A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW

Michael Saint-Guillain, Yves Deville, Christine Solnon

A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW

Michael Saint-Guillain*, Yves Deville* & Christine Solnon**

*ICTEAM, Université catholique de Louvain, Belgium
**Université de Lyon, CNRS
**INSA-Lyon, LIRIS, UMR5205, F-69621, France

Abstract. We consider a dynamic vehicle routing problem with time windows and stochastic customers (DS-VRPTW), such that customers may request for services as vehicles have already started their tours. To solve this problem, the goal is to provide a decision rule for choosing, at each time step, the next action to perform in light of known requests and probabilistic knowledge on requests likelihood. We introduce a new decision rule, called Global Stochastic Assessment (GSA) rule for the DS-VRPTW, and we compare it with existing decision rules, such as MSA. In particular, we show that GSA fully integrates nonanticipativity constraints so that it leads to better decisions in our stochastic context. We describe a new heuristic approach for efficiently approximating our GSA rule. We introduce a new waiting strategy. Experiments on dynamic and stochastic benchmarks, which include instances of different degrees of dynamism, show that not only our approach is competitive with state-of-the-art methods, but also enables to compute meaningful offline solutions to fully dynamic problems where absolutely no a priori customer request is provided.

1 Introduction

Dynamic (or *online*) vehicle routing problems (D-VRPs) arise when information about demands is incomplete, *e.g.*, whenever a customer is able to submit a request during the online execution of a solution. D-VRP instances usually indicate the deterministic requests, *i.e.*, those that are known before the online process if any. Whenever some additional (stochastic) knowledge about unknown requests is available, the problem is said to be *stochastic*. We focus on the *Dynamic* and *Stochastic* VRP with *Time Windows* (DS-VRPTW). These problems arise in many practical situations, as door-to-door or door-to-hospital transportation of elderly or disabled persons. In many countries, authorities try to set up dial-a-ride services, but escalating operating costs and the complexity of satisfying all customer demands become rapidly unmanageable for solution methods based on human choices [10]. However, such complex dynamic problems need reliable and efficient algorithms that should first be assessed on reference problems, such as the DS-VRPTW.

In this paper, we present *a new heuristic method for solving the DS-VRPTW*, based on a Stochastic Programming modeling. By definition, our approach enables a *higher level of anticipation* than heuristic state-of-the-art methods. The resulting new online decision rule, called Global Stochastic Assessment (GSA), comes with a theoretical

analysis that clearly defines the nature of the method. We propose a *new waiting strategy* together with a heuristic algorithm that embeds GSA. We compare GSA with the state-of-the-art method MSA from [7], and provide a *comprehensive experimental study* that highlights the contributions of existing and new waiting and relocation strategies.

This paper is organized as follows. Section 2 describes the problem. Section 3 presents the state-of-the-art method we compare to and briefly discuss related works. GSA is then presented in Section 4. Section 5 describes an implementation that embeds GSA, based on heuristic local search. Finally, section 6 resumes the experimental results. A conclusion follows in section 7.

2 Description of the DS-VRPTW

Notations. We note $[l, u]$ the set of all integer values i such that $l \leq i \leq u$. A sequence $< x^i, x^{i+1}, \ldots, x^{i+k} >$ (with $k \geq 0$) is noted $x^{i..i+k}$, and the concatenation of two sequences $x^{i..j}$ and $x^{j+1..k}$ (with $i \leq j < k$) is noted $x^{i..j}.x^{j+1..k}$. Random variables are noted $\boldsymbol{\xi}$ and their realizations are noted ξ. We note $\xi \in \boldsymbol{\xi}$ the fact that ξ is a realization of $\boldsymbol{\xi}$, and $p(\boldsymbol{\xi} = \xi)$ the probability that the random variable $\boldsymbol{\xi}$ is realized to ξ. Finally, we note $\mathbb{E}_{\boldsymbol{\xi}}[f(\xi)]$ the expected value of $f(\boldsymbol{\xi})$ which is defined by $\mathbb{E}_{\boldsymbol{\xi}}[f(\xi)] = \sum_{\xi \in \boldsymbol{\xi}} p(\boldsymbol{\xi} = \xi) \cdot f(\xi)$.

Input Data of a DS-VRPTW. We consider a discrete time horizon $[1, H]$ such that each online event or decision occurs at a discrete time $t \in [1, H]$, whereas each offline event or decision occurs at time $t = 0$. The DS-VRPTW is defined on a complete and directed graph $G = (V, E)$. The set of vertices $V = [0, n]$ is composed of a depot (vertex 0) and n customer regions (vertices 1 to n). To each arc $(i, j) \in E$ is associated a travel time $t_{i,j} \in \mathbb{R}_{\geq 0}$, that is the time needed by a vehicle to travel from i to j, with $t_{i,j} \neq t_{j,i}$ in general. To each customer region $i \in [1, n]$ is associated a load q_i, a service duration $d_i \in [1, H]$ and a time window $[e_i, l_i]$ with $e_i, l_i \in [1, H]$ and $e_i \leq l_i$.

The set of all customer requests is $R \subseteq [1, n] \times [0, H]$. For each request $(i, t) \in R$, t is the time when the request is revealed. When $t = 0$, the request is known before the online execution and it is said to be *deterministic*. When $t > 0$, the request is revealed during the online execution at time t and it is said to be *online* (or dynamic). There may be several requests for a same vertex i which are revealed at different times. During the online execution, we only know a subset of the requests of R (*i.e.*, those which have already been revealed). However, for each time $t \in [1, H]$, we are provided a probability vector P^t such that, for each vertex $i \in [1, n]$, $P^t[i]$ is the probability that a request is revealed for i at time t.

There are k vehicles and all vehicles have the same capacity Q.

Solution of a DS-VRPTW. At the end of the time horizon, a solution is a subset of requests $R_a \subseteq R$ together with k routes (one for each vehicle). Requests in R_a are said to be *accepted*, whereas requests in $R \setminus R_a$ are said to be *rejected*. The routes must satisfy the constraints of the classical VRPTW restricted to the subset R_a of accepted requests, *i.e.*, each route must start at the depot at a time $t \geq 1$ and end at the depot at a time $t' \leq H$, and for each accepted request $(i, t) \in R_a$, there must be exactly one route

that arrives at vertex i at a time $t' \in [e_i, l_i]$ with a current load $l \leq Q - q_i$ and leaves vertex i at a time $t'' \geq t' + d_i$. The goal is to minimize the number of rejected requests.

