used by Gambardella et al. [9] and Rizzoli et al. [31]. Other heuristic approaches, such as Tabu Search (TS), were also used to tackle the Dynamic Pickup and Delivery Problem (D-PDP) [2, 6] and the Dynamic Dial-a-Ride Problem (D-DARP) [1, 3].

In contrast, continuous reoptimization approaches perform the optimization throughout the day in an *optimization loop* and store information on good solutions in an *adaptive memory* (see Figure 1). In parallel, a *decision loop* aggregates the information from the memory whenever needed. The advantage of such approaches is that the computational power utilization is maximized, at the price of possibly cumbersome implementation.

To the best of our knowledge, the first application of continuous reoptimization is due to Gendreau et al. [10]. Their approach consists in the adaptation of the Tabu Search (TS) framework introduced by Taillard et al. [36] to a dynamic context motivated by the local operation of long distance express courier services, which can be seen as a D-VRPTW. The general idea is to maintain a pool of good routes–the *adaptive memory* [37]–which is used to generate initial solutions for a parallel tabu search. The parallelized search is done by partitioning the routes of the current solution and optimizing them in independent threads. Whenever a new customer request arrives, it is checked against all the solutions from the adaptive memory to decide whether it should be accepted or rejected. This framework was also implemented for the D-VRP [15, 16]. Bent and Van Hentenryck [4] generalized this framework and introduced the Multiple Plan Approach (MPA) to tackle the D-VRPTW. The general idea is to populate and maintain a solution pool (the routing *plans*) that are used to

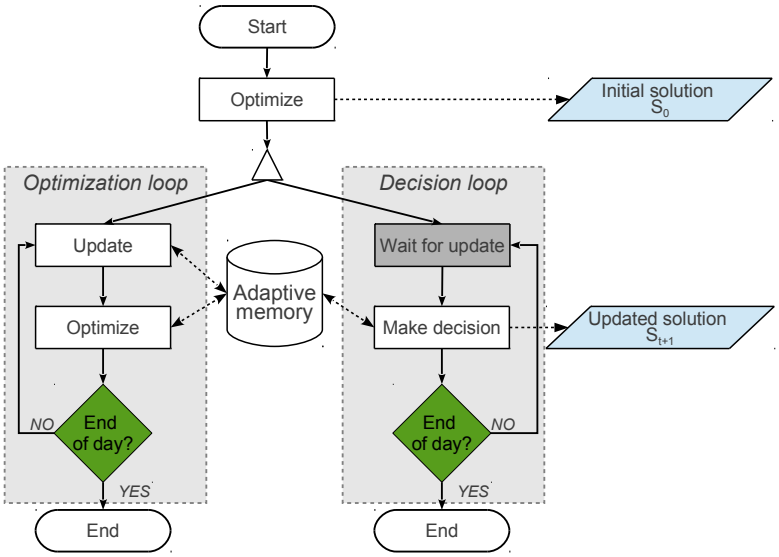


Figure 3: Overview of continuous reoptimization approaches

generate a *distinguished solution*. Whenever a new customer arrives, a procedure is called to check whether it can be served or not; if it can be served, then the customer is inserted in the solution pool and incompatible solutions are discarded. Pool updates are performed periodically or whenever a vehicle finishes servicing a customer. This pool-update phase is crucial and ensures that all solutions are coherent with the current state of vehicles and customers. The pool can be seen as an adaptive memory that maintains a set of alternative solutions. Following a different approach, Benyahia and Potvin [5] studied the D-PDP and proposed a Genetic Algorithm (GA) that models the decision process of a human dispatcher. More recently, other GAs were also used for the same problem [8, 13] and for the D-VRP [38]. Genetic algorithms in dynamic contexts are very similar to those designed for static problems, except that they run throughout the planning horizon and solutions are constantly adapting to the changes made to the input.

In this work we propose two parallelized periodic reoptimization approaches. Section 2 presents a parallel adaptive large neighborhood search to tackle the D-VRPTW; Section 3 introduces a bi-objective extension of the D-VRPTW and proposes a reoptimization approach; finally, Section 4 concludes this work and gives directions for further research.

2 Fast reoptimization for dynamic routing

The proposed approach is based on a parallel Adaptive Large Neighborhood Search (pALNS) algorithm which is used to compute an initial solution, and then, to reoptimize the solution whenever a new customer request arrives. In the remainder of this section we present the original Adaptive Large Neighborhood Search (ALNS) algorithm, discuss the proposed parallelization scheme and the reoptimization approach, and present computational results on the D-VRPTW.

2.1 The Adaptive Large Neighborhood Search

The ALNS algorithm, originally proposed by Pisinger and Ropke [25], is an extension of the Large Neighborhood Search (LNS) algorithm [34]. LNS works by successively destroying (removing customers) and repairing (inserting customers back) a current solution, using *destroy* and *repair operators*. ALNS adds an adaptive layer that randomly selects operators depending on their past performance, automatically fitting the algorithm to the instance at hand. We refer the interested reader to Pisinger and Ropke [26] for a detailed description of LNS, ALNS, and related methods.

Algorithm 1 Adaptive Large Neighborhood Search (ALNS) algorithm

Input: $\mathbf{\Pi}_0$ initial solution, z evaluation function, Θ^-/Θ^+ set of destroy/repair operators, I number of iterations
Output: $\mathbf{\Pi}^*$ the best solution found

1: $\mathbf{\Pi}^* \leftarrow \mathbf{\Pi}_0$ ▷ Initialize best solution
2: $\mathbf{\Pi} \leftarrow \mathbf{\Pi}_0$ ▷ Initialize current solution
3: **for** I iterations **do**
4: $d \leftarrow \texttt{select}(\Theta^-); r \leftarrow \texttt{select}(\Theta^+)$ ▷ Select destroy/repair
5: $\mathbf{\Pi}' \leftarrow \texttt{r}(\texttt{d}(\mathbf{\Pi}))$ ▷ Generate a neighbor
6: **if** $\texttt{accept}(\mathbf{\Pi}', \mathbf{\Pi})$ **then** ▷ $\mathbf{\Pi}'$ is accepted as current solution
7: $\mathbf{\Pi} \leftarrow \mathbf{\Pi}'$ ▷ Update current solution
8: **end if**
9: **if** $z(\mathbf{\Pi}') < z(\mathbf{\Pi}^*)$ **then** ▷ An improvement has been found
10: $\mathbf{\Pi}^* \leftarrow \mathbf{\Pi}'$ ▷ Update best solution
11: **end if**
12: $\texttt{updateScore}(d, r, \mathbf{\Pi}')$ ▷ Update scores
13: **end for**
14: **return** $\mathbf{\Pi}^*$

Algorithm 1 presents the outline of the ALNS approach. ALNS starts with an initial solution $\mathbf{\Pi}_0$. Then for I iterations, the algorithm selects destroy and repair operators (line 4) with a roulette wheel that reflects their past performance. Destroy operators remove a subset of customers from the current solution, while repair operators reinsert them using heuristics that are known to perform well on the problem at hand (line 5). The resulting new solution is conditionally accepted as current solution according to a simulated annealing criterion (line 6). At the end of each iteration, the scores of the destroy and repair operators are updated depending on the solution they generated (line 12).

