CIRRELT
1971 – 2021

CIRRELT-2021-17

Dynamic Optimization Algorithms for Same-Day Delivery Problems

Jean-François Côté
Thiago Alves de Queiroz
Francesco Gallesi
Manuel Iori

April 2021

Document de travail également publié par la Faculté des sciences de l'administration de l'Université Laval, sous le numéro FSA-2021-004

Bureau de Montréal
Université de Montréal
C.P. 6128, succ. Centre-Ville
Montréal (Québec) H3C 3J7
Tél : 1-514-343-7575
Télécopie : 1-514-343-7121

Bureau de Québec
Université Laval,
2325, rue de la Terrasse
Pavillon Palasis-Prince, local 2415
Québec (Québec) G1V 0A6
Tél : 1-418-656-2073
Télécopie : 1-418-656-2624

Dynamic Optimization Algorithms for Same-Day Delivery Problems

Jean-François Côté[1,2,*], Thiago Alves de Queiroz[2], Francesco Gallesi[3], Manuel Iori[1,4]

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
2. Department of Operations and Decision Systems, Université Laval, Québec, Canada
3. Institute of Mathematics and Technology, Federal University of Catalão, 75704-020, Catalão, Brazil
4. Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, 42122, Reggio Emilia, Italy

Abstract. We study a dynamic vehicle routing problem where stochastic customers request urgent deliveries characterized by restricted time windows. The aim is to use a fleet of vehicles to maximize the number of served requests and minimize the traveled distance. This problem is known in the literature as the same-day delivery problem. It is of high importance because it models a number of real-world applications, including the delivery of online purchases. We solve the same-day delivery problem by proposing efficient solution algorithms, ranging from a simple reoptimization heuristic to a sophisticated branch-and-regret heuristic in which sampled scenarios are used to anticipate future events. All algorithms adopt a tailored adaptive large neighborhood search to optimize the routing plans. We also present two new consensus functions to select routing plans for implementation, and propose strategies for determining the number and size of the sampled scenarios. The algorithms are also adapted to solve the problem variant where vehicles are allowed to perform preemptive returns to the depot. Extensive computational experiments on a large variety of instances prove the outstanding performance of the proposed algorithms, also in comparison with recent literature, in terms of served requests, traveled distance, and computing time.

Keywords: Same-day delivery, dynamic pickup and delivery, branch-and-regret, consensus function, preemptive depot return.

Acknowledgements. Jean-François Côté acknowledges support by the Natural Sciences and Engineering Council of Canada (NSERC) under grant 2015-04893. Thiago Alves de Queiroz acknowledges support by the National Council for Scientific and Technological Development (CNPq) under grant 311185/2020-7 and the State of Goiás Research Foundation (FAPEG). Manuel Iori acknowledges support from the University of Modena and Reggio Emilia under grant FAR 2018. We thank Compute Canada for providing high-performance computing facilities.

---

* Corresponding author: Jean-Francois.Cote@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2021

1 Introduction

In recent years, online purchases have become more and more a common practice to request services or goods at home, changing the habit in which many traditional markets operate. A huge number of e-commerce sellers appeared on the web, and many companies changed their focus to direct-to-consumer deliveries to expand their business. Dayarian et al. (2020) mentioned that online shopping followed by home delivery has annually increased by around 8.5% in mature markets (e.g., the United States) and close to 300% in developing markets (e.g., India). This trend has increased by the Covid-19 pandemic, which made people reluctant to leave their homes.

As a consequence, the delivery of online purchases has become a crucial logistic activity. From an Operations Research perspective, managing efficiently and effectively this activity is not an easy task. Requests arrive dynamically during the day and must be served within predetermined time windows. Although previous information could be collected, it might be hard to respond timely to peaks of requests at certain hours of the day. Optimization plays thus an important role in achieving affordable logistic costs.

Historically, this type of problem has been addressed in the field of *stochastic dynamic vehicle routing problems* (SDVRPs), for which a broad and wide literature is known (e.g., Pillac et al. 2013 and Ritzinger et al. 2016). In recent years, the interest in such problems has risen even more, and a particular effort has been put on applications where requests have very strict time windows, of typically one hour or less. Such applications appear in dedicated online services for purchase and delivery of parcels (e.g., Amazon Prime Now) or meals (e.g., Uber Eats). Innovative optimization techniques have been consequently developed to tackle these challenging problems (e.g., Ulmer 2020).

In this context, Voccia et al. (2019) introduced a particular SDVRP, the *same-day delivery problem* (SDDP), in which requests from a limited geographical area arrive dynamically during the day and each must be served within a strict time window. All goods are based at a central depot where a fleet of identical vehicles is available to perform the deliveries. Requests that the fleet cannot handle are passed to a *third-party logistic operator* (3PL). The aim is to minimize the number of unserved requests, supposing that previous stochastic information on the arrival process is available and can be used to help decision at the very moment in which a request is issued. Formally speaking, the SDDP of Voccia et al. (2019) corresponds to a dynamic vehicle routing problem with stochastic customers and mul-

tiple delivery routes per vehicle. Clearly, the problem models a variety of applications, including the distribution of online purchases.

The SDDP shares with other SDVRPs a number of difficult questions that a decision-maker should consider when planning the distribution service, for instance: How many vehicles do we need to perform the requests efficiently? How can we efficiently route the vehicles? Should we wait for new requests, or should we start delivering as soon as possible? Can we make use of information from the past to devise better routing plans? Should we dynamically reroute the vehicles when new requests arrive, asking for anticipated returns to the depot? How can we effectively estimate the cost of a routing plan, given the limited information we have?

In this paper, we propose solution approaches that can help a decision-maker to obtain proper answers to the above questions in the context of the SDDP. We investigate the problem of minimizing the number of rejected requests and, as a secondary objective, the total routing cost. Assuming hard time window constraints, requests impossible to deliver on time are assigned to the 3PL. We do not allow transshipments among vehicles, but allow vehicles to wait at the depot to anticipate future requests.

In detail, our contributions consist of:

- a new way to represent the SDDP as a pickup-and-delivery problem with time windows and release dates, which brings advantages when considering preemptive vehicle returns to the depot and when planning efficient routes containing real and fictive requests from sampled scenarios;
- different solution algorithms, namely:
  - a simple but efficient reoptimization heuristic that locks past decisions, adds newly revealed requests, and then optimizes the routing plan with a function that combines a regret heuristic, local searches, and an *adaptive large neighborhood search* (ALNS);
  - a *scenario-based planning approach* (SBPA) that considers fictive requests sampled by a probability distribution. In each scenario, a routing plan is optimized with the ALNS, and a *consensus function* chooses which plan to adopt after considering all scenarios. We also propose new consensus functions and show that one of them has a better performance than a previous function in the literature;
  - a *branch-and-regret* (B&R) heuristic that extends the scenario-based planning heuristic by evaluating different possible alter-

natives for each request. We consider as alternatives a vehicle departing now, a vehicle departing in the future, and the assignment of a request to the 3PL. We create different routing plans by branching on these alternatives. The plan to implement is chosen with the help of a consensus function;

- the evaluation of the important problem variant in which the *preemptive depot return* (PDR) of the vehicles to the depot is allowed. In the PDR variant, vehicles are allowed to return to the depot after serving a request even if they still have items on board demanded by successive requests. This possibly allows to collect further requests and optimize the route the vehicle will perform after departing again from the depot.