As not all requests are known at time 0, the solution cannot be computed offline, and it is computed during the online execution. More precisely, at each time $t \in [1, H]$, an action a^t is computed. Each action a^t is composed of two parts: first, for each request $(i, t) \in R$ revealed at time t, the action a^t must either accept the request or reject it; second, for each vehicle, the action a^t must give operational decisions for this vehicle at time t (*i.e.*, service a request, travel towards a vertex, or wait at its current position). Before the online execution (at time 0), some decisions are computed offline. Therefore, we also have to compute a first action a^0.

A solution is a sequence of actions $a^{0..H}$ which covers the whole time horizon. This sequence must satisfy VRPTW constraints, *i.e.*, the actions of $a^{0..H}$ must define k routes such that each request accepted in $a^{0..H}$ is served once by one of these routes within the time window associated with the served vertex and without violating capacity constraints. We define the objective function ω such that $\omega(a^{0..t})$ is $+\infty$ if $a^{0..t}$ does not satisfy VRPTW constraints, and $\omega(a^{0..t})$ is the number of requests rejected in $a^{0..t}$ otherwise. Hence, a solution is a sequence $a^{0..H}$ such that $\omega(a^{0..H})$ is minimal at the end of the horizon.

Stochastic program. There are different notations used for formulating stochastic programs; we mainly use those from [8]. For each time $t \in [1, H]$, we have a vector of random variables $\boldsymbol{\xi}^t$ such that, for each vertex $i \in [1, n]$, $\boldsymbol{\xi}^t[i]$ is realized to 1 if a request for i is revealed at time t, and to 0 otherwise. The probability distribution of $\boldsymbol{\xi}^t$ is defined by P^t, *i.e.*, $p(\boldsymbol{\xi}^t[i] = 1) = P^t[i]$ and $p(\boldsymbol{\xi}^t[i] = 0) = 1 - P^t[i]$. We note $\boldsymbol{\xi}^{1..t}$ the random matrix composed of the random vectors $\boldsymbol{\xi}^1$ to $\boldsymbol{\xi}^t$. A realization $\xi^{1..H} \in \boldsymbol{\xi}^{1..H}$ is called a *scenario*.

At each time $t \in [1, H]$, the action a^t must contain one accept or reject for each request which is revealed in ξ^t. Therefore, we note $A(\xi^t)$ the set of all actions that contain an accept or a reject for each vertex $i \in [1, n]$ such that $\xi^t[i] = 1$. Of course, these actions also contain other decisions related to the k vehicles. We also note $A(\xi^{t1..t2})$ the sequence of sets $< A(\xi^{t1}), \ldots, A(\xi^{t2}) >$ where $t1 \leq t2$.

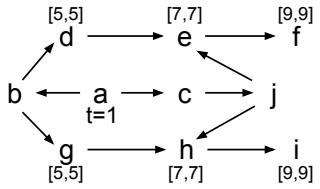
Hence, at each time t, given the sequence $a^{0..t-1}$ of past actions, the best action a^t is obtained by solving the multistage stochastic problem defined by eq. (1):

$$a^t = \underset{a^t \in A(\xi^t)}{\operatorname{argmin}} \mathbb{E}_{\boldsymbol{\xi}^{t+1}} \left[\min_{a^{t+1} \in A(\xi^{t+1})} \mathbb{E}_{\boldsymbol{\xi}^{t+2}} \left[\cdots \min_{a^{H-1} \in A(\xi^{H-1})} \mathbb{E}_{\boldsymbol{\xi}^{H}} \left[\min_{a^{H} \in A(\xi^{H})} \omega(a^{0..H}) \right] \cdots \right] \right] \tag{1}$$

Note that this multistage stochastic problem is different from the two-stage stochastic problem defined by eq. (2):

$$a^t = \underset{a^t \in A(\xi^t)}{\arg\min} \mathbb{E}_{\boldsymbol{\xi}^{t+1..H}} [\min_{a^{t+1..H} \in A(\xi^{t+1..H})} \omega(a^{0..H})] \tag{2}$$

Indeed, eq. (1) enforces *nonanticipativity constraints* so that, at each time $t' > t$, we consider the action $a^{t'}$ which minimizes the expectation with respect to $\boldsymbol{\xi}^{t'}$ only, without considering the possible realizations of $\boldsymbol{\xi}^{t'+1..H}$. Eq. (2) does not enforce these constraints and considers the best sequence $a^{t+1..H}$ for each realization $\xi^{t+1..H} \in \boldsymbol{\xi}^{t+1..H}$.



Time	2	3	4	5	6	7	8	9
Scenario $\xi_1^{2..5}$	$\emptyset$	$\emptyset$	$\{d, e, f\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
Scenario $\xi_2^{2..5}$	$\emptyset$	$\emptyset$	$\{g, h, i\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

At time $t = 1$, there is only 1 vehicle which is on vertex a, and we have to choose between 2 possible actions: *travel to b* or *travel to c*

Fig. 1. A simple example of nonanticipation. The graph is displayed on the left. Time windows are displayed in brackets. For every couple of vertices (i, j), if an arrow $i \to j$ is displayed then $t_{i,j} = 2$; otherwise $t_{i,j} = 20$. To simplify, we consider only 2 equiprobable scenarios (displayed on the right). These scenarios have the same prefix (at times 2 and 3 no request is revealed) but reveal different requests at time 4. When using eq. (1) at time $t = 1$, we choose to travel to c as the expected cost with nonanticipativity constraints is 1: At time 4, only one scenario will remain and if this scenario is ξ_1 (resp. ξ_2), request $(d, 4)$ (resp. $(g, 4)$) will be rejected. When using eq. (2), we choose to travel to b as the expected cost without nonanticipativity constraints is 0 (for each possible scenario, there exists a sequence of actions which serves all requests: travel to d, e, and f for ξ_1 and travel to g, h, and i for ξ_2). However, if we travel to b, at time 3 we will have to choose between traveling to d or g and at this time the expected cost of both actions will be 1.5: If we travel to d (resp. g), the cost with scenario ξ_1 is 0 (resp. 3) and the cost with scenario ξ_2 is 3 (resp. 0). In this example, the nonanticipativity contraints of multistage problem (1) thus leads to a better action than the two-stage relaxation (2).

Therefore, eq. (1) may lead to a larger expectation of ω than eq. (2), as it is more constrained. However, the expectation computed in eq. (1) leads to better decisions in our context where some requests are not revealed at time t. This is illustrated in Fig. 1.