2.2 Parallel Adaptive Large Neighborhood Search

We propose pALNS, an extension of the Adaptive Large Neighborhood Search (ALNS) algorithm that includes a novel parallelization scheme that efficiently spreads the computational effort among independent processors.

Algorithm 2 presents the outline of pALNS. The algorithm maintains a pool $\mathcal{P}$ of N promising solutions that are optimized in K subprocesses (note that $N \geq K$). For each *master* iteration, a subset of K promising solutions is selected randomly (line 2) and distributed among independent subprocesses. Each subprocess performs I^p ALNS iterations (lines 3-14) by destroying and repairing the current solution $\mathbf{\Pi}^p$ as in the original ALNS algorithm. The final current solution of each subprocess is

Algorithm 2 Parallel Adaptive Large Neighborhood Search (pALNS) algorithm

Input: $\mathcal{P}$ initial solutions, z evaluation function, Θ^-/Θ^+ set of destroy/repair operators, N maximum size of the solution pool, K number of subprocesses, I^m number of master iterations, I^p number of iterations performed in parallel.
Output: $\mathbf{\Pi}^*$, the best solution found

1: **for** I^m iterations **do**
2: $\quad\mathcal{P}' \leftarrow \texttt{selectSubset}\,(\mathcal{P}, K)$ ▷ Select a subset of K solutions
3: $\quad$**parallel forall** $\mathbf{\Pi}$ **in** $\mathcal{P}'$ **do**
4: $\quad\quad\mathbf{\Pi}^p \leftarrow \mathbf{\Pi}$ ▷ Current solution for this subprocess
5: $\quad\quad$**for** I^p iterations **do**
6: $\quad\quad\quad d \leftarrow \texttt{select}\,(\Theta^-)\,; r \leftarrow \texttt{select}\,(\Theta^+)$ ▷ Select destroy/repair
7: $\quad\quad\quad\mathbf{\Pi}' \leftarrow \texttt{r}\,(\texttt{d}\,(\mathbf{\Pi}^p))$ ▷ Destroy and repair current solution
8: $\quad\quad\quad$**if** $\texttt{accept}\,(\mathbf{\Pi}', \mathbf{\Pi}^p)$ **then**
9: $\quad\quad\quad\quad\mathbf{\Pi}^p \leftarrow \mathbf{\Pi}'$ ▷ $\mathbf{\Pi}'$ is accepted as current solution
10: $\quad\quad\quad$**end if**
11: $\quad\quad\quad\texttt{updateScore}\,(d, r, \mathbf{\Pi}')$ ▷ Update d and r scores
12: $\quad\quad$**end for**
13: $\quad\quad\mathcal{P} \leftarrow \mathcal{P} \cup \{\mathbf{\Pi}^p\}$ ▷ Add $\mathbf{\Pi}^p$ to the pool $\mathcal{P}$
14: $\quad$**end forall**
15: $\quad\mathcal{P} \leftarrow \texttt{retain}\,(\mathcal{P}, N)$ ▷ Retain at most N solutions in the pool $\mathcal{P}$
16: **end for**
17: **return** $\mathbf{\Pi}^* = \arg\min_{\mathbf{\Pi}\in\mathcal{P}}\{z(\mathbf{\Pi})\}$

added to the pool of promising solutions (line 13) and a filtering procedure ensures that the pool contains at most N solutions, including the best solution found so far (line 15). The algorithm stops after I^m master iterations, which corresponds to $I = I^m \times I^p$ ALNS iterations. Note that the implementation of pALNS ensures that no synchronization is required between subprocesses to avoid deadlocks. The following paragraphs present in more detail the different components of the algorithm.

2.2.1 Destroy

Destroy operators remove a random fraction $\xi \in [\xi_{min}, \xi_{max}]$ of the customers from the current solution. We denote $\mathcal{R}$ the set of customers served in the solution, and $\mathcal{U}$ the set of customers that are not served. We used three destroy operators originally proposed by Pisinger and Ropke [25]: *random*, *related*, and *critical*.

The random destroy operator selects customers randomly and removes them from their actual tours.

The related destroy attempts to remove customers that share some characteristics. Let the *relatedness* r_{ij} of customers i and j be a measure of how related two customers are (the lower the r_{ij}, the more related i and j). The procedure starts by randomly removing a *seed* customer i ($\mathcal{U} = \{i\}$), then it iteratively selects a customer $i \in \mathcal{U}$, and removes the most related customer j^*:

$$j^* = \arg\min_{j\in\mathcal{R}}\{r_{ij}\} \tag{1}$$

There are different ways to measure the relatedness. We propose a new metric that can be precalculated, namely a-priori relatedness, that does not depend on the actual

position of customers in tours:

$$r_{ij}^{s} = \left(1 + \frac{c_{ij}}{M_c}\right)^{\theta_c} \left(1 + \frac{|b_i - b_j|}{M_t}\right)^{\theta_t} \tag{2}$$

Where c_{ij} is the distance between i and j, b_i and b_j are the end of the time windows of customers i and j, M_c and M_t are scaling constants, θ_c and θ_t define the weight given to the geographic distance between the two customers, and the difference between due dates respectively.
On the other hand, time-oriented relatedness [25] measures the difference between the current times of service A_i and A_j of customers i and j:

$$r_{ij}^{t} = |A_i - A_j| \tag{3}$$

Finally, critical destroy consists in removing the customer i^* such that the cost of the resulting solution is minimal:

$$i^* = \arg\max_{i \in \mathcal{R}} \{c_{i-1,i+1} - c_{i-1,i} - c_{i,i+1}\} \tag{4}$$

Where $i-1$ and $i+1$ are the predecessor and successor of i.

In practice related and critical operators are randomized and the $\lfloor y^p |\mathcal{R}| \rfloor$-th best customer is selected, where y is a random number in $[0, 1)$ and $p \geq 1$ is a parameter that controls the level of randomness (the lower the p, the more randomness is introduced).