All solution approaches have been tested by means of extensive computational experiments. The simple reoptimization heuristic is efficient when the number of vehicles is small compared to the number of requests, and thus the best option is usually to start delivering as soon as possible. Instead, the SBPA is more efficient when there is more room for optimization, so waiting or rerouting can lead to a cost reduction. This is obtained at the expense of a non-negligible computational effort. The B&R may lead to further improvements with respect to SBPA, and even to a reduction in the required computing time.

Further computational experiments have analyzed the number of ALNS iterations, the number of sampled scenarios, and the length of the time horizon where to sample fictive requests. The gain obtained by allowing PDRs is discussed in detail, showing how this practice can benefit the problem. Besides that, our algorithms consistently improve the previous ones from both Ulmer et al. (2019) and Voccia et al. (2019). Thus, we can conclude that they represent an important contribution in the crucial field of dynamic routing of urgent deliveries.

The remainder of the paper is organized as follows. In Section 2, we discuss the literature on the SDDP and related problems, pointing out the main previous contributions and the similarities/differences from our work. Section 3 contains a formal description of the SDDP. In Section 4, we present the details of our solution approaches and show how they can be adapted to deal with different problem variants. Section 5 is devoted to the computational study, with detailed results for diverse sets of parameter configurations and problem instances. In Section 6, we give some concluding remarks and point toward future, promising research directions.

2 Literature Review

This section presents a literature review focused on SDDPs arising in the context of e-commerce. For a more exhaustive review on SDVRPs, we refer to Pillac et al. (2013), for a general overview, to Ritzinger et al. (2016), for a survey and a comparison of the solution quality obtained by a broad range of methods from the literature, and to Ulmer et al. (2020), who present the different techniques used to solve SDVRPs.

The SDDP shares characteristics with several well-known problems in the literature. Typically, vehicles perform several short trips from the depot to the customers when capacity or time is limited. This feature appears in the *multi-trip vehicle routing problem* (MTVRP) and was introduced by Fleischmann (1990), who propose a savings algorithm to build routes and a bin packing heuristic to combine the built routes into work shifts. Several other works, including Battarra et al. (2009), Azi et al. (2014) and Cattaruzza et al. (2014), propose more complex heuristics. Exact algorithms based on column generation are proposed by Azi et al. (2010), Mingozzi et al. (2013) and Paradiso et al. (2020). A survey on MTVRPs can be found in Cattaruzza et al. (2018).

Another relevant SDDP characteristic is related to the *release date* of the requests. This represents the moment in which the requested merchandise becomes available for delivery. This implies that a request can only be part of a route that departs later than the request release date. Vehicles can then perform new routes by returning to the depot when new requests become available. This is particularly relevant in problems where requests are not all known at the beginning of the time horizon. This characteristic is studied by Arda et al. (2014) in a production and transportation problem. In this study, requests are produced on a make-to-order basis. Some requests are known, and their release dates can be obtained by considering the production process. Some other requests become known only when the customers have confirmed them, leading to an uncertain release date. Stochastic information on these dates is used within several heuristics to solve a stochastic optimization model over a rolling horizon. Cattaruzza et al. (2016) consider the case where a warehouse is receiving loads of merchandise by trucks all day long. Once a load has been prepared for shipping, a route can be planned to perform the deliveries. The authors propose a genetic algorithm and solve instances with up to 100 customers.

The earliest studied applications of SDDPs can be found in the domain of e-groceries. Lin and Mahmassani (2002) provides answers to several matters that are still studied today: impact of different time window sizes, the effect

of city configuration, and delivering from either a centralized warehouse or from several grocery stores. They find out that narrow time windows are economically viable when the demand is high as this minimizes the idle time of the vehicles. A low demand should be coupled with larger time windows. In addition, a bricks-and-clicks strategy offers a less risky alternative to a single warehouse strategy. Their study diverges from our application as customers must order before a cut-off time. Once the cut-off time has passed, vehicles leave the depot to perform their routes, one per vehicle.

For the delivery of short life span products, Azi et al. (2012) consider the case where customers are not known at the beginning of the day but are gradually revealed as time goes on. These products cannot stay on board the vehicles for longer than a specific time. This imposes the vehicles to perform several short routes, instead of only a single long one, from the depot to the customers. When a new event is triggered, the current known information is used to plan new routes that serve the highest revenue requests.

Voccia et al. (2019) study the SDDP in the context of online purchases where all requests arrive dynamically over the course of the day. A vehicle fleet is available to pick up the goods from the depot and deliver them to the customers. Requests are typically associated with short time windows, and the objective is to maximize the number of served requests regardless of the distance traveled by the fleet. They model the problem as a Markov decision process and propose an SBPA to obtain a route plan at each decision epoch. The approach uses sampled scenarios of future requests to build a set of route plans. A *consensus function* is used to select the route plan that shares the most characteristics with the others. Such an approach was originally proposed by Bent and Van Hentenryck (2004) for the dynamic vehicle routing problem with time windows, where the objective was to maximize the number of served customers. Results indicate that more customer requests can be delivered using sampled scenarios than using a simple reoptimization heuristic.

The *dispatch wave problem* by Klapp et al. (2018) shares several characteristics with the SDDP of Voccia et al. (2019). In this problem, requests do not have time windows, and a single vehicle is available to perform the deliveries. The vehicle can depart from the depot only at specific times, called *waves*, and must be back at the depot before a specific deadline. Klapp et al. (2018) propose two different approaches to obtain a priori solutions. In an a priori solution, routes are built in a first stage without knowing all the information. Then, in a second stage, they are updated whenever new information is revealed by means of a recourse policy. The first approach is based on the solution of a stochastic model using a heuristic. The second

approach generates new a priori solutions on a rolling horizon fashion: a solution is obtained at the beginning of the time horizon, then, each time the vehicle returns to the depot, a new solution is computed. Results on instances with up to 50 requests indicate that the second approach can reduce costs by an average of 9% over the first approach.

Archetti et al. (2020) study the dynamic traveling salesman problem with stochastic release dates, which is very similar to the dispatch wave problem. It differs from it because the time horizon is not bounded and the objective is to minimize the total travel time plus the waiting time at the depot. As noted by the authors, this type of problem might be encountered in situations where goods need to be shipped to the depot and the transportation time might be affected by traffic or other unforeseen events. Their approaches are similar to those by Klapp et al. (2018), and the best results can be achieved by using a rolling horizon heuristic. Results indicate that reoptimizing at short intervals while the vehicle is waiting is beneficial for reducing the objective function.

An important issue in the SDDP is to decide whether a vehicle should be waiting at the depot for new incoming requests or should depart as soon as possible to perform deliveries and return to the depot at an earlier time to accommodate more requests later on. Waiting strategies play a crucial role in SDVRPs as they can help serve more customers and reduce the traveled distances. Some strategies only use the information known at the time the decision is taken, whereas others use stochastic information to try to predict new incoming requests. Mitrović-Minić and Laporte (2004) provide four waiting strategies that do not use any knowledge about future events. The first two are opposite strategies: the *Drive-First* is a no-wait strategy where the vehicle departs as soon as possible, whereas the *Wait-First* imposes to wait whenever possible. Routes obtained from a deterministic approach that does not make use of a waiting strategy are typically those obtained by applying the Drive-First strategy, like those in Azi et al. (2012). The other two strategies from Mitrović-Minić and Laporte (2004) are in-between the previous two, and attempt to assign an amount of waiting time to some key moments of the routes. Results indicate that waiting typically produces shorter routes but increases the number of used vehicles, whereas the no-wait strategy results in the use of fewer vehicles but at an increase in traveled distances.