3 Related Work

The first D-VRP is proposed in [29], which introduces a single vehicle Dynamic Dial-a-Ride Problem (D-DARP) in which customer requests appear dynamically. Then, [20] introduced the concept of immediate requests that must be serviced as soon as possible, implying a replanning of the current vehicle route. Complete reviews on D-VRP may be found in [21,18]. In this section, we more particularly focus on stochastic D-VRP. [18] classifies approaches for stochastic D-VRP in two categories, either based on *stochastic modeling* or on *sampling*. Stochastic modeling approaches formally capture the stochastic nature of the problem, so that solutions are computed in the light of an overall stochastic context. Such holistic approaches usually require strong assumptions and efficient computation of complex expected values. Sampling approaches try to capture stochastic knowledge by sampling scenarios, so that they tend to be more focused on local stochastic evidences. Their local decisions however allow sample-based methods to scale up to larger problem instances, even under challenging timing constraints. One usually needs to find a good compromise between having a high number of scenarios, providing a better representation of the real distributions, and a more restricted number of these leading to less computational effort.

Algorithm 1: The ChooseRequest-ε Expectation Algorithm

1 **for** $a^t \in A(\xi^t)$ **do** $f(a^t) \leftarrow 0$;
2 Generate a set S of α scenarios using Monte Carlo sampling
3 **for** *each scenario* $s \in S$ *and each action* $a^t \in A(\xi^t)$ **do**
4 $\quad f(a^t) \leftarrow f(a^t)$+cost of (approximate) solution to scenario s starting with a^t
5 **return** $\arg\min_{a^t \in A(\xi^t)} f(a^t)$

[7] studies the DS-VRPTW and introduces the Multiple Scenario Approach (MSA). A key element of MSA is an adaptive memory that stores a pool of solutions. Each solution is computed by considering a particular scenario which is optimized for a few seconds. The pool is continuously populated and filtered such that all solutions are consistent with the current system state. Another important element of MSA is the *ranking function* used to make operational decisions involving idle vehicles. The authors designed 3 algorithms for that purpose:

- *Expectation* [3,4] samples a set of scenarios and selects the next request to be serviced by considering its average cost on the sampled set of scenarios. Algorithm 1 [27] depicts how it chooses the next action a^t to perform. It requires an optimization for each action $a^t \in A(\xi^t)$ and each scenario $s \in S$ (lines 3-4), which is computationally very expensive, even with a heuristic approach.
- *Regret* [3,6] approximates the expectation algorithm by recognizing that, given a solution sol_s^* to a particular scenario s, it is possible to compute a good approximation of the local loss inquired by performing another action than the next planned one in sol_s^*.
- *Consensus* [4,7] selects the request that appears the most frequently as the next serviced request in the solution pool.

Quite similar to the consensus algorithm is the Dynamic Sample Scenario Hedging Heuristic introduced by [14] for the stochastic VRP. Also, [15] designed a Tabu Search heuristic for the DS-VRPTW and introduced a vehicle-waiting strategy computed on a future request probability threshold in the near region. Finally, [5] extends MSA with waiting and relocation strategies so that the vehicles are now able to relocate to promising but unrequested yet vertices. As the performances of MSA has been demonstrated in several studies [5,12,22,19], it is still considered as a state-of-the-art method for dealing with DS-VRPTW.

Other studies of particular interest for our paper are [13], on the dynamic and stochastic pickup and delivery problem, and [22], on the DS-DARP. Both consider local search based algorithms. Instead of a solution pool, they exploit one single solution that minimizes the expected cost over a set of scenarios. However, in order to limit computational effort, only near future requests are sampled within each scenario. Although the approach of [22] is similar to the one of [13], the set of scenarios considered is reduced to one scenario. Although these later papers show some similarities with the approach we propose, they do not provide any mathematical motivation and analysis of their methods.

4 The global Stochastic Assessment decision rule

The two-stage stochastic problem defined by eq. (2) may be solved by a sampling solving method such as MSA, which solves a deterministic VRPTW for each possible scenario (i.e., realization of the random variables) and selects the action a^t which minimizes the sum of all minimum objective function values weighted by scenario probabilities. However, we have shown in Section 2 that eq. (2) does not enforce nonanticipativity constraints because the different deterministic VRPTW are solved independently. To enforce nonanticipativity constraints while enabling sampling methods, we push these constraints in the computation of the optimal solutions for all different scenarios: Instead of computing these different optimal solutions independently, we propose to compute them all together so that we can ensure that whenever two scenarios share a same prefix of realizations, the corresponding actions are enforced to be equal.

At each time $t \in [0, H]$, let r be the number of different possible realizations of $\boldsymbol{\xi}^{t+1..H}$, and let us note $\xi_1^{t+1..H}, \ldots, \xi_r^{t+1..H}$ these realizations. Given the sequence $a^{0..t-1}$ of past actions, we choose action a^t by using eq. (3)

$$a^t = \underset{a^t \in A(\xi^t)}{\arg\min}\, \mathcal{Q}(a^{0..t}, \{\xi_1^{t+1..H}, \ldots, \xi_r^{t+1..H}\}) \tag{3}$$

which is called the deterministic equivalent form of eq. (1).
$\mathcal{Q}(a^{0..t}, \{\xi_1^{t+1..H}, \ldots, \xi_r^{t+1..H}\})$ solves the deterministic optimization problem

$$\min_{a_1^{t+1..H} \in A(\xi_1^{t+1..H}), \ldots, a_r^{t+1..H} \in A(\xi_r^{t+1..H})} \sum_{i=1}^{r} p(\boldsymbol{\xi}^{t+1..H} = \xi_i^{t+1..H}) \cdot \omega(a^{0..t}.a_i^{t+1..H}) \tag{4}$$

$$s.t.\ (\xi_i^{t+1..t'} = \xi_j^{t+1..t'}) \Rightarrow (a_i^{t+1..t'} = a_j^{t+1..t'}),\ \forall t' \in [t+1, H],\ \forall i, j \in [1, r] \tag{5}$$

The nonanticipativity constraints (5) state that, when 2 realizations $\xi_i^{t+1..H}$ and $\xi_j^{t+1..H}$ share a same prefix from $t+1$ to t', the corresponding actions must be equal [23].