2.2.2 Repair

Repair operators attempt to insert customers that are currently unserved. Our implementation is based on *regret-q heuristics* [27]: at each iteration the algorithm inserts (at the best position) the customer with the lowest *regret* value. The regret-q value r_i^q of customer i is a measure of how desirable it is to insert i in the current iteration assuming that the best insertion will no longer be feasible in the next iteration. It is defined as:

$$r_i^q = \sum_{h=2}^{q} \left(\Delta z_i^h - \Delta z_i^1\right) \tag{5}$$

Where Δz_i^q is the cost of the q-th best insertion of customer $i \in \mathcal{U}$. Note that ties are resolved by selecting the customer with the lowest Δz_i^1 value, and therefore regret-1 corresponds to the classical best insertion heuristic. We used three regret levels: regret-1, regret-2, and regret-3.

2.2.3 Adaptive layer

At each iteration, the pALNS algorithm selects a destroy and a repair operator using a selection roulette, such that operator $\theta \in \Theta^{\bullet}$ is selected with probability w_θ, where $\Theta^{\bullet}$ is either the set of destroy (Θ^-) or repair (Θ^+) operators. Probabilities are initialized with value $\frac{1}{|\Theta^{\bullet}|}$, and then updated every l iterations (a *segment*) as follows:

$$w_\theta \leftarrow (1-\rho) w_\theta + \rho \frac{s_\theta}{\sum_{\theta \in \Theta^{\bullet}} s_\theta} \tag{6}$$

Where $\rho \in [0, 1]$ is the *reaction factor* which defines how quickly probabilities are adjusted, and s_θ is the *score* of operator θ in the last l iterations. The scores s_θ are

reset to 0 every l iterations and updated at the end of each iteration depending on the new solution: a score of σ_1 is granted for a new best solution, σ_2 for an improving solution, σ_3 for a non-improving but accepted solution, and σ_4 for a rejected solution. It is worth noting that in contrast with the adaptive scheme originally proposed by Pisinger and Ropke [25], this formula ensures that $\sum_{\theta\in\Theta}. w_\theta = 1$ at all time, which makes it easier to interpret the relative weight of each component.

2.2.4 Objective function

The initial solution or the solution resulting from the destroy operator can leave some customers unserved ($\mathcal{U} \neq \emptyset$). Therefore we need to be able to evaluate a partial solution $\mathbf{\Pi}'$ to account for the unserved customers. Given an evaluation function z and an initial solution $\mathbf{\Pi}_0$, Pisinger and Ropke [25] define the cost of partial solution $\mathbf{\Pi}'$ as follows:

$$z_\phi(\mathbf{\Pi}') = z(\mathbf{\Pi}') + \phi|\mathcal{U}|z(\mathbf{\Pi}_0) \tag{7}$$

Where ϕ is a parameter that controls the unserved customer penalty.

2.2.5 Acceptance criterion

As in the original ALNS, the pALNS algorithm relies on a simulated annealing acceptance criterion which accepts a new solution $\mathbf{\Pi}'$ with probability $e^{\frac{z(\mathbf{\Pi})-z(\mathbf{\Pi}')}{T}}$, where T is the *temperature* parameter. The temperature is initialized with the value T_0 and it is reduced at each iteration by a *cooling factor* c. The two parameters T_0 and c are set depending on the initial solution and the target number of iterations [32]. Given an initial solution $\mathbf{\Pi}_0$, T_0 is defined such that a solution with value $(1+w)z(\mathbf{\Pi}_0)$ is accepted with probability p, and c is set such that the temperature after n iterations is equal to αT_0.

2.2.6 Computation of an initial solution

The pALNS algorithm requires an initial solution which is computed with a regret-3 constructive heuristic: starting with empty routes for each vehicle, the algorithm iteratively inserts the customer with the lowest regret value as described in §2.2.2.

2.2.7 Solution pool

The solution pool acts as a shared memory and allows subprocesses to collaborate efficiently. In the original algorithm, the simulated annealing acceptance criterion results in a search scheme that starts from a diversification phase, in which poor solutions may be accepted as current solutions, and progressively switch to an intensification phase, in which only improving solutions are accepted. The use of a solution pool that would contains the N best solutions found so far tend to break this scheme, as poor solutions may never be kept in the pool and will therefore not be exploited properly. To overcome this limitation we propose to maintain a pool of *diverse* solutions that are promising in terms of cost.

This is achieved by the `retain` method (line 15) which ensures that $\mathcal{P}$ contains at most N solutions: if $|\mathcal{P}| > N$ then the method retains the N best solutions according to the fitness function f:

$$f(\mathbf{\Pi}) = (1-\lambda)\mathrm{rank}_z(\mathbf{\Pi}) + \lambda\mathrm{rank}_d(\mathbf{\Pi}) \tag{8}$$

Where λ is a weight between 0 and 1, $\mathrm{rank}_z(\mathbf{\Pi})$ is the rank of solution $\mathbf{\Pi}$ according to its objective value, and $\mathrm{rank}_d(\mathbf{\Pi})$ is the rank of $\mathbf{\Pi}$ according to its average *broken-pairs distance* [29] relative to the other solutions from $\mathcal{P}$. The broken pairs distance counts the number of arcs that differ between two solutions. This fitness function is inspired by the *biased fitness* introduced by Vidal et al. [39] in a genetic algorithm with diversity management. The weight λ can either be fixed a-priori, or adjusted throughout the search to switch from diversification ($\lambda = 1$) to intensification ($\lambda = 0$). Note that we ensure that $\mathcal{P}$ always contains the best solution found so far.

2.3 Parallel reoptimization approach for the D-VRPTW

Figure 4 illustrates the proposed reoptimization approach: the algorithm starts by producing an initial solution S_0 by using a constructive heuristic coupled with the pALNS described in the previous section. Then each time a new customer appears, it fixes the currently executed portion of the routes, and re-runs the pALNS for a limited number of iterations to produce an updated solution S'_t. If pALNS is able to insert the new customer request, then the customer is *accepted* and S'_t becomes the new current solution, otherwise the customer is *rejected* and S_t remains as the current solution.