Waiting strategies are also found in Voccia et al. (2019). In their Wait-First strategy, the amount of waiting time is calculated as the maximum delay that can be added to the routes that are about to start while respecting the time window constraints. A second strategy is somehow hidden in the

details of the SBPA. Indeed, as noted by Bent and Van Hentenryck (2007), the SBPA has an implicit waiting strategy when real requests are assigned to a route that also contains fictive requests. In the context of the SDDP, the fictive requests indicate that new requests might arrive in the near future and the vehicle should be waiting for some time before delivering the real ones. This means that some vehicles might be waiting at the depot for new requests, while others apply a Drive-First strategy and depart as soon as possible to perform deliveries. Results in Voccia et al. (2019) indicate that the SBPA with the Wait-First strategy requires 2.8 times more computation time than the SBPA without it, and it cannot serve more requests.

In Bent and Van Hentenryck (2007), an SBPA is used to analyze the solutions of the sampled scenarios and decide whether or not a vehicle should wait at its current location for possible new requests. This is achieved by counting the number of times a vehicle has as next visit either a real or a fictive request in the solutions of the sampled scenarios. The vehicle waits if the resulting number is higher for the fictive requests. Results show that such a strategy was helpful at maximizing the number of served customers.

In our approach, at a certain epoch, we firstly optimize scenarios using a heuristic, and secondly optimize again with the same heuristic but forcing a waiting time for all the vehicles that are at the depot. If the cost of the second attempted option is smaller or equivalent to that of the first option, then we simply wait for the next event. This has the advantage of producing a much quicker computation. Otherwise, we proceed in a B&R fashion: we consider each request in the pool, evaluate if it has to be served now, served later, or rejected. For each alternative, the evaluation is obtained by building a solution with a heuristic. This is performed for all scenarios and all alternatives, and then the alternative giving the lowest average cost over all scenarios is chosen and implemented. This scheme has the advantage of being adaptable to different problems.

Another feature of SDDPs concerns allowing or not vehicles to perform PDRs. A PDR implies that a vehicle interrupts its current route and returns to the depot to pick up new requests. This might allow serving more requests and reducing traveled distances. The PDRs are not allowed in Azi et al. (2012), nor in Voccia et al. (2019). The idea was introduced by Ulmer et al. (2019), who exploit it inside an algorithm based on approximate dynamic programming for the solution of single-vehicle instances. Results show that PDRs can effectively allow more requests to be served. This contrasts Klapp et al. (2018), who obtain small marginal benefits by using PDRs in their computational experience.

In some applications, it might be required to decide immediately if a new

request is accepted and delivered or if it is rejected. This requirement can be useful when an amount of work has to be performed on the request before it can be available for shipment. Thus, rejecting the request at a later stage might cause additional unnecessary costs. This is defined *request acceptance policy* in Klapp et al. (2020) or *customer acceptance problem* in Marlin and Thomas (2020). To our knowledge, this policy was first considered in Azi et al. (2012). In this work, to decide whether to accept a request or, the authors first check if the request can be feasibly inserted in the current routes. If no feasible insertion position is found, then the request is rejected; otherwise, a lookahead algorithm, similar to an SBPA, is executed to anticipate future requests. The lookahead algorithm evaluates the cost of performing the delivery or not on a set of sampled scenarios. The least-cost alternative is selected. If the request is accepted, a new routing plan is obtained by reoptimizing the set of accepted known customers. This request acceptance policy is also studied in detail in Klapp et al. (2020) for the dispatch wave problem. They use approaches similar to those of Klapp et al. (2018) and calculate that an immediate request acceptance policy increases costs by an average of 4.5%. In Marlin and Thomas (2020), they model the SDDP as a Markov Decision Process and rely on approximate dynamic programming and several value function approximation methods for solving the problem.

Our approaches for solving the SDDP rely on sampling scenarios that incorporate stochastic knowledge of future events. These approaches have the advantage of being flexible and can solve a broad range of problems with different characteristics. They also scale relatively well when the number of customers and vehicles increases compared to approaches that rely on Markov Decision Processes (which typically solve small instances having few requests and one vehicle). They work by sampling the probability distribution of future events. Sampled scenarios are generated each time a new event occurs and an optimization phase is executed next to obtain an implementable routing plan. This consists of finding a plan that can be implemented in all scenarios, leading to low-cost solutions. Typically, each scenario is solved separately to alleviate the computational complexity. Unfortunately, each routing plan is specifically tailored for its scenario and implementing one of the scenario solutions does not ensure achieving a low-cost solution in the other scenarios. The literature has come up with different approaches for addressing this problem. They can be seen as partial explorations of a branch-and-bound tree for a stochastic integer program (see Haneveld and van der Vlerk 1999). Scenarios are solved at each node of the tree to fix some first-stage decisions. For example, the SBPA proposed

by Bent and Van Hentenryck (2004) solves each scenario and selects the plan having the most parts in common with other plans by using a consensus function. This can be interpreted as solving only the root node of the tree.

Another strategy brought up by Løkketangen and Woodruff (1996), called *progressive hedging heuristic*, and later used by Hvattum et al. (2006) among others, is to solve all scenarios and then find the alternative a that is the most common alternative among the alternatives that are not performed in all the resulting solutions. This alternative is then fixed and plans, not having it, are optimized. The process iterates until all plans implement the same set of alternatives. Again, this can be seen as solving a node in a branch-and-bound tree, selecting a variable, and creating a single child node.

Hvattum et al. (2007) propose the B&R heuristic as an improvement of the progressive hedging heuristic that consists of evaluating the cost of performing or not alternative a. The least-cost alternative, a or not a, is then fixed, and the algorithm iterates until all plans implement the same alternatives. The results in Hvattum et al. (2007) indicate that the B&R heuristic is superior in terms of solution quality to the progressive hedging heuristic. The branch-and-bound tree, in this case, is partially explored: a node is solved, a variable is chosen for branching, and child nodes are generated and solved. The exploration continues by adopting the least-cost child node until a solution is reached. This is the most complex approach that we adopt in our solution methods, as outlined below in Section 4.

3 Problem Definition

The SDDP considers a complete directed graph $G = (L_0, A)$, where L_0 comprises a depot, vertex 0, and a set L of customer locations distributed over a geographical area. The depot is equipped with a fleet of M identical vehicles and is associated with start ($t = 0$) and end ($t = T$) times between which vehicles can depart and arrive. The time interval $[0, T]$ corresponds to the working hours of the depot. With each arc $(i, j) \in A$ are associated a deterministic travel time t_{ij} and a traveling distance c_{ij}, which are known in advance. During the time horizon, requests arrive at a rate $\lambda_i \geq 0$ from each location $i \in L$.

Let R be the set of requests that occur during the daily time horizon. Set R is composed of requests that are known in advance (from before the starting time of the operations) and others that will be revealed as time

unfolds. Each request $k \in R$ is revealed at a release time r_k and a delivery time window $[e_k, l_k]$. Each request should be picked at the central depot and delivered by a vehicle. In case a vehicle arrives at a customer location for delivering request k before the start time e_k, it waits until e_k to start the service. The service must begin before l_k, so we consider *hard time window* constraints. Requests found impossible to deliver on time are assigned to a 3PL operator paying an additional cost. We assume that the delivery costs incurred by the fleet for performing a request are always lower than the cost of the 3PL operator.