Solving eq. (3) is computationally intractable for two reasons. First, since the number r of possible realizations of $\boldsymbol{\xi}^{t+1..H}$ is exponential in the number of vertices and in the remaining horizon size $H - t$, considering every possible scenario is intractable in practice. We therefore consider a smaller set of α scenarios $S = \{s_1, ..., s_\alpha\}$ such that each scenario $s_i \in S$ is a realization of $\boldsymbol{\xi}^{t+1..H}$, *i.e.*, $\forall i \in [1, \alpha], s_i \in \boldsymbol{\xi}^{t+1..H}$. This set S is obtained by Monte Carlo sampling [2]. All elements of S share the same probability, *i.e.*, $p(\boldsymbol{\xi}^{t+1..H} = s_1) = \ldots = p(\boldsymbol{\xi}^{t+1..H} = s_\alpha)$.

Second, solving eq. (3) basically involves solving to optimality problem $\mathcal{Q}$ for each possible action $a^t \in A(\xi^t)$. Each problem $\mathcal{Q}$ involves solving a VRPTW for each possible scenario of S, while ensuring nonanticipativity constraints between the different solutions. As the VRPTW problem is an $\mathcal{NP}$-hard problem, we propose to compute an upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$ based on a given sequence $a_R^{t+1..H}$ of future route actions. Because we impose the sequence $a_R^{t+1..H}$, the set of possible actions at time t is limited to those directly compatible with it, denoted $\tilde{A}(\xi^t, a_R^{t+1..H}) \subseteq A(\xi^t)$. That limitation enforces $\omega(a^{0..H}) < +\infty$. This finally leads to the GSA decision rule:

(GSA)
$$a^t = \underset{a^t \in \tilde{A}(\xi^t, a_R^{t+1..H})}{\arg\min}\, \overline{\mathcal{Q}}(a^{0..t}, a_R^{t+1..H}, S) \tag{6}$$

which, provided realization ξ^t, sampled scenarios S and future route actions $a_R^{t+1..H}$, selects the action a^t that minimizes the expected approximate cost over scenarios S. Notice that almost all the anticipative efficiency of the GSA decision rule relies on the sequence $a_R^{t+1..H}$, which directly affects the quality of the upper bound $\overline{\mathcal{Q}}$.

Sequence $a_R^{t+1..H}$ of future route actions. This sequence is used to compute an upper bound of $\mathcal{Q}$. For each time $t' \in [t+1, H]$, the route action $a_R^{t'}$ only contains operational decisions related to vehicle routing (*i.e.*, for each vehicle, travel towards a vertex, or wait at its current position) and does not contain decisions related to requests (*i.e.*, request acceptance or rejection). The more flexible $a_R^{t'}$ with respect to S, the better the bound $\overline{\mathcal{Q}}$. We describe in Section 5 how a flexible sequence is computed through local search.

Computation of an upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$. Algorithm 2 depicts the computation of an upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$ given a sequence $a_R^{t+1..H}$ of route actions consistent with past actions $a^{0..t}$. For each scenario s_i of S, Algorithm 2 builds a sequence $b^{0..H}$ for s_i, which starts with $a^{0..t}$, and whose end $b^{t+1..H}$ is computed from $a_R^{t+1..H}$ in a greedy way. At each time $t' \in [t+1..H]$, each request revealed at time t' in scenario s_i is accepted if it is possible to modify $b^{t'..H}$ so that one vehicle can service it; it is rejected otherwise. One can consider $b^{t'..H}$ as being a set of vehicle routes, each defined by a sequence of planned vertices. Each planned vertex comes with specific decisions: a waiting time and whether a service is performed. In this context, $trytoServe$ performs a deterministic linear time modification of $b^{t'..H}$ such that (j, t') corresponds to the insertion of the vertex j in one of the routes defined by $b^{t'..H}$, at the best position with respect to VRPTW constraints and travel times, without modifying the order of the remaining vertices. At the end, Algorithm 2 returns the average number of rejected requests for all scenarios. Note that, when modifying a sequence of actions so that a request can be accepted (line 6), actions $b^{t'..H}$ can be modified, but $b^{0..t'-1}$ are not modified. This ensures that $\overline{\mathcal{Q}}$ preserves the nonanticipativity constraints. Indeed, the fact that two identical scenarios prefixes could be assigned two different subsequences of actions implies that either $trytoServe((j, t'), b^{t'..H})$ is able to modify an action

Algorithm 2: The $\overline{\mathcal{Q}}(a^{0..t}, a_R^{t+1..H}, S)$ approximation function

1 Precondition: $a_R^{t+1..H}$ is a sequence of route actions consistent with $a^{0..t}$
2 **for** *each scenario* $s_i \in S$ **do**
3 $\quad nbRejected[i] \leftarrow 0$; $b^{0..t} \leftarrow a^{0..t}$; $b^{t+1..H} \leftarrow a_R^{t+1..H}$
4 $\quad$ **for** $t' \in [t+1..H]$ **do**
5 $\quad\quad$ **for** *each request* (j, t') *revealed at time* t' *for a vertex* j *in scenario* s_i **do**
6 $\quad\quad\quad c^{t'..H} \leftarrow trytoServe((j, t'), b^{t'..H})$
7 $\quad\quad\quad$ **if** $b^{t+1..t'-1} \cdot c^{t'..H}$ *is feasible* **then** $b^{t'..H} \leftarrow c^{t'..H}$
8 $\quad\quad\quad$ **else** add the decision *reject(j,t')* to $b^{t'}$ and increment $nbRejected[i]$;
9 **return** $\frac{1}{|S|} \cdot \sum_{s_i \in S} nbRejected[i]$

$b^{t<t'}$ or is a nondeterministic function. In both cases, there is a contradiction. Finally, notice that contrary to other local search methods based on Monte Carlo simulation as in [13,22], GSA considers the whole timing horizon when evaluating a first-stage solution against a scenario.

Comparison to MSA GSA has two major differences with MSA. Given a set of scenarios, GSA maintains only one solution, namely the sequence $a_R^{t+1..H}$, that best suits to a pool of scenarios whilst MSA computes a set of solutions, each specialized to one scenario from the pool. Furthermore, by preserving nonanticipativity GSA approximates the multistage problem of equations (1,3). In contrary, MSA relaxes these constraints and therefore approximates the two-stage problem (2) [27].