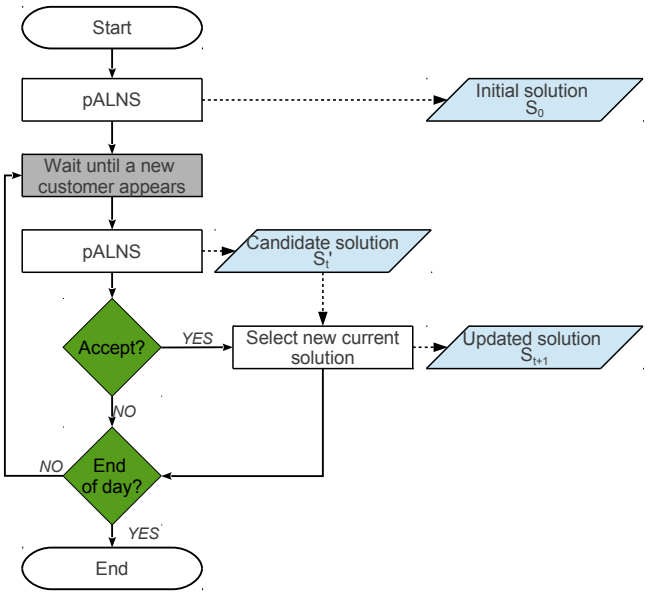


Figure 4: Overview of the proposed approach

It is important to note that the immediate commitment of idle vehicles to customers may lead to difficulties when new customers appear. Figure 5 illustrates this with a single vehicle. Suppose that at time t a vehicle is assigned to a customer i, if the vehicle is dispatched immediately to i (upper left time line), it will travel to i then wait at its destination until the start of the time window (black brackets). On the other hand, if a waiting strategy is used (lower left time line), the vehicle will remain idle until the latest moment such that it will not wait at i. If at time $t + 1$ a new customer j appears, in the first case j cannot be served as the vehicle is already waiting at i, while in the second case a visit to j can be inserted right before i. As a

consequence, vehicles are considered to remain idle at their current location until the latest departure time such that it will not wait at the next customer, leaving time for further insertions.

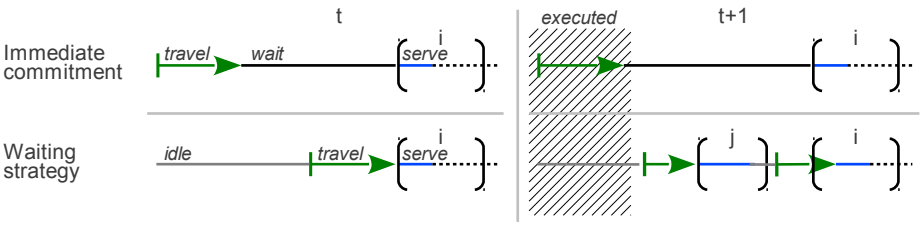


Figure 5: Illustration of the waiting strategy.

2.4 Computational results

To assess the effect of parallelization we tested our algorithm on the static instances for the VRPTW proposed by Solomon [35] on a quad-core desktop computer[1]. For the detailed parameter setting of the algorithm please refer to Appendix A.

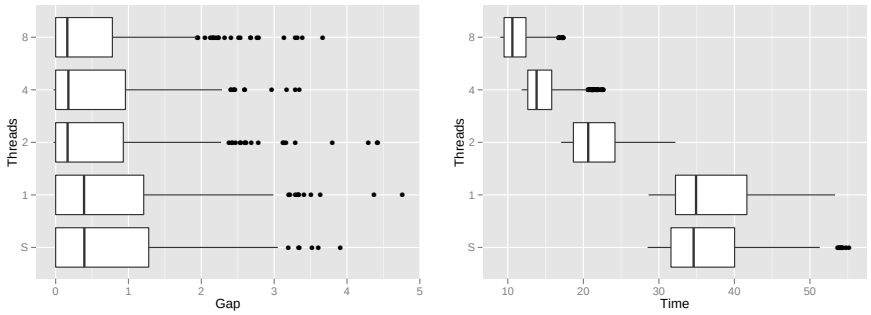
		Parallel - Num. of Threads							
	Seq.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Gap	0.74%	0.72%	0.55%	0.69%	0.54%	0.70%	0.52%	0.66%	0.48%
Gap (st. dev.)	0.87%	0.88%	0.76%	0.89%	0.70%	0.86%	0.74%	0.82%	0.66%
Time (s)	36.58	37.32	22.07	17.60	14.70	14.69	13.39	12.37	11.32
Time (s, st. dev.)	6.27	6.33	4.06	3.17	2.72	2.57	2.50	2.27	2.15

Table 1: Comparison of gap to the best known solutions and running times for different levels of parallelization.

Table 1 presents aggregated values over the 53 instances, with ten run per instance and 25,000 ALNS iterations[2]. The first column corresponds to the original sequential (Seq.) implementation of the ALNS, and the following to the parallel implementation with 1 to 8 threads. The first and second rows contain the mean and standard deviation of the gap value relative to either the optimal or the best known solution. Finally, the third and fourth rows show the mean and standard deviation of the CPU times. Note that increasing the number of threads has a limited impact on the gap to the best known solutions, which is consistently around 0.6%, but it allows a reduction of running times by a factor 3.3. Figure 6 presents the box plot of the distribution of the gap and CPU times for the sequential (S) and parallel implementations with 1, 2, 4, and 8 threads. A graphical analysis shows that the median gap and variance slightly decrease with the number of threads. In contrast, the median running time and variance decreases sharply with the number of threads. Therefore, we selected the configuration with 8 threads as it offers the best compromise between

[1]CPU: Intel i7 860 (4x2.8GHz), RAM: 6GB DDR3, OS: Ubuntu 11.10 x64, Java 7

[2]To ensure that $I = I^m \times I^p \times K \simeq 25000$, we used $I^m = \left\lceil \frac{25000}{40 \times K} \right\rceil$



a. Gap (%) vs. number of threads

b. CPU time (s) vs. number of threads

Figure 6: Impact of the number of threads on the gap and CPU time.

speed and quality. Note that the processor used is a quad-core with Intel hyper-threading technology which allows two threads per core. This partially explains the relatively small reduction of CPU times when switching from 4 to 8 threads.

We tested the pALNS algorithm on the instances proposed by Lackner [19] and based on the Solomon [35] benchmark, in which a fraction of the customers is revealed dynamically. The instances contain 100 customers located randomly (R), in clusters (C), or combining both (RC); while the planning horizon is either short (type 1) or long (type 2); and the number of dynamic customers (or degree of dynamism, δ) is either 10, 30, 50, 70, or 90. These instances are organized combining location, horizon length, and degree of dynamism. We consider the minimization of the traveled distance. For each instance, we performed 10 simulations in which pALNS is initially run for 25,000 iterations to produce an initial solution. Then, each time a new customer appears, pALNS is run for 5,000 iterations to produce a solution that will be used until the next customer is revealed. Finally, pALNS is run for 50,000 iterations to solve the a-posteriori problem, in which all the accepted customers are assumed to be known beforehand.