Each vehicle may perform multiple trips during the time horizon, starting at any time $t \geq 0$ and finishing at any time $t \leq T$. The trips performed by the vehicle may involve the following actions:

(i) wait at the depot for new requests;

(ii) pick up at the depot one or more requests; and

(iii) deliver one or more requests to customers.

Once a vehicle departs from a location, it cannot divert its path until it has reached its next location. After a delivery, a vehicle can continue its trip as planned or interrupt the trip and return to the depot. This means the vehicle is not required to finish serving all its onboard requests before returning to the depot. In other words, we allow *preemptive returns* to the depot. Once a vehicle has picked up a request, it must serve it, so it cannot unload any request at the depot. In other words, we forbid *transshipments* among vehicles. When using the 3PL operator for delivering a request, we simply assume we pay a high cost and we do not explicitly model the 3PL operator's routes. The decision of assigning a request to the 3PL operator is postponed until we detect that the fleet will not be able to deliver such a request.

The objective of the SDDP is to determine the trips performed by the vehicles during the time horizon, aiming first at maximizing the number of served requests and secondly at minimizing the total traveled distance.

4 Problem Modeling and Solution

We model the SDDP as a *dynamic pickup and delivery problem with time windows and release dates* (DPDP). Under this representation, each request $k \in R$ corresponds to a pair of nodes (i, j), where i is the pickup node and j the delivery node. In our case, i is always coincident with the depot, whereas

j is associated with a customer location in L. The time window $[e_i, l_i]$ of the pickup node i is set to $[r_k, l_k - t_{0k}]$, whereas that of the delivery node j is set to $[e_j, l_j] = [e_k, l_k]$, for each $k \in R$.

The use of a DPDP representation of the problem gives some advantages. First, it is easy to model a preemptive return to the depot, as one simply needs to insert a pickup in the middle of two deliveries. Second, the constraint forbidding the unloading of already picked up requests at the depot is naturally taken into account by the classical DPDP constraint that imposes a delivery node to be visited by the same vehicle that visited the corresponding pickup node. Several authors, as Voccia et al. (2019) and Archetti et al. (2020), modeled each request of the SDDP as a single delivery node located at a customer location. This has the advantage of being faster to optimize, but it is less flexible and might remove some sequencing possibilities. For example, in Voccia et al. (2019) if a fictive delivery (originated when modeling the stochastic component of the problem) is visited by a vehicle right after a real delivery, then the vehicle is sent back to the depot. This makes routes containing real and fictive customers difficult to plan efficiently. Such routes indicate that it is beneficial to wait for incoming requests to regroup the deliveries. To counter this side effect, Voccia et al. (2019) proceed with a two-steps approach: first, they optimize the routes, and then they employ a mechanism that considers waiting at the depot. In some ways, they do not fully utilize the information contained in the scenarios.

We tackle the dynamic aspect of the SDDP in the classical SDVRP approach (see, e.g., Gendreau et al. 1999): each time a new event occurs, all the known information is gathered together, and an optimization step is executed. This step requires making several decisions (e.g., if a vehicle departs from the depot or another performs a preemptive return). This classical approach may change consistently according to the way events and optimization are taken into account. During the working period, an *event* occurs each time new information becomes known, or new decision has to be taken. In our work, we consider three types of events:

- arrival of a new request when there is at least one vehicle available at the depot;
- arrival of a vehicle at the depot or completion of the waiting period of a vehicle (still at the depot); and
- completion of a delivery by a vehicle that is supposed to continue its route (and not return to the depot).

The first two types of events are the only ones considered in Voccia et al. (2019). The third type is useful for considering preemptive returns to the depot.

When an event occurs, all known information is collected (known requests, the position of the vehicles, goods that are on board the vehicles, etc.), and an optimization algorithm is invoked to take the next routing decisions. To this aim, in our work we developed three solution algorithms, described next in Sections 4.1, 4.2 and 4.3. The subsequent sections are instead devoted to discuss the way the scenarios are generated, the ALNS algorithm that we developed to optimize the routing aspect and that is invoked at each event, and the steps to be performed to adapt the solution algorithms to deal with the main problem variants.

4.1 Reoptimization Heuristic

The *reoptimization heuristic* is a dynamic algorithm that ignores the stochastic aspects of the problem. It works as shown in Algorithm 1. First, an empty *routing plan* s is created, and all known requests at the beginning of the time horizon are added to s. A routing plan, or just *plan* for short in the following, is a partial solution to the dynamic problem, which might be later modified/integrated according to new information revealed. Second, s is optimized, meaning that routes are generated using the ALNS of Section 4.5. Then, each time there is a new event, all actions that were performed prior to the current time and all those being currently performed are locked in plan s. The possible newly revealed requests are added to s, and another call to the optimization algorithm is performed to adjust plan s to cope with the new information. Once all events have been considered, the algorithm terminates with a final plan containing all routing actions.

The term reoptimization heuristic that we adopt originates from the recent survey by Ulmer et al. (2020), but the algorithm is also known as *greedy heuristic* (see, e.g., Bent and Van Hentenryck 2004) or *myopic heuristic* (see, e.g., Hvattum et al. 2006, 2007 and Archetti et al. 2020). As can be guessed by the names, this algorithm can be quite simple, although this is not the case for our implementation, as we rely on a tailored metaheuristic at steps 2 and 6.

4.2 Scenario-based Planning Approach

The SBPA was originally proposed by Bent and Van Hentenryck (2004) for the dynamic vehicle routing problem. At each event, SBPA generates

Algorithm 1 Reoptimization heuristic

1: Create a plan s that contains the requests known at time 0
2: Optimize s with ALNS
3: **while** there is an event **do**
4: Lock all performed actions in plan s
5: Add the new requests to s
6: Optimize s with ALNS and implement the new actions
7: **end while**

scenarios including *future requests* that might appear later on, also called *fictive requests*, by sampling the probability distribution of their appearance. An optimization algorithm then solves each scenario, and routing plans are generated starting from the solutions obtained. The method selects the routing plan to be adopted by using a *consensus function*. A consensus function is a function that receives in input the set of routing plans generated for all scenarios and then returns a score for each of them. The plan with the highest score is the one adopted for implementation. Bent and Van Hentenryck (2004), Voccia et al. (2019) and Song et al. (2020) use consensus functions that score each plan according to the number of features that are present in other plans. This choice aims to select a plan with the highest chance to be cheaply implementable in all scenarios. Voccia et al. (2019) use a function, which we name *Route Similarity* (RS), that consists of counting the number of times the routes of a plan appear in other plans.

Algorithm 2 gives an intuitive SBPA pseudo-code. The first five steps are the same steps adopted in the reoptimization heuristic. Then, at each event, a set of scenarios containing fictive requests is generated (step 5). A plan s_ω is created for each scenario ω, containing all known requests, locked components of solution s, and a set of fictive requests from ω. Each plan s_ω is optimized by solving the subproblem, and the routes having at least one fictive request are removed. This removal can be seen as a way to delay the departure of the corresponding vehicles to possibly accommodate new requests that might arrive in the near future. The last step 12 consists of selecting the plan with the highest consensus function score and implementing it.

To better describe the algorithms, in the remainder of the paper, we will resume the following example a number of times.