In particular, given a pool of scenarios obtained by Monte Carlo sampling, MSA Expectation Algorithm 1 reformulates eq. (2) as a *sample average approximation* (SAA, [1,28]) problem. The SAA tackles each scenario as a separate deterministic problem. For a specific scenario $\xi^{t+1..H}$, it considers the recourse cost of a solution starting with actions $a^{0..t}$. Because the scenarios are not linked by nonanticipativity constraints, two scenarios i and j that share the same prefix $\xi^{t+1..t'}$ can actually be assigned two solutions performing completely different actions $a_i^{0..t'}$ and $a_j^{0..t'}$, for some $t' > t$. The evaluation of action a^t over the set of scenarios is therefore too optimistic, leading to a suboptimal choice. By definition, the *Regret* algorithm approximates the Expectation algorithm. The *Regret* algorithm then also approximates a two-stage problem. The *Consensus* algorithm selects the most suggested action among plans of the pool. By selecting the most frequent action in the pool, *Consensus* somehow encourages nonanticipation. However, the nonanticipativity constraints are not enforced as each scenario is solved separately. *Consensus* also approximates a two-stage problem.

5 Solving the Dynamic and Stochastic VRPTW

GSA alone does not permit to solve a DS-VRPTW instance. In this section, we now show how the decision rule, as defined in eq. 6, can be embedded in an online algorithm that solves the DS-VRPTW. Finally, we present the different waiting and relocation strategies we exploit, including a new waiting strategy.

5.1 Embedding GSA

In order to solve the DS-VRPTW, we design Algorithm 3, which embeds the GSA decision rule.

Main Algorithm. It is parameterized by: α which determines the size of the pool S of scenarios; β which determines the frequency for re-initializing S; and δ_{ins} which limits the time spent for trying to insert a request in a sequence.

It runs in *real time*. It is started before the beginning of the time horizon, in order to compute an initial pool S of α scenarios and an initial solution $a_R^{1..H}$ with respect to offline requests (revealed at time 0). It runs during the whole time horizon, and loops on lines 3 to 11. It is stopped when reaching the end of the time horizon. The *real*

Algorithm 3: LS-based GSA

1 Initialize S with α scenarios and compute initial solution $a_R^{1..H}$ w.r.t. known requests
2 $t \leftarrow 1$;
3 **while** *real time has not reached the end of the time horizon* **do**
  `/* Beginning of the time unit */`
4 $(a^t, a_R^{t+1..H}) \leftarrow$ `handleRequests`$(a^{0..t-1}, a_R^{t..H}, \xi^t)$
5 execute action a^t and update the pool S of scenarios w.r.t. to ξ^t
  `/* Remaining of the time unit */`
6 **while** *real time has not reached the end of time unit* t **do**
7 $b_R^{t+1..H} \leftarrow$ `shakeSolution`$(a_R^{t+1..H})$
8 **if** $\overline{\mathcal{Q}}(a^{0..t}, b_R^{t+1..H}, S) < \overline{\mathcal{Q}}(a^{0..t}, a_R^{t+1..H}, S)$ **then** $a_R^{t+1..H} \leftarrow b_R^{t+1..H}$;
9 **if** *the number of iterations since the last re-initialization of* S *is equal to* β **then**
10 Re-initialize the pool S of scenarios w.r.t. $\boldsymbol{\xi}^{t+1..H}$
11 $t \leftarrow t+1$ `/* Skip to next time unit */`
12 **Function** `handleRequests`$(a^{0..t-1}, a_R^{t..H}, \xi^t)$
13 $b^{0..t-1} \leftarrow a^{0..t-1}$; $b^{t..H} \leftarrow a_R^{t..H}$
14 **for** *each request revealed for a vertex* j *in realization* ξ^t **do**
15 **if** *we find, in less than* δ_{ins}*, how to modify* $b^{t..H}$ *s.t. request* (j,t) *is served* **then**
16 modify $b^{t..H}$ to accept request (j,t)
17 **else**
18 modify $b^{t..H}$ to reject request (j,t)
19 **return** $(b^t, b^{t+1..H})$

time is discretized in H time units, and the variable t represents the current time unit: It is incremented when real time exceeds the end of the t^{th} time unit. In order to be correct, Algorithm 3 requires the real computation time of lines 4 to 11 to be smaller than the real time spent in one time unit. This is achieved by choosing suitable values for parameters α and δ_{ins}.

Lines 4 and 5 describe what happens whenever the algorithm enters a new time unit: Function `handleRequests` (described below) chooses the next action a^t and updates $a_R^{t+1..H}$; Finally, S is updated such that it stays coherent with respect to realization ξ^t. Each scenario $\xi^{t..H} \in S$ is composed of a sequence of sampled requests. To each customer region i is associated an upper bound $\overline{r}_i = \min(l_0 - t_{i,0} - d_i, l_i - t_{0,i})$ on the time unit at which a request can be revealed in that region, like in [7]. That constraint prevents tricky or inserviceable requests to be sampled. At time t, a sampled request (i,t) which doesn't appear in ξ^t is either removed if $t \geq \overline{r}_i$ or randomly delayed in $\xi^{t+1..H} \in S$ otherwise.

The algorithm spends the rest of the time unit to iterate over lines 7 to 10, in order to improve the sequence of future route actions $a_R^{t+1..H}$. We consider a hill climbing strategy: The current solution $a_R^{t+1..H}$ is shaked to obtain a new candidate solution $b_R^{t+1..H}$, and if this solution leads to a better upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$, then it becomes the new current solution. Shaking is performed by the `shakeSolution` function. This function considers different neighborhoods, corresponding to the following move operators:

relocate, swap, inverted 2-opt, and cross-exchange (see [16,26] for complete descriptions). As explained in Section 5.2, depending on the chosen waiting and relocation strategy, additional move operators are exploited. At each call to the `shakeSolution` function, the considered move operator is changed, such that the operators are equally selected one after another in the list. Every β iterations, the pool S of scenarios is re-sampled (lines 9-10). This re-sampling introduces diversification as the upper bound computed by $\overline{\mathcal{Q}}$ changes. We therefore do not need any other meta-heuristic such as Simulated Annealing.

Function `handleRequest` is called at the beginning of a new time unit t, to compute action a^t in light of online requests (if any). It implements the GSA decision rule defined in eq. (6). The function considers each request revealed at time t for a vertex j, in a sequential way. For each request, it tries to insert it into the sequence $a_R^{t,H}$ (*i.e.*, modify the routes so that a vehicle visits j during its time window). As in `shakeSolution`, local search operations are performed during that computation. The time spent to find a feasible solution including the new request is limited to δ_{ins}. If such a feasible solution is found, then the request is accepted, otherwise it is rejected. If there are several online requests for the same discretized time t, we process these requests in their real-time order of arrival, and we assume that all requests are revealed at different real times.

5.2 Waiting and Relocation strategies

As defined in section 2, a vehicle that just visited a vertex usually has the choice between traveling right away to the next planned vertex or first waiting for some time at its current position. Unlike in the static (and deterministic) case, in the dynamic (and stochastic) VRPTW these choices may have a significant impact on the solution quality.