Table 2 presents the *Value of Information* (VI) [22] for each instance group and degree of dynamism (δ). The value of information for instance I is defined as the ratio $\frac{z(I)-z(I_{\text{off}})}{z(I_{\text{off}})}$ where $z(I)$ is the value of the solution found by the algorithm for the dynamic instance, and $z(I_{\text{off}})$ is the value of the solution for a-posteriori instance I_{off}. As expected, results indicate that the VI increases with the degree of dynamism, which can be explained by the fact that suboptimal routing decisions add up over time, and more decisions are made in highly dynamic instances. However, even when 90 out of 100 customers appear dynamically, the VI is of just 11% on average, which means that the algorithm is still able to produce a final routing that is very close to what would have been done if all the customers were known from the beginning of the day.

Table 3 presents a comparison of approaches for the Lackner [19] instances. The first and second columns present the traveled distance and number of rejected customers for pALNS, averaged over 10 runs and for each group and degree of dynamism. The third and fourth columns report the average distance, relative average additional distance (in parenthesis), and number of rejected customers for the Large Neighborhood Search (LNS) approach proposed by Hong [14], while the fifth and sixth columns report the same values for the Genetic Algorithm (GA) developed by

δ	R1	C1	RC1	R2	C2	RC2	Avg.
10	2.05%	2.89%	3.06%	1.70%	1.66%	1.61%	**2.14%**
30	4.67%	5.83%	5.83%	4.34%	1.74%	4.70%	**4.54%**
50	6.41%	9.28%	9.03%	8.15%	2.82%	5.38%	**6.93%**
70	8.29%	11.18%	10.24%	10.17%	5.41%	8.60%	**9.03%**
90	9.33%	12.49%	11.84%	11.83%	6.51%	12.33%	**10.71%**

Table 2: Average value of information for the Lackner [19] instances

Lackner [19]. Note that the experimental setting of the two cited studies is not explicitly presented, which limits the relevance of direct comparisons. Nonetheless, figures show that our approach is competitive both in terms of traveled distance and number of rejected customers. In addition, average running times are of just 5.3s for the initial optimization, and 2.0s for subsequent reoptimizations, which is significantly less than the 33s and 47s reported by Hong [14] and Lackner [19] respectively.

3 Route consistency in dynamic routing: a bi-objective approach

Most studies on dynamic routing consider that routes are designed online, which means that vehicle drivers do not know their next destination until they finish serving their current customer. Although this assumption is theoretically appealing and allows a better optimization of the cost function, it may not be desirable if drivers are used to know their routes from the beginning of the day. In practice, having a set of routes known a-priori that are then changed may be desirable over purely dynamic routing. Hence there is a need for approaches able to maintain consistency in the vehicles routes throughout the day while ensuring cost efficiency.

To the best of our knowledge, all studies on dynamic routing focus on the optimization of a single criterion, such as the minimization of the total traveled distance or the maximization of the number of served customers. On the other hand, and as surveyed by Jozefowiez et al. [18], a growing number of studies on static routing consider multiple objectives in an attempt to better fit operational contexts. In this section we present a preliminary study that takes into account driver inconvenience. The proposed approach is an adaptation of the pALNS algorithm that simultaneously minimizes a cost function and maximizes the route consistency throughout the day.

3.1 Measuring consistency

Assuming that an initial set of routes are handed to the drivers at the beginning of the day, it seems natural to consider them as the *reference* routes for each driver. To prevent multiple and unnecessary changes in routes, we assume that drivers will only be informed of changes in their routes at the last possible moment. As a consequence, a change will take effect only when necessary. From the driver's perspective, four types of changes can be made to the route: one or more customers may be a) inserted between existing customers; b) removed; c) swapped within the same route; d) substituted by a customer previously unvisited. In this context, minimizing inconvenience

		pALNS		Hong [14]		Lackner [19]	
Group	δ	**Dist.**	**Rej.**	**Dist.**	**Rej.**	**Dist.**	**Rej.**
R1	**10**	1197.4	0.25	1257.1 (4.99%)	0.17	1278.1 (6.74%)	0.47
	30	1212.9	0.80	1286.6 (6.08%)	0.58	1337.9 (10.30%)	0.72
	50	1225.0	1.25	1295.8 (5.78%)	0.67	1330.0 (8.57%)	0.78
	70	1237.3	1.71	1331.3 (7.60%)	1.75	1336.1 (7.98%)	0.94
	90	1230.1	2.55	1335.9 (8.60%)	2.33	1278.3 (3.92%)	0.75
C1	**10**	850.6	0.11	895.8 (5.31%)	0.22	996.4 (17.14%)	0.00
	30	874.9	0.11	962.1 (9.97%)	0.33	1066.9 (21.95%)	0.00
	50	903.4	0.11	1001.2 (10.82%)	0.22	1236.1 (36.82%)	0.00
	70	919.1	0.11	1031.7 (12.25%)	0.22	1261.3 (37.24%)	0.00
	90	929.9	0.11	1039.8 (11.81%)	0.22	1479.6 (59.11%)	0.00
RC1	**10**	1389.4	0.04	1436.2 (3.37%)	1.13	1426.9 (2.70%)	0.46
	30	1421.5	0.28	1492.2 (4.98%)	1.13	1439.7 (1.28%)	0.42
	50	1463.4	0.23	1514.7 (3.50%)	1.38	1448.1 (-1.05%)	0.46
	70	1470.1	0.58	1511.3 (2.80%)	1.88	1488.4 (1.25%)	0.58
	90	1495.5	0.51	1513.9 (1.23%)	2.00	1475.2 (-1.36%)	0.42
R2	**10**	893.0	0.00	950.0 (6.39%)	0.09	1052.9 (17.90%)	0.03
	30	915.6	0.00	985.5 (7.63%)	0.00	1085.4 (18.54%)	0.15
	50	948.6	0.00	1016.5 (7.17%)	0.00	1138.8 (20.05%)	0.21
	70	967.7	0.00	1032.0 (6.65%)	0.09	1116.9 (15.42%)	0.30
	90	981.7	0.00	1047.8 (6.73%)	0.09	1193.3 (21.55%)	0.52
C2	**10**	597.2	0.00	594.7 (-0.42%)	0.00	629.1 (5.35%)	0.00
	30	597.6	0.00	651.4 (9.01%)	0.00	632.3 (5.81%)	0.04
	50	604.0	0.00	605.0 (0.17%)	0.00	689.3 (14.12%)	0.13
	70	619.2	0.00	636.5 (2.79%)	0.00	743.8 (20.12%)	0.21
	90	625.7	0.00	636.8 (1.78%)	0.00	792.5 (26.66%)	0.29
RC2	**10**	1024.4	0.00	1103.3 (7.70%)	0.00	1220.9 (19.18%)	0.00
	30	1053.1	0.00	1166.0 (10.73%)	0.25	1244.9 (18.21%)	0.04
	50	1060.5	0.00	1190.5 (12.26%)	0.13	1244.9 (17.38%)	0.00
	70	1091.4	0.00	1239.5 (13.57%)	0.00	1269.3 (16.30%)	0.00
	90	1130.3	0.00	1257.2 (11.23%)	0.13	1346.8 (19.16%)	0.13
Average			**0.29**	**(+6.75%)**	**0.50**	**(+15.61%)**	**0.27**

Table 3: Comparison of approaches for the Lackner [19] instances.

is therefore equivalent to minimizing the number of changes communicated to the driver.