Example 1. We are given three routing plans (i.e., solutions), each having three routes that can possibly depart immediately. By defining a plan in

Algorithm 2 Scenario-based planning heuristic

Input: Consensus function f

Create a plan s that contains the requests known at time 0
while there is an event or time = 0 **do**
  Lock all performed actions in solution s
  Add the new requests to s
  Create a scenario set Ω of fictive requests
  for each scenario ω in Ω **do**
    $s_\omega = s \cup \omega$
    Optimize s_ω with ALNS
    Remove routes from s_ω containing fictive requests
  end for
  Implement the plan s_ω having the highest consensus function score $f(s_\omega)$
  $s = s_\omega \setminus \{\omega\}$
end while

square brackets, and a route in round brackets, the example is: [(1-3-5-7), (4), (6-2)], [(1-3), (4-7), (6)] and [(wait), (4), (6)].

We noticed that the RS function selects the plan with the fewest number of actions when the plans are highly heterogeneous. This is a *least-commitment strategy* according to Ghiani et al. (2012). In Example 1, the routes of plans #1 and #2 appear just once, whereas those of plan #3 appear twice each, and hence plan #3 would be the one chosen and implemented by the RS function. The drawback of a plan built with the least-commitment strategy is that it might require a larger number of routes, thus increasing the distance, without performing the most common actions found among scenarios. For Example 1, in 2 out of 3 plans, the most common actions are to depart now and deliver requests 1, 3, 4, 6, and 7.

In the following, we propose two new consensus functions that look at different actions in the plans. The first, called *Assignment Similarity* (AS) function, consists in counting the number of times the pairs (request, route number) of a plan appear in other plans. Ties are broken by plan number. In Example 1, plan #1 has an AS score of 6, plan #2 of 6, and plan #3 of 4. The first plan would thus be the one selected by function AS.

The second function, called *Edit Distance* (ED), sums the number of changes required in a plan to obtain each of the other plans. The plan having the smallest number of required changes is selected. Function ED

builds upon the Levenshtein distance (Levenshtein 1966), which is used to count the minimum number of changes required to change one word into another word. For Example 1, to obtain plan #2 from plan #1, we would need to remove requests 5 and 7 from route 1, add request 7 to route 2, and remove request 2 (resulting in 4 changes). To obtain plan #3 from plan #1, we would need instead to remove requests, 1, 2, 3, 5, and 7 (for a total of 5 changes). The ED score of plan #1 is thus 9. Similarly, the ED score of plan #2 is 7, and that of plan #3 is 8. The second plan would thus be selected by function ED. A detailed computational assessment of the RS, AS, and ED consensus functions is provided below in Section 5.

In general, it is not obvious which plan to select. The purpose of the next section is to propose a method that can produce plans with a high level of similarity to make decisions.

4.3 Branch-and-regret Heuristic

The B&R heuristic was proposed by Hvattum et al. (2007) for solving SDVRPs. Its main components are similar to those adopted in the SBPA, but the method goes a step forward to find solutions that should be better on average. In the B&R, at each new event, scenarios are generated and plans are obtained by optimizing the scenarios. Next, the costs of implementing different alternatives are evaluated with the aim of finding a plan that can be implemented in all scenarios. In Hvattum et al. (2007), this is achieved in two steps: first, assign a narrower time window to each known request, and second, select which requests are served next by the available vehicles. The average cost of a narrower time window for a specific request is obtained by fixing the time window to be in the next time interval in each scenario. The average cost of *regretting* this alternative is also calculated by changing the time window to a farther time in the future. The *branching* consists in imposing the time window leading to the least average cost. The second step, aimed at selecting which requests are served next, is performed once the time windows of all known requests have been fixed. Once this two-step decision phase is concluded, the actions to be performed immediately are implemented, whereas the other actions, like the time window changes, are canceled (as they can be reevaluated at the next event).

In our study, we differ from Hvattum et al. (2007) by evaluating different alternatives that are valuable in the context of the SDDP. This is done in three steps:

1. evaluating if the vehicles at the depot should wait;

2. ensuring that the same alternative is implemented in each scenario for each known request;

3. selecting a plan with a consensus function.

In the first step, we evaluate two alternatives: 1) vehicles at the depot perform their routes as planned, and 2) vehicles at the depot wait for one unit of time. For both alternatives, we optimize all scenarios and compute their average cost. If the second alternative is not more expensive than the first one, then we opt to wait. To this aim, we create a new event whose delay is set to the maximum delay that can be added to the routes of the scenarios so that they remain feasible. If, instead, the second alternative is more expensive than the first one, then we proceed to the next step.

In the second step, we make sure that for all known requests, exactly one among the three following alternatives is selected in all scenarios:

- a vehicle departs now from the depot to deliver the request (*go now*);
- a vehicle departs from the depot at a future time to deliver the request (*wait*);
- the request is not delivered at all (*reject*).

If at least one request does not implement the same alternative in all scenarios, we iterate through these *unfixed* requests to ensure that all plans implement the same alternative for each request. At each iteration, we select the unfixed request having the highest count of the *go now* alternative. Next, we evaluate the average cost of performing each of the previous three alternatives. This is done by calculating the routing cost of imposing each alternative in each scenario. The alternative having the lowest average cost is chosen and implemented. This evaluation process is performed until each known request implements the same alternative in all scenarios.

Finally, the third step is to simply choose a plan using one of the three previously defined consensus functions (RS, AS, or ED).

A pseudo-code of our B&R heuristic is presented in Algorithm 3. The first steps are very similar to those of SBPA. At each event, the plan is locked, the new requests are added to s, and a set of scenarios is generated. Each scenario ω is optimized to obtain a plan s_ω. We evaluate the alternative of having the vehicles wait at the depot. If this is not more expensive than having the vehicles perform their routes as planned, we wait for the next event. Next, we calculate from the plans s_ω the set L of requests that do not implement the same alternative in all scenarios. Next, we perform the

following steps until L is empty. First, for each request $r \in L$, we calculate $gonow[r]$ as the number of times r is in a route that departs now in the plans s_ω. We select the request r having the highest $gonow[r]$ value. We define Φ_r as the set of alternatives for request r and for each alternative $\phi \in \Phi_r$ we impose ϕ in each scenario ω to obtain the plan s_ω^ϕ. We calculate the average cost of each alternative, and the alternative ϕ with the lowest cost is chosen and implemented ($s_\omega = s_\omega^\phi$). Set L is then updated using the plans s_ω, and the process is iterated for the next request. Once L is empty, we implement the plan s_ω with the highest score, that is, we remove the future requests from s_ω to obtain s.

Algorithm 3 Branch-and-Regret heuristic

Input: Consensus function f

Create a plan s that contains the requests known at time 0
while there is an event or time = 0 **do**
  Lock all performed actions in plan s
  Add the new requests to s
  Generate scenario set Ω of fictive requests
  for scenario ω in Ω **do**
    Set $s_\omega = s \cup \omega$ and optimize plan s_ω with ALNS
  end for
  Evaluate if the vehicles at the depot should wait, if so, wait for the next event
  L: the requests that do not implement the same alternative in all scenarios
  while L is not empty **do**
    Calculate $gonow[r]$ from plans s_ω for each $r \in L$
    Select the request $r = \arg\max_{r \in R'}\{gonow[r]\}$
    for each alternative ϕ in Φ_r and scenario ω in Ω **do**
      Set $s_\omega^\phi = s_\omega$ and optimize s_ω^ϕ imposing ϕ on r
    end for
    Implement the least-cost alternative ϕ on r
    $s_\omega = s_\omega^\phi$
    Update L from the plans s_ω
  end while
  Select the plan s_ω having the highest score on consensus function f
  $s = s_\omega \setminus \{\omega\}$
end while

4.4 Scenario Generation

Both SBPA and B&R use sampled scenarios to guide the decision process. Many authors (as Hvattum et al. 2006, 2007) generate new scenarios at each event. Others (as Bent and Van Hentenryck 2004, Voccia et al. 2019, and Song et al. 2020) generate all sampled scenarios at the beginning of the time horizon and then, as time progresses, update the scenarios by adding the new revealed requests and removing those that did not appear. The number of scenarios has a major impact on solution quality and computation time. Typically, computation time increases linearly in the number of scenarios. Different conclusions are taken in the literature on the number of scenarios that produce the best trade-off between solution quality and computing effort. Hvattum et al. (2006) tested several sizes ranging from 30 to 600 scenarios and concluded that the best option is to have as many scenarios as possible. In a different setting, Hvattum et al. (2007) obtained a different conclusion as, after testing sizes from 1 to 60 scenarios, they obtained their best solutions with 30 scenarios. In Voccia et al. (2019), the authors made tests using 10, 25, and 50 scenarios, and noted that the best performance was obtained using 10 scenarios. Clearly, all these conclusions are problem-dependent and based on the particular instances addressed in those papers. In Section 5.1 below, we obtain some more insights on this aspect by means of extensive tests.