Waiting and relocation strategies have attracted a great interest on dynamic and stochastic VRP's. In this section, we present and describe how waiting and relocation strategies are integrated to our framework, including a new waiting strategy called *relocation-only*.

Relocation strategies Studies in [8,9] already showed that for a dynamic VRP with no stochastic information, it is optimal to relocate the vehicle(s) either to the center (in case of single-vehicle) or to strategical points (multiple-vehicle case) of the service region. The idea evolved and has been successfully adapted to routing problems with customer stochastic information, in reoptimization approaches as well as sampling approaches.

Relocation strategies explore solutions obtained when allowing a vehicle to move towards a customer vertex even if there is no request received for that vertex at the current time slice. Doing so, one recognizes the fact that, in the context of dynamic and stochastic vehicle routing, a higher level of anticipation can be obtained by considering to reposition the vehicle after having serviced a request to a more stochastically fruitful location. Such a relocation strategy has already been applied to the DS-VRPTW in [5].

Waiting strategies In a dynamic context, the planning of a vehicle usually contains more time than needed for traveling and servicing requests. When it finishes to service

a request, a vehicle has the choice between waiting for some time at its location or leaving for the next planned vertex. A good strategy for deciding where and how long to wait can potentially help at anticipating future requests and hence increase the dynamic performances. We consider three existing waiting strategies and introduce a new one:

- *Drive-First (DF)*: The basic strategy aims at leaving each serviced request as soon as possible, and possibly wait at the next vertex before servicing it if the vehicle arrives before its time window.
- *Wait-First (WF)*: Another classical waiting strategy consists in delaying as much as possible the service time of every planned requests, without violating their time windows. After having serviced a request, the vehicle hence waits as long as possible before moving to the next planned request.
- *Custom-Wait (CW)*: A more tailored waiting strategy aims at controlling the waiting time at each vertex, which becomes part of the online decisions.
- *Relocation-Only waiting (RO)*: In order to take maximum benefit of relocation strategy while avoiding the computational overhead due to additional decision variables involved in custom waiting, we introduce a new waiting strategy. It basically applies *drive-first* scheduling to every request and then applies *wait-first* waiting only to those requests that follow a relocation one. By doing so, a vehicle will try to arrive as soon as possible at a planned *relocation request*, and wait there as long as possible. In contrary, it will spend as less time as possible at non-relocation request vertices. Note that if it is not coupled to a relocation strategy, RO reduces to DF. Furthermore, RO also reduces to the dynamic waiting strategy described in [17] if we define the service zones as being delimited by relocation requests. However, our strategy differs by the fact that service zones in our approach are computed in light of stochastic information instead of geometrical considerations.

Depending on the waiting strategy we apply and whether we use relocation or not, additional LS move operators are exploited. Specifically, among the waiting strategies, only *custom-wait* requires additional move operators aiming at either increasing or decreasing the waiting time at a random planned vertex. *Relocation* also requires two additional move operators that modify a given solution by either inserting or removing a relocation action at a random vertex.

6 Experimentations

We now describe our experimentations and compare our results with those of the state of the art MSA algorithm of [7].

6.1 Algorithms

Different versions of Algorithm 3 have been experimentally assessed, depending on which waiting strategy is implemented and whether in addition we use the relocation strategy or not.

Surprisingly, the *wait-first* waiting strategy, as well as its version including *relocation*, produced very bad results in comparison to other strategies, rejecting more

than twice more online requests in average. Because of its computational overhead, the *custom-wait* strategy also produced bad results, even with relocation. For conciseness we therefore do not report these strategies in the result plots.

The 3 different versions of Algorithm 3 we thus consider are the following: GSA*df*, which stands for GSA with *drive-first* waiting strategy, GSA*dfr* which stands for GSA with *drive-first* and *relocation* strategies, and finally GSA*ro* with means GSA using *relocation-only* strategy. Recall that, by definition, the *relocation-only* strategy involves relocation. In addition to those 3 algorithms, as a baseline we consider the GLS*df* algorithm, which stands for *greedy local search* with *drive-first* waiting. This algorithm is similar to the dynamic LS described in [22], to which we coupled a Simulated Annealing metaheuristic. In this algorithm, stochastic information about future request is not taken into account and a neighboring solution is solely evaluated by its total travel cost.

Finally, GSA and GLS are compared to two MSA algorithms, namely MSA*d* and MSA*c* depending on whether the *travel distance* or the *consensus function* are used as ranking functions.

6.2 Benchmarks

The selected benchmarks are borrowed from [7] which considers a set of benchmarks initially designed for the static and deterministic VRPTW in [25], each of these containing 100 customers. In our stochastic and dynamic context, each customer becomes a request region, where dynamic requests can occur during the online execution.

The original problems from [7] are divided into 4 classes of 15 instances. Each class is characterized by its degree of dynamism (DOD, the ratio of the number of dynamic requests revealed at time $t > 0$ over the number of a priori request known at time $t = 0$) and whether the dynamic requests are known early or lately along the online execution. The time horizon $H = 480$ is divided into 3 time slices. A request is said to be early if it is revealed during the first time slice $t \in [1, 160]$. A late request is revealed during the second time slice $t \in [161, 320]$. There is no request revealed during the third time slice $t \in [321, 480]$, but the vehicles can use it to perform customer operations.

In Class 1 there are many initial requests, many early requests and very few late requests. Class 2 instances have many initial requests, very few early requests and some late requests. Class 3 is a mix of classes 1 and 2. In Class 4, there are few initial requests, few early requests and many late requests. Finally, classes 1, 2 and 3 have an average DOD of 44%, whilst Class 4 has an average DOD of 57%.

In [5], a fifth class is proposed with a higher DOD of 81% in average. Unfortunately, we were not able to get those Class 5 instances. We complete these classes by providing a sixth class of instance, with DOD of 100%. Each instance hence contains no initial request, an early request with probability 0.3 and a late request with probability 0.7.

Figure 2 summarizes the different instance classes.