We use the *edit distance* (or Levenshtein distance) as a proxy for the driver's inconvenience. The edit distance between two routes is defined as the minimum number of insertions, removals, or substitutions of customers that have to be applied to transform one route into the other. Therefore the inconvenience of a new solution relative to a reference solution is equal to the sum of edit distances between each vehicle's reference and new routes. The advantage of this metric is that it is efficiently computed and models accurately the changes described above, and it can be adapted to give weights to each type of change. The main limitation of this proxy is that it does not necessarily reflect the effective number of changes communicated to the driver as sections of the route may be changed later.

Figure 7 illustrates the evaluation of the edit distance between a reference and a new route. The gray nodes correspond to the portion of the route that has already been executed. The distance between the reference and new route is 3, with 1 substitution (SUB), 1 insertion (INS), and 1 removal (REM).

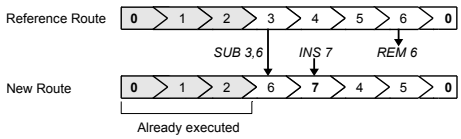


Figure 7: Example of the edit distance between two routes.

3.2 The proposed approach

The proposed approach, namely parallel Bi-objective Adaptive Large Neighborhood Search (pBiALNS), is an extension of the pALNS algorithm described in Section 2, and it is inspired by the bi-objective LNS proposed by Schmid and Hartl [33]. In a nutshell, the central idea is to maintain and optimize a set of non-dominated and possibly infeasible solutions. In addition, our approach introduces a parallelization scheme that improves performance and allows its use in a dynamic context.

The adaptation of the pALNS algorithm to deal with the bi-objective case is straightforward: the algorithm maintains the set $\bar{\mathcal{P}}$ of non-dominated solutions that are optimized in K subprocesses. For I^m master iterations, a subset of K non-dominated solutions is selected randomly and distributed among independent subprocesses. Each subprocess performs I^p ALNS iterations by destroying and repairing the current solution, considering only the main objective (cost). In contrast to the original pALNS algorithm, each temporary solution is considered for inclusion in the set of non-dominated solutions, and the number of solutions stored in $\bar{\mathcal{P}}$ is not limited. Finally, the algorithm returns the whole set of non-dominated solutions $\bar{\mathcal{P}}$, from which the decision maker selects a single solution.

It is important to note that the optimization itself, which takes place in the ALNS iterations, only considers the minimization of the cost. Therefore, there is an implicit lexicographic ordering of the objectives, the maximization of the consistency being handled implicitly with the set of non-dominated solutions. This choice is motivated by the fact that at each ALNS iteration the algorithm needs to introduce changes in the current solution by removing and inserting customers, and introducing the consistency at this level would steer the approach away from cost-effective solutions.

Note that pBiALNS may visit infeasible solutions that do not visit all customers. Therefore, we define a dominance relation that ensures that no feasible solution will be dominated by an infeasible solution:

Definition 1 (Dominance). *A solution* Π *dominates (denoted* $\prec$*) a solution* Π' *if and only if* Π *is as good as* Π' *in both objectives, and strictly better in one objective, and either* Π *is feasible or both* Π *and* Π' *are infeasible.*

3.3 Computational results

We tested the pBiALNS approach on the Lackner [19] instances described in §2.4 with a similar experimental setting. pALNS is first run for 25,000 iterations to produce the *reference* (initial) solution; then, each time a new customer appears pBiALNS is run for 5,000 iterations to produce a set of candidate *new* solutions to choose from; finally, pALNS is run for 50,000 iterations to produce the a-posteriori solution to the problem.

Figure 8 represents the objective space explored by pBiALNS after 5,0000 iterations for one instance, at a given step of the simulation (ie., after a new customer appeared). The graph illustrates the diversity of solutions offered to the decision maker, ranging from the least-cost solution (upper left) to the most similar to the reference solution (lower right). For the purpose of benchmarking and to assess the tradeoff between the two objectives, we define a *threshold selection policy* and select the non-dominated solution that is closest to the reference, allowing a deviation in cost of at most γ percent from the least-cost solution (green diamond). This policy models the behavior of an expert dispatcher who would select one solution among the non-dominated set.

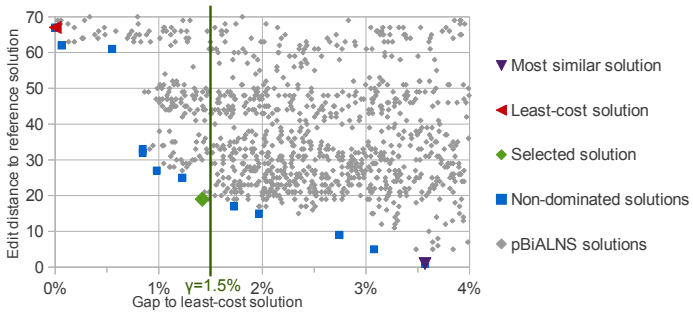


Figure 8: Objective space for instance R101 and illustration of the threshold policy.

Table 4 presents (a) the average edit distance between the final solution and the reference solution, and (b) the average gap between the cost of the final solution and the cost of a solution evaluated a-posteriori and the average number of rejected requests, for different values of γ and degree of dynamism (δ). Running times are of 2.5 seconds on average at each decision. As expected, the edit distance relative to the reference solution is negatively correlated to γ, and is minimal for $\gamma = \infty$. In this case we always choose the solution which is the closest to the reference solution, in other words we simply insert new customers in the current solution, which leads to a distance equal to the number of accepted dynamic customers. It is important to note that the quality of the routing, measured by the gap to the static solution, is positively correlated to γ. This confirms the intuition that poor routing decisions tend to add

up over time and can lead to larger deviations at the end of the day. Our results also indicate that, for problems with low degree of dynamism, it can be worth sacrificing quality of solution to gain route stability. For instance, with $\delta = 10$, the value $\gamma = 5\%$ leads to a gap of 6% versus 2% with $\gamma = 0\%$, but it reduces the number of required changes by a factor 3. However, this statement no longer holds for instances with higher degrees of dynamism where numerous changes are necessary to insert all customers. In this case it is better to focus on optimizing the routing, as it does not lead to excessive instability in routes.