The size of the sampling horizon is also discussed in Voccia et al. (2019). Basically, the idea they investigated is to maintain in the scenarios the fictive requests that appear in the successive ρ instants of time. This reduces the subproblem size, lowers computation times, and emphasizes the decisions that have to be taken in the immediate future. The rationale is that fictive requests in a faraway future might simply act as noise in the decision-making process. This strategy is also tested in Section 5.1. In addition, we also propose a new alternative method to the sampling horizon of Voccia et al. (2019). As it can be noted, using a fixed sampling horizon can perform well when the length of the horizon is correlated with the data of the instance. However, lower quality results might be expected when ρ is too small and the time windows are large, as this might lead to too early departures. The alternative method that we developed, called *correlated-data sampling* (CDS), incorporates only the fictive requests having a release time lower than the farthest end time window of any known request plus a constant value $\bar{\rho}$. The idea, computationally tested below in Section 5, is to consider only fictive requests that can impact the decision process.

4.5 Optimizing Subproblems

The three heuristic approaches that we implemented are based on the iterated solution of DPDP subproblems that appear during the search. This is made by the *Optimize* function, invoked by Algorithms 1, 2, and 3. The function receives as input a plan that might contain empty routes and not-yet assigned requests, routes with completed requests, or routes with a mix of completed and non completed requests. It aims to obtain a plan that satisfies all constraints of the DPDP indicated in Section 4 and has the minimum total cost. In addition, when invoked by the B&R heuristic, Optimize returns a high cost for any plan not respecting the decisions taken at the previous branches. The function that we implemented is based on the execution of four steps.

First, the unassigned requests are sequentially inserted inside the plan using the *Regret-k* heuristic of Potvin and Rousseau (1993). The algorithm works as follows: for each request, it calculates the minimal insertion cost of the request inside each route. Then, at each iteration, it selects the request having the largest sum of differences between the best insertion cost and the insertion cost into the other k best routes, in absolute value. The selected request is inserted in the route with the best insertion cost. The minimal insertion cost of the remaining requests inside this route is updated. If we cannot feasibly insert a request, then this request is inserted into a bank and resumed later in the local search phase.

Second, the local search operators *Relocate* and *Exchange* are executed. The two algorithms operate similarly: Relocate removes a request from its current position in a route or from the request bank, and attempts to reinsert it in another position in the same route or another one, whereas Exchange takes two requests from different routes and tries to insert them in each other route. Both methods look for insertion positions that minimize the cost of the resulting plan. If improving positions are found, they are implemented; otherwise, the requests are reinserted in their original positions. The two operators are executed, one after the other, until no further improvement can be found.

Third, the ALNS of Ropke and Pisinger (2006) is called. At each iteration, a removal and an insertion heuristic are selected from a pool of heuristics. A random number of requests are then removed from the plan and inserted in the request bank using the removal heuristics. Then, the insertion heuristic tries to insert them back in the plan with the hope of finding an improved solution. In addition, the removal and insertion heuristics make sure that completed or fixed requests remain in their position.

Our implementation is the same as the one described in Ropke and Pisinger (2006), with the following exceptions: 1) we only use the Random and Shaw removal heuristics; 2) the cooling rate is set to 0.8; and 3) the number of iterations is decided according to the tests of Section 5.

Fourth, the two local searches invoked at step two are invoked once more in a last attempt to improve the plan.

4.6 Dealing with Preemptive Depot Returns

The option of allowing or not PDRs was formally proposed by Ulmer et al. (2019). For many works in the literature (namely, Azi et al. 2012, Voccia et al. 2019, Klapp et al. 2018 and Archetti et al. 2020), once a vehicle departs to perform deliveries, it has to complete its entire route before returning to the depot. PDRs allow vehicles to return to the depot before the routes are completed. Enabling this might help at reducing distances or at delivering more requests. The conclusion on whether this policy is advantageous or not are mixed. Klapp et al. (2018) claim that the benefits are marginal. On the other hand, Ulmer et al. (2019) state that the policy leads to relevant savings.

Our framework can easily deal with the PDR variant because of the way the SDDP is modeled. As mentioned, we represent the SDDP as a DPDP, and having pickup and delivery nodes enable subproblems to decide easily if a pickup is to be inserted between two deliveries belonging to a newly departed route. The pickup would represent a return to the depot. Each vehicle maintains a *current node* that represents the first node after which a request can be inserted. Route modifications can only occur after that node. Note that if we want to forbid PDRs, then we simply set the current node to the last delivery node of the route.

5 Experimental Results

We executed an extensive computational evaluation of our solution approaches to evaluate their performance under different configurations and for different instance sizes. We first selected a varied subset of 125 instances among those proposed by Voccia et al. (2019) and used them to analyze how the algorithmic performances are affected by changes in the main parameters. The outcome of this first set of tests is discussed in Section 5.1. After having determined the best configurations, we evaluated the impact of PDRs in Section 5.2, where we also compare our results with those by Ulmer et al. (2019) on the set of instances that they created. Then, in Section 5.3

we compare our algorithms with those of Voccia et al. (2019) on their entire set of instances.

Our algorithms have been coded in C++ and our tests have been executed by using a single core of an Intel 2.667 GHz Westmer EP X5650 processor. For comparison purposes, we have made the instances that we used publicly available at `https://sites.google.com/view/jfcote/`.

5.1 Parameter Setting and Sensitivity Analysis

In this section, we analyze the results obtained by our algorithms using different parameter configurations. The instances that we used for this analysis are a subset of 125 instances selected from the benchmark presented by Voccia et al. (2019). We considered instances with a number of customers varying from 71 to 110, and divided into three types according to the way customer locations have been generated: clustered (C); randomly dispersed (R); and both randomly dispersed and clustered (RC). In addition, the instances are characterized by five types of time windows. The first four types, namely TW.d1, TW.f, TW.h, and TW.r, have all a one-hour deadline but differ on the start time of the time windows, which is equal to the release date for TW.d1, a fixed time in the future for TW.f, the remaining hours of the day for TW.h, and randomly dispersed times for TW.r. The fifth type, called TW.d2, is equivalent to TW.d1 but has a two-hour deadline. We considered instances with an arrival rate equal to 0.002 per minute, for 100 potential customers, which considering a time horizon of 480 minutes, produce on average 96 customers. The instances are grouped according to 15 different location distributions, namely: C_1 to C_5 (50 instances in total, 10 per location); R_1 to R_5 (50 instances, 10 per location); and RC_1 to RC_5 (25 instances, 5 per location).