6.3 Results

Computations are performed on a cluster composed of 32 64-bits AMD Opteron(tm) Processor 6284 SE cores, with CPU frequencies ranging from 1400 to 2600 MHz. Executables were developed with C++ and compiled on a Linux Red Hat environment

	DOD	$t = 0$	$t \in [1, 160]$	$t \in [161, 320]$	$t \in [321, 480]$
Class 1,2,3	44%	$P^0[i] = 0.5$	$P^{[1,160]}[i] = 0.25$	$P^{[161,320]}[i] = 0.25$	$P^{[321,480]}[i] = 0$
Class 4	57%	$P^0[i] = 0.2$	$P^{[1,160]}[i] = 0.2$	$P^{[161,320]}[i] = 0.6$	$P^{[321,480]}[i] = 0$
Class 5	81%	$P^0[i] = 0.1$	$P^{[1,160]}[i] = 0.1$	$P^{[161,320]}[i] = 0.8$	$P^{[321,480]}[i] = 0$
Class 6	100%	$P^0[i] = 0$	$P^{[1,160]}[i] = 0.3$	$P^{[161,320]}[i] = 0.7$	$P^{[321,480]}[i] = 0$

Fig. 2. Summary of the test instances, grouped per degree of dynamism. $P^{[t,t']}[i]$ represents the probability that a request gets revealed during the time slice defined by interval $[t, t']$.

with GCC 4.4.7. Average results over 10 runs are reported. In [7], 25 minutes of offline computation are allocated to MSA, in order to decide the first online action at time $t = 1$. During online execution, each time unit within the time horizon was executed during 7.5 seconds by the simulation framework. In order to compensate the technology difference, we decided in this study to allow only 10 minutes of offline computation and 4 seconds of online computation per time unit. Thereafter, in order to highlight the contribution of the offline computation in our approach, the amount of time allowed at pre-computation is increased to 60 minutes, while each time unit still lasts 4 seconds. According to preliminary experiments, both the size of the scenario pool and the resampling rate are set to $\alpha = \beta = 150$ for all our algorithms except GLS*df*.

Figure 3 gives a graphical representation of our algorithms results, through performance profiles. Performance profiles provide, for each algorithm, a cumulative distribution of its performance compared to other algorithms. For a given algorithm, a point (x, y) on its curve means that, in $(100 \cdot y)\%$ of the instances, this algorithm performed at most x times worse than the best algorithm on each instance taken separately. Instances are grouped by DOD and by offline computation time. Classes 1, 2 and 3 have a DOD of 44%, hence they are grouped together. An algorithm is strictly better than another one if its curve stays above the other algorithm's curve. For example on the 60min plot of Class 6, GLS*df* is the worst algorithm in 95% of Class 6 instances, outperforming GSA*df* in the remaining 5% (but not the other algorithms). On the other hand, provided these 60 minutes of offline computation, GSA*ro* obtains the best results in 55% of the instances, whereas only 30% for GSA*df* and GSA*dfr*. See [11] for a complete description of performance profiles. Detailed results are provided in the extended version [30].

Our algorithms compare fairly with MSA, especially on lately dynamic instances of Class 4. Given more offline computation, our algorithms get stronger, although that MSA benefits of the same offline time in every plots. Surprisingly, GLS*df* performs well compared to other algorithms on classes 1,2 and 3. The low DOD that characterizes these instances tends to lower the contribution of stochastic knowledge against the computational power of GLS*df*. Indeed, approximating the stochastic evaluation function over 150 scenarios is about 10^3 times more expensive than GLS*df* evaluation function. However, as the offline computation time and the DOD increase, stochastic algorithms tend to outperform their deterministic counterpart.

We notice that the relocation strategy gets stronger as the offline computation time increases. This is due to the computational overhead induced by relocation vertices. GSA*df* is then the good choice under limited offline computation time. However, both

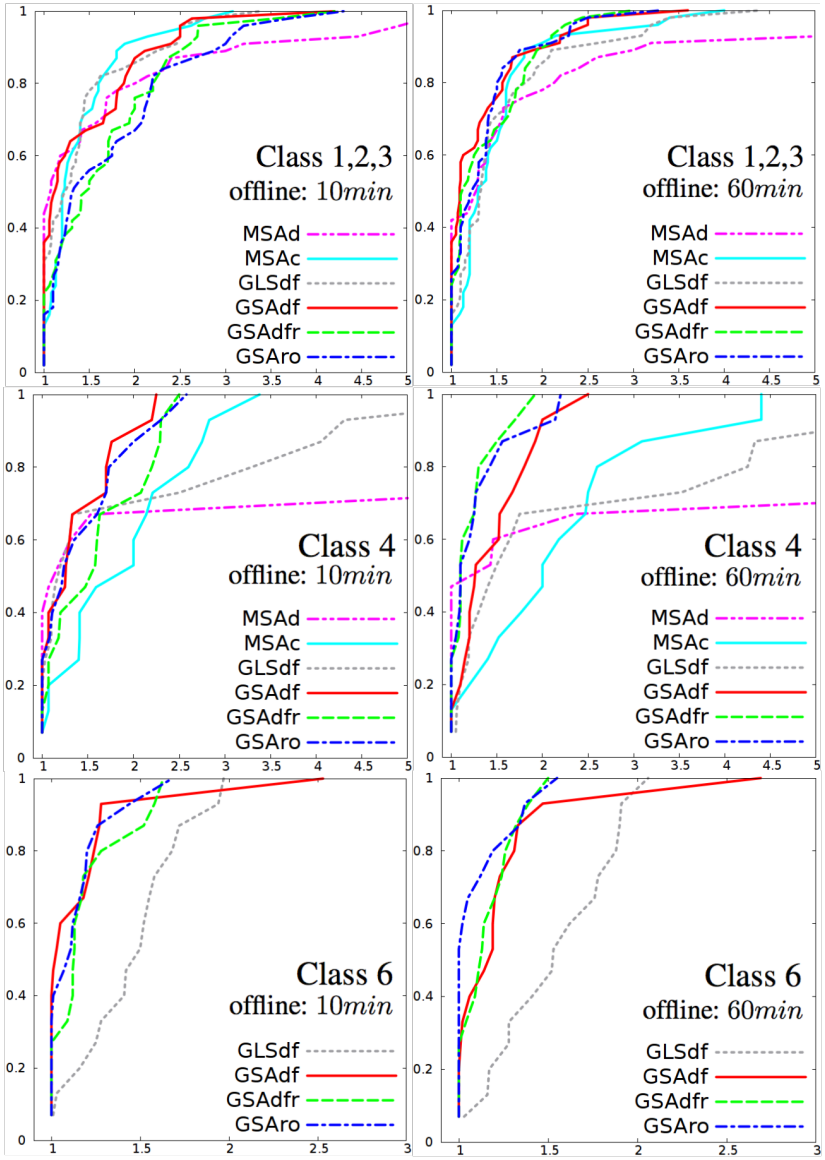


Fig. 3. Performance profiles on classes [1, 2 ,3], Class 4 and Class 6 problem instances

GSA*ro* and GSA*dfr* tend to outperform the other strategies when provided enough offline computation and high DOD.