(a) Average edit distance to reference solution

	γ					
δ	**0%**	**1%**	**2%**	**5%**	**10%**	∞
10	32.8	19.3	17.0	12.9	12.6	9.8
30	59.4	48.1	44.2	39.2	36.4	29.6
50	78.2	70.3	65.9	61.4	58.2	49.4
70	87.6	84.0	81.7	78.5	75.7	69.2
90	95.7	94.5	93.9	92.7	91.3	88.9

(b) Average gap to a-posteriori solution (%) and number of rejected requests (in parenthesis)

	γ					
δ	**0%**	**1%**	**2%**	**5%**	**10%**	∞
10	2.0 (0.1)	2.8 (0.1)	4.2 (0.1)	6.1 (0.1)	8.1 (0.1)	11.2 (0.2)
30	4.3 (0.3)	5.6 (0.3)	6.5 (0.3)	10.9 (0.2)	16.3 (0.2)	29.3 (0.4)
50	6.4 (0.3)	7.6 (0.3)	9.1 (0.4)	13.1 (0.3)	18.7 (0.3)	50.1 (0.6)
70	9.0 (0.4)	10.3 (0.4)	11.8 (0.5)	15.3 (0.5)	20.5 (0.4)	71.0 (0.8)
90	9.8 (0.7)	10.8 (0.7)	11.6 (0.6)	14.4 (0.6)	19.4 (0.7)	95.5 (1.1)

Table 4: Evolution of the distance to reference solution and gap to a-posteriori solution for different degrees of dynamism and values of γ

4 Conclusions

In this work we proposed an efficient parallelization scheme for an Adaptive Large Neighborhood Search, namely pALNS. This algorithm distributes the optimization of promising solutions across multiple processors, resulting in factor 3.3 speedups on a quad-core desktop machine. The efficiency of pALNS relies on the presence of a promising solution pool with diversity management, which prevents deadlocks between optimization threads, and improves the exploration of the search space. We illustrated the efficiency of pALNS on the Solomon [35] CVRPTW instances, for which it produces solutions in average 0.7% away from the optimal/best known solution in just 12s.

We also introduced a fast-reoptimization approach based on pALNS to tackle the dynamic VRPTW. This approach consists in running pALNS to produce an initial solution at the beginning of the day, and then running it for a limited number of iterations whenever a new customer appears. We tested our approach on the instance set proposed by Lackner [19]. Computational results show that pALNS is capable of

achieving state of the art results in competitive time, bringing improvements of up to 12% over previous approaches.

Finally, we presented a preliminary bi-objective extension of the classical D-VRPTW that attempts to captures the drivers inconvenience resulting from dynamic routing. It is based on the notion of having a reference routing plan handled to the drivers at the beginning of the period, that will then undergo changes as new customers arrive. We introduced an inconvenience metric that measures the consistency between an updated routing plan and the reference plan. We proposed a fast bi-objective optimization approach based on pALNS, namely pBiALNS, which maintains and optimizes in parallel the set of non-dominated solutions.

This optimization algorithm was used coupled with a threshold policy modeling an expert dispatcher to tackle the D-VRPTW instances proposed by Lackner [19]. Our results indicate that there is a clear tradeoff between minimizing the traveled distance and maintaining consistency in routes. Furthermore, it appears that for problems with a low degree of dynamism it can be worth sacrificing cost efficiency to maintain consistency. In contrast, in highly dynamic problems the priority should be given to the minimization of the cost, as it does not lead to excessive inconsistency in routing.

Future research should focus on the development of a continuous reoptimization approach based on pALNS that runs throughout the day and maintains a pool of alternative promising solutions as adaptive memory. In addition, pALNS could be improved by having completely independent subprocesses that pull their starting solution from the pool, and push their final solution, without waiting for other subprocesses to finish. pBiALNS could be refined to better approximate the Pareto front, first in the selection of the non-dominated solutions to optimize, then by applying a local search or a path relinking between non-dominated solutions.

Acknowledgements Financial support for this work was provided by the CPER Vallée du Libre (Contrat de Projet Etat Region, France); and the CEIBA (Centro de Estudios Interdisciplinarios Básicos y Aplicados en Complejidad, Colombia). This support is gratefully acknowledged.

References

[1] Attanasio, A., Cordeau, J. F., Ghiani, G., and Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, doi:10.1016/j.parco.2003.12.001.

[2] Barcelo, J., Grzybowska, H., and Pardo, S. (2007). Vehicle routing and scheduling models, simulation and city logistics. In Zeimpekis, V., Tarantilis, C. D., Giaglis, G. M., and Minis, I., editors, *Dynamic Fleet Management*, volume 38 of *Operations Research/Computer Science Interfaces*, pages 163–195. Springer US.

[3] Beaudry, A., Laporte, G., Melo, T., and Nickel, S. (2010). Dynamic transportation of patients in hospitals. *OR Spectrum*, 32:77–107, doi:10.1007/s00291-008-0135-6.

[4] Bent, R. and Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6):977–987.

[5] Benyahia, I. and Potvin, J. Y. (1998). Decision support for vehicle dispatching using genetic programming. *IEEE Transactions on Systems Man and Cybernetics Part A - Systems and Humans*, 28(3):306–314.

[6] Chang, M. S., Chen, S., and Hsueh, C. (2003). Real-time vehicle routing problem with time windows and simultaneous delivery/pickup demands. *Journal of the Eastern Asia Society for Transportation Studies*, 5:2273–2286.

[7] Chen, Z. and Xu, H. (2006). Dynamic column generation for dynamic vehicle routing with time windows. *Transportation Science*, 40(1):74–88.

[8] Cheung, B. K. S., Choy, K. L., Li, C.-L., Shi, W., and Tang, J. (2008). Dynamic routing model and solution methods for fleet management with mobile technologies. *International Journal of Production Economics*, 113(2):694–705, doi:10.1016/j.ijpe.2007.10.018.

[9] Gambardella, L., Rizzoli, A., Oliverio, F., Casagrande, N., Donati, A., Montemanni, R., and Lucibello, E. (2003). Ant colony optimization for vehicle routing in advanced logistics systems. In *Proceedings of the International Workshop on Modelling and Applied Simulation (MAS 2003)*, pages 3–9.

[10] Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, E. (1999). Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33(4):381–390, doi:10.1287/trsc.33.4.381.

[11] Gendreau, M. and Potvin, J.-Y., editors (2004). *Transportation Science*, volume 4. INFORMS. Special issue on real-time fleet management.

[12] Ghiani, G., Guerriero, F., Laporte, G., and Musmanno, R. (2003). Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151(1):1 – 11, doi:10.1016/S0377-2217(02)00915-3.

[13] Haghani, A. and Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, 32(11):2959 – 2986, doi:10.1016/j.cor.2004.04.013.

[14] Hong, L. (2012). An improved lns algorithm for real-time vehicle routing problem with time windows. *Computers & Operations Research*, 39(2):151 – 163, doi:10.1016/j.cor.2011.03.006.

[15] Ichoua, S., Gendreau, M., and Potvin, J.-Y. (2000). Diversion issues in real-time vehicle dispatching. *Transportation Science*, 34(4):426–438, doi:10.1287/trsc.34.4.426.12325.

[16] Ichoua, S., Gendreau, M., and Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379 – 396, doi:10.1016/S0377-2217(02)00147-9.

[17] Ichoua, S., Gendreau, M., and Potvin, J.-Y. (2007). Planned route optimization for real-time vehicle routing. In Zeimpekis, V., Tarantilis, C. D., Giaglis, G. M., and Minis, I., editors, *Dynamic Fleet Management*, volume 38 of *Operations Research/Computer Science Interfaces*, pages 1–18. Springer US.

[18] Jozefowiez, N., Semet, F., and Talbi, E.-G. (2008). Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293 – 309, doi:10.1016/j.ejor.2007.05.055.

[19] Lackner, A. (2004). *Dynamische Tourenplanung mit ausgewählten Metaheuristiken*. PhD thesis, Georg-August-Universität Göttingen.

[20] Larsen, A. (2001). *The Dynamic Vehicle Routing Problem*. PhD thesis, Technical University of Denmark (DTU).

[21] Larsen, A., Madsen, O. B. G., and Solomon, M. M. (2008). Recent developments in dynamic vehicle routing systems. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 199–218. Springer US.

[22] Lund, K., Madsen, O. B. G., and Rygaard, J. M. (1996). Vehicle routing problems with varying degrees of dynamism. Technical report, IMM Institute of Mathematical Modelling.

[23] Montemanni, R., Gambardella, L. M., Rizzoli, A. E., and Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4):327–343, doi:10.1007/s10878-005-4922-6.

[24] Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2011). A review of dynamic vehicle routing problems. Technical Report CIRRELT-2011-62, CIRRELT.

[25] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, doi:10.1016/j.cor.2005.09.012.

[26] Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US.

[27] Potvin, J.-Y. and Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331 – 340, doi:10.1016/0377-2217(93)90221-8.

[28] Powell, W. B. (2007). *Approximate dynamic programming: solving the curses of dimensionality*, volume 703 of *Wiley Series in Probability and Statistics*. Wiley-Interscience, Hoboken, New Jersey.

[29] Prins, C. (2009). Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*, 22(6):916–928, doi:10.1016/j.engappai.2008.10.006.

[30] Psaraftis, H. (1980). A dynamic-programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154.

[31] Rizzoli, A., Montemanni, R., Lucibello, E., and Gambardella, L. (2007). Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1(sa):135–151.

[32] Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

[33] Schmid, V. and Hartl, R. F. (2011). Large neighborhood search for solving the Bi-Objective Capacitated m-Ring-Star Problem. In Di Gaspero, L., Schaerf, A., and Stützle, T., editors, *Proceedings of the 9th Metaheuristics Conference (MIC 2011)*, pages 700–703. Università degli Studi di Udine.

[34] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin / Heidelberg.

[35] Solomon, M. M. (1987). Algorithms for the vehicle-routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

[36] Taillard, E., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

[37] Taillard, E. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1 – 16, doi:10.1016/S0377-2217(00)00268-X.

[38] Van Hemert, J. I. and Poutré, J. L. (2004). Dynamic routing problems with fruitful regions: Models and evolutionary computation. In Yao, X., Burke, E., Lozano, J. A., Smith, J., Merelo-Guervós, J. J., Bullinaria, J. A., Rowe, J., Tino, P., Kabán, A., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature*, volume 3242 of *Lecture Notes in Computer Science*, pages 692–701. Springer Berlin / Heidelberg.

[39] Vidal, T., Crainic, T., Gendreau, M., and Prins, C. (2011). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time windows. Technical Report CIRRELT-2011-61, CIRRELT.

[40] Zeimpekis, V., Tarantilis, C. D., Giaglis, G. M., and Minis, I., editors (2007). *Dynamic Fleet Management*, volume 38 of *Operations Research Computer Science Interfaces Series*. Springer US.

Appendices

A Parameter setting

Table 5 presents the detail parameter setting used in the pALNS algorithm. The number of parallel iterations and the maximum size of the pool where selected after running experiments with values $I^p \in \{10, 50, 100, 500, 1000\}$ and $N \in \{1, 5, 10, 20, 30, 40, 50\}$. We also tested two schemes for the solution pool, the first with a fixed value of 0.5 for λ, the second using an adaptive scheme starting with $\lambda = 0.5$ and decreasing its value using the same process as the one used to decrease the simulated

annealing temperature. Over all our experiments the combination of an adaptive diversity management with $I^p = 50$ and $N = 40$ showed the best results for 25,000 pALNS iterations, and $I^p = 100$ and $N = 10$ for 5,000 pALNS iterations. The remaining parameters were directly derived from the work by Pisinger and Ropke [25].

Parameter	Value	Description
K	8	Number of threads
I^p	50 (100)	Number of parallel iterations
N	40 (10)	Maximum promising solution pool size
ϕ	0.10	Penalization for unserved customers
ξ_{min}	0.10	Minimum proportion of customers to be removed
ξ_{max}	0.40	Maximum proportion of customers to be removed
w	0.05	Reference objective degradation
p	0.5	Initial probability of accepting a degrading solution
α	0.002	Fraction of the initial temperature to be reached at the end
ρ	0.40	Reaction factor
σ_1	1.00	Score for new best solution
σ_2	0.25	Score for improving solution
σ_3	0.40	Score for non-improving accepted solution
σ_4	0.00	Score for rejected solution
l	100	Operator probability (w_θ) update frequency

Table 5: Detailed parameter setting for the pALNS algorithm for 25,000 iterations, values in parenthesis indicate adjusted values for 5,000 iterations.