In all the tests of this section, we considered a fleet of 10 vehicles, 60 minutes of time horizon, and 30 scenarios. Moreover, waiting at the depot is allowed, but PDRs are not. In the tables below, we evaluate each algorithm in terms of:

- %filled = percentage of requests served;
- dist. = total distance traveled by the vehicles;
- time = computing time in seconds.

In Table 1, we evaluate our approaches by attempting different numbers of ALNS iterations. We tested the Reoptimization approach, three SBPA configurations attempting the three consensus functions (namely, RS, AS

and ED) and three B&R configurations using the same consensus functions. Each algorithm has been executed with a number of ALNS iterations varying in the set $\{50, 100, 250, 500, 1000\}$. The values shown in the table are the average of the values obtained on the 125 instances. A final line showing average values for the entire column is also reported to gain some insight into the impact of the attempted parameter on all algorithms. The rightmost column presents, instead, the average results over all tests performed with the given algorithm.

The results of Table 1 show that the Reoptimization heuristic is very quick but has a worse performance than the other algorithms that make use of stochastic information. It cannot reach high %filled values, which is on average 90.49%, but at the same time, it produces a high average traveled distance, which is always above 2900. The computing time increases almost linearly with the number of ALNS iterations performed. Among the three SBPA configurations, the RS function by Voccia et al. (2019) achieves a good 91.22% of average %filled, being better than both AS and ED. The traveled distance is also usually lower for RS, whereas the computing times are quite similar. The B&R algorithms obtained the best quality solutions, with an average %filled value above 92% for all configurations. In addition, their computing time is similar, if not lower, to that required by the SBPA. Among the consensus functions, both RS and AS perform very well, and ED has a slightly weaker performance.

The use of 1000 ALNS iterations allows almost all algorithms to produce their best results, but this is obtained at the expense of high computing times. The work by Hvattum et al. (2006) affirmed that a better solution to the subproblem did not necessarily lead, on their instances, to a better overall solution of the dynamic problem. Our results show that an increase in the ALNS iterations can lead to higher %filled values, as can be noticed by the last overall average line in the table. However, we believe that a good trade-off between quality and time is obtained by using 100 ALNS iterations, and we kept this value in all the successive computational tests.

Next, we evaluate our algorithms by attempting different time horizon strategies. The first one, proposed by Voccia et al. (2019) and discussed in Section 4.4, works on the size of the sampling horizon by maintaining the requests that appear in the next ρ instants of time. The newly-introduced CDS strategy maintains in the scenarios only the fictive requests that have a release time lower than the farthest end time window of any known request plus a constant time $\bar{\rho}$.

The results of the first horizon strategy are shown in Table 2, where we attempted different ρ values. We tested all algorithms from Table 1, except

Table 1: Attempting different ALNS iterations

	ALNS It.=50			ALNS It.=100			ALNS It.=250			ALNS It.=500			ALNS It.=1000			average
algorithm	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled
Reoptimization	90.43	3007.5	0.1	90.26	2984.9	0.3	90.54	2977.4	0.6	90.58	2993.6	1.1	90.65	2969.3	2.2	90.49
SBPA-RS	90.88	2709.5	117.0	91.58	2693.6	194.7	90.84	2640.8	427.3	90.97	2628.4	821.2	91.83	2625.8	1640.1	91.22
SBPA-AS	90.53	2869.4	109.6	91.30	2851.2	182.1	90.65	2814.1	398.4	90.86	2814.9	749.5	90.94	2819.2	1484.0	90.86
SBPA-ED	90.15	2775.2	116.1	90.51	2763.9	196.6	90.39	2728.6	435.3	90.64	2714.1	831.6	90.61	2683.1	1627.9	90.46
B&R-RS	91.67	2390.5	105.5	92.16	2405.8	172.1	92.26	2394.5	373.5	92.12	2383.8	713.9	92.50	2400.8	1403.5	92.14
B&R-AS	92.14	2421.5	94.3	92.20	2428.4	150.1	91.92	2421.0	313.9	92.08	2419.5	589.2	92.35	2416.0	1171.8	92.14
B&R-ED	91.71	2417.9	102.1	91.97	2402.8	162.7	92.26	2414.8	351.3	92.20	2411.1	659.5	92.38	2404.6	1327.8	92.10
average	91.07	2655.9	92.1	91.43	2647.2	151.2	91.27	2627.3	328.6	91.35	2623.6	623.7	91.61	2617.0	1236.8	91.35

for the Reoptimization heuristic, which is not affected by the horizon strategy. The columns have the same meanings as those of Table 1. Intuitively, a smaller time horizon requires shorter computation times and provides good solutions because the problems are smaller. Larger time horizons should provide more robustness by requiring more time to compute, but they could possibly lead to worse solutions because of the increased size of the problems solved.

The results in Table 2 show that among the SBPA configurations, the best %filled and dist. average values are achieved with function RS and $\rho = \infty$ (i.e., maintaining all the requests of the day in the sampled scenarios). The computing times are quite similar for all SBPA configurations, and they increase with the size of the time horizon, with AS being slightly faster than RS and ED. The B&R results are better than those obtained with the SBPA configurations, but the improvement over SBPA decreases when the size of the time horizon increases. The best results are obtained by configurations RS and ED for $\rho = \infty$. The computing times are always shorter than those required by the SBPA. The difference is remarkable for $\rho = \infty$, where the B&R algorithm requires on average about two-thirds of the time spent by the SBPA approaches. The work by Voccia et al. (2019) concluded that a larger value of the time horizon does not necessarily lead to a better solution, but our results show that, on the contrary, the largest value (i.e., $\rho = \infty$) leads to the greatest performance in terms of solution quality.

In Table 3, we evaluate the same algorithms of Table 2 but using the CDS strategy. We vary the time horizon by attempting different values of $\bar{\rho}$, from -30 to $+15$. Both SBPA and B&R algorithms appear to be very robust with respect to changes in this parameter. The computing times just slightly increase when $\bar{\rho}$ increases, whereas %filled and dist. remain quite untouched. Once more, we can notice a better behavior of the B&R approaches over the SBPA ones.

Table 2: Attempting different time horizons (Voccia et al. 2019 strategy)

	ρ=15			ρ=30			ρ=60			ρ=120			ρ=∞		
algorithm	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time
SBPA-RS	90.82	2781.1	74.9	91.22	2655.2	118.4	91.57	2693.6	202.8	91.28	2729.8	374.5	92.22	2601.9	1146.2
SBPA-AS	90.53	2925.5	68.5	91.15	2818.2	107.7	91.30	2851.2	189.2	90.60	2993.4	351.5	91.59	3071.1	1067.2
SBPA-ED	90.85	2776.6	73.8	90.48	2688.1	120.1	90.51	2763.9	203.9	90.37	2827.6	368.6	89.88	2837.3	1124.0
B&R-RS	91.73	2371.8	58.4	91.33	2374.3	95.0	92.16	2405.8	171.2	92.91	2459.8	336.5	92.93	2578.0	816.1
B&R-AS	91.04	2417.1	50.4	91.29	2389.2	79.4	92.20	2428.4	149.6	92.39	2482.1	297.3	92.82	2607.8	735.8
B&R-ED	91.50	2363.4	56.4	91.64	2378.1	93.3	91.97	2402.8	166.1	92.74	2472.0	321.0	92.94	2609.0	761.5
average	91.08	2605.9	63.7	91.18	2550.5	102.3	91.62	2590.9	180.5	91.71	2660.8	341.6	92.06	2717.5	941.8

The largest average %filled value of 92.14% is achieved for both $\bar{\rho} = 0$ and $\bar{\rho} = 15$, with a slight smaller time for $\bar{\rho} = 0$. The values obtained are better than those shown in Table 2, where the best average %filled value is 92.06%, and this proves the relevance of the newly-introduced CDS strategy. In the next tests, we hence use the CDS strategy with $\bar{\rho} = 0$. Notice that this means that we consider only fictive requests whose release time is within the known request's time windows.

Table 3: Attempting different time horizons (correlated-data strategy)

	$\bar{\rho} = -30$			$\bar{\rho} = -15$			$\bar{\rho} = 0$			$\bar{\rho} = 15$		
algorithm	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time
SBPA-RS	91.83	2746.2	729.5	92.03	2742.1	746.0	91.46	2749.6	768.2	92.12	2739.0	789.1
SBPA-AS	91.48	3026.8	687.1	91.05	3026.6	710.1	91.55	3051.6	734.9	91.26	3028.6	753.6
SBPA-ED	90.43	2823.1	714.3	89.57	2830.2	731.6	90.80	2852.5	751.2	90.57	2862.0	774.3
B&R-RS	93.35	2526.8	513.7	93.41	2531.6	527.9	92.90	2542.3	535.6	92.93	2547.5	557.5
B&R-AS	92.92	2556.2	459.8	93.37	2559.3	479.2	93.21	2570.9	493.6	93.14	2575.7	509.7
B&R-ED	92.61	2522.5	479.6	92.65	2538.5	498.1	92.92	2557.3	517.0	92.84	2554.5	532.7
average	92.11	2700.3	597.3	92.01	2704.7	615.5	92.14	2720.7	633.4	92.14	2717.9	652.8

The last parameter that we evaluate in this section is the number of sampled scenarios to adopt. In Table 4, we consider the same approaches used in the previous tables, and test them with a number of sample scenario that varies from 5 to 30. We can notice that the number of scenarios positively affects the %filled values, which increase slightly but constantly. Also, the distance increases slightly, but this might be due to the higher number of requests delivered. As expected, the number of scenarios has a relevant impact on the computing times. Voccia et al. (2019) indicated that 10 scenarios are sufficient to attain the highest %filled values and the lowest traveled distances. Based on the results we obtained, we opted to use 30

scenarios in all next experiments. Accordingly, we decided to adopt the AS consensus functions, which is the function that gives the best results for 30 scenarios for both SBPA and B&R.

Table 4: Attempting different numbers of sampled scenarios

	Scenarios=5			Scenarios=10			Scenarios=20			Scenarios=30		
algorithm	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time	%filled	dist.	time
SBPA-RS	92.14	2793.5	125.4	91.84	2781.1	252.2	92.32	2760.2	506.0	91.46	2749.6	749.4
SBPA-AS	91.15	3032.2	122.3	91.41	3055.7	240.6	91.25	3036.4	482.8	91.55	3051.6	714.7
SBPA-ED	90.49	2881.5	123.5	90.18	2846.0	248.5	90.26	2844.7	495.5	90.80	2852.5	732.9
B&R-RS	92.24	2515.3	57.9	92.63	2516.6	140.2	92.80	2535.5	329.9	92.90	2542.3	522.4
B&R-AS	91.80	2535.6	54.2	92.65	2532.0	129.5	92.91	2568.0	307.3	93.21	2570.9	480.3
B&R-ED	92.01	2527.2	57.0	92.73	2530.8	133.8	93.03	2544.8	313.9	92.92	2557.3	503.3
average	91.64	2714.2	90.1	91.90	2710.4	190.8	92.09	2714.9	405.9	92.14	2720.7	617.2

Summarizing, based on the sensitivity analysis that we performed, we use 100 ALNS iterations, CDS strategy with $\bar{\rho}$=0, AS consensus function, and the number of scenarios equal to 30. These parameters are used to obtain the results of all approaches in the next sections.

5.2 Evaluations of Preemptive Depot Returns

In Table 5, we evaluate the problem variant that considers PDRs. We consider the Reoptimization heuristic and the SBPA and B&R algorithms with the AS consensus function. Each approach is tested by allowing or not PDRs. We present the average values obtained on the set of 125 instances already adopted in Section 5.1. To gain some more insight on the impact of PDRs, we tested the algorithms by varying the number of available vehicles, ranging from 6 to 12. The last three groups of lines in the table show the overall average values for, respectively, %filled, dist., and time, by considering all three algorithms with and without PDR, and the relative deviations between these values (computed as $(z_{PDR} - z_{no\ PRD})/z_{no\ PDR}$, where z is one of the three measures).

The use of PDRs allows to obtain very interesting results. It consistently increases the percentage of requests filled with no PDR. This can be noticed for all algorithms, with SBPA-AS being the one with the greatest benefit. Considering the overall averages, the increase in the %filled value is particularly evident when the number of vehicles is small, and decreases when it is higher. This is due to the larger need for optimization when the problem is more constrained, and the vehicles are a very scarce resource. Another interesting behavior can be noticed for the traveled distance. For small vehicle

numbers, the PDRs consistently increase the number of requests delivered, and this automatically leads to larger distance values. When, instead, the fleet is large, the increase in %filled is not so relevant, but PDRs manage to decrease the traveled distance, which is another positive result. It is also worth noticing that when PDRs are not allowed, B&R-AS consistently achieves the best results in terms of %filled. When, instead, PDRs are allowed, B&R is better for instances with 10 or 12 vehicles, whereas SBPA-AS is better for 6 and 8 vehicles. In terms of computing times, PDRs require just a slightly larger computing effort with respect to the case with no PDRs, ranging from 2.3% to 14.1% and being 6.6% on average. Overall, we can conclude that the use of PDRs is important and can lead to relevant benefits in SDDPs.

Table 5: Results when preemptive depot returns are allowed

	vehicles=6		vehicles=8		vehicles=10		vehicles=12		average	
algorithm	no PDR	PDR	no PDR	PDR	no PDR	PDR	no PDR	PDR	no PDR	PDR
reoptimization	76.86	81.96	84.90	88.35	90.26	92.19	93.87	94.57	86.47	89.27
SBPA-AS	77.55	85.02	86.07	91.10	91.55	94.00	94.25	95.85	87.35	91.49
B&R-AS	79.31	83.79	88.25	90.87	93.21	94.53	95.63	96.08	89.10	91.32
average %filled	77.91	83.59	86.41	90.11	91.67	93.57	94.59	95.50	87.64	90.69
% filled deviation (%)		7.3		4.3		2.1		1.0		3.5
average dist.	2169.4	2262.7	2589.1	2631.3	2869.1	2869.7	3027.3	2994.6	2663.7	2689.6
dist. deviation (%)		4.3		1.6		0.0		-1.1		1.0
average time	456.3	520.8	436.5	459.8	398.4	407.7	381.6	394.3	418.2	445.6
time deviation (%)		14.1		5.3		2.3		3.3		6.6

Table 6 details the average %filled results from Table 5 by considering the five types of time windows that have been adopted in the instance creation (as described at the beginning of this section). Each type has been adopted for 25 out of 125 instances. Hence, each value in the table corresponds to an average of 100 solutions, obtained by solving the 25 instances with a fleet of 6, 8, 10, and 12 vehicles. We believe that this analysis is of interest because we can notice a different impact of PDRs on the time window types. For TW.d1, TW