As it contains no deterministic request, in Class 6 the offline computation is not applicable to those algorithms that does not exploit the relocation strategy, i.e. GLS*df* and GSA*df*. Class 6 shows that, despite the huge difference in the number of iterations performed by GLS*df* on one hand and stochastic algorithms on the other, the laters clearly outperform GLS*df* under fully dynamic instances. We also notice in this highly dynamic context that GSA*ro* tends to outperform GSA*dfr* as offline computation increases, highlighting the anticipative contribution provided by the *relocation-only* strategy, centering waiting times on relocation vertices.

7 Conclusions

We proposed GSA, a decision rule for dynamic and stochastic vehicle routing with time windows (DS-VRPTW), based on a stochastic programming heuristic approach. Existing related studies, such as MSA, simplify the problem as a two-stage problem by using sample average approximation. In contrary, the theoretical singularity of our method is to approximate a multistage stochastic problem through Monte Carlo sampling, using a heuristic evaluation function that preserves the nonanticipativity constraints. By maintaining one unique anticipative solution designed to be as flexible as possible according to a set of scenarios, our method differs in practice from MSA which computes as many solutions as scenarios, each being specialized for its associated scenario. Experimental results show that GSA produces competitive results with respect to state-of-the-art. This paper also proposes a new waiting strategy, *relocation-only*, aiming at taking full benefit of relocation strategy.

In a future study we plan to address a limitation of our solving algorithm which embeds GSA, namely the computational cost of its evaluation function. One possible direction would be to take more benefit of each evaluation, by spending much more computational effort in constructing neighboring solutions, e.g. by using Large Neighborhood Search [24]. Minimizing the operational cost, such as the total travel distance, is usually also important in stochastic VRPs. Studying the aftereffect when incorporating it as a second objective should be of worth. It is also necessary to consider other types of DS-VRPTW instances, such as problem sets closer to public or good transportation. Finally, the conclusions we made in section 2 about the shortcoming of a two-stage formulation (showed in Fig. 1) are theoretical only, and should be experimentally assessed.

Acknowledgement

Christine Solnon is supported by the LABEX IMU (ANR-10-LABX-0088) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR). This research is also partially supported by the UCLouvain Action de Recherche Concertée ICTM22C1.

References

1. Shabbir Ahmed and Alexander Shapiro. The sample average approximation method for stochastic programs with integer recourse. *Submitted for publication*, 2002.
2. Sø ren Asmussen and Peter W Glynn. *Stochastic Simulation: Algorithms and Analysis: Algorithms and Analysis*, volume 57. Springer, 2007.
3. Russell Bent and Pascal Van Hentenryck. Regrets only! online stochastic optimization under time constraints. *AAAI*, pages 501–506, 2004.
4. Russell Bent and Pascal Van Hentenryck. The Value of Consensus in Online Stochastic Scheduling. *ICAPS*, (1):219–226, 2004.
5. Russell Bent and Pascal Van Hentenryck. Waiting and Relocation Strategies in Online Stochastic Vehicle Routing. *IJCAI*, pages 1816–1821, 2007.
6. Russell Bent, Irit Katriel, and Pascal Van Hentenryck. Sub-optimality approximations. *Principles and Practice of Constraint . . .*, pages 1–15, 2005.
7. Russell W Bent and Pascal Van Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6):977–987, 2004.
8. DJ Bertsimas and G Van Ryzin. A stochastic and dynamic vehicle routing problem in the Euclidean plane. *Operations Research*, 1991.
9. DJ Bertsimas and G Van Ryzin. Stochastic and Dynamic Vehicle Routing in the Euclidean Plane with Multiple Capacitated Vehicles. *Operations Research*, 1993.
10. Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *4OR: A Quarterly Journal of Operations Research*, 1(2):89–101, 2003.
11. Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.
12. Truls Flatberg, Geir Hasle, Oddvar Kloster, Eivind J Nilssen, and Atle Riise. Dynamic and stochastic vehicle routing in practice. In *Dynamic Fleet Management*, pages 41–63. Springer, 2007.
13. Gianpaolo Ghiani, Emanuele Manni, Antonella Quaranta, and Chefi Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96–106, January 2009.
14. Lars M. Hvattum, Arne Lø kketangen, and Gilbert Laporte. Solving a Dynamic and Stochastic Vehicle Routing Problem with a Sample Scenario Hedging Heuristic. *Transportation Science*, 40(4):421–438, November 2006.
15. Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Exploiting Knowledge About Future Demands for Real-Time Vehicle Dispatching. *Transportation Science*, 40(2):211–225, May 2006.
16. Gerard A P Kindervater and Martin W P Savelsbergh. Vehicle routing: handling edge exchanges. *Local search in combinatorial optimization*, pages 337–360, 1997.
17. Snežana Mitrović-Minić and Gilbert Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, August 2004.
18. Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.
19. Victor Pillac, Christelle Guéret, and Andrés L. Medaglia. An event-driven optimization framework for dynamic vehicle routing. *Decision Support Systems*, 54(1):414–423, December 2012.
20. Harilaos N Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

21. Harilaos N Psaraftis. Dynamic vehicle routing: Status and prospects. *annals of Operations Research*, 61(1):143–164, 1995.
22. M Schilde, K F Doerner, and R F Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & operations research*, 38(12):1719–1730, December 2011.
23. Alexander Shapiro, Darinka Dentcheva, and Andrzej P Ruszczy\'nski. *Lectures on stochastic programming: modeling and theory*, volume 9. SIAM, 2009.
24. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998.
25. MM Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2), 1987.
26. É Taillard and P Badeau. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation ...*, pages 1–36, 1997.
27. Pascal Van Hentenryck, Russell Bent, and Eli Upfal. *Online stochastic optimization under time constraints*, volume 177. September 2009.
28. Bram Verweij, Shabbir Ahmed, Anton J Kleywegt, George Nemhauser, and Alexander Shapiro. The sample average approximation method applied to stochastic routing problems: a computational study. *Computational Optimization and Applications*, 24(2-3):289–333, 2003.
29. Nigel H M Wilson and Neil J Colvin. *Computer control of the Rochester dial-a-ride system*. Massachusetts Institute of Technology, Center for Transportation Studies, 1977.
30. Michael Saint-Guillain, Yves Deville, and Christine Solnon A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW - Extended version. arXiv:1502.01972 [cs.AI], 2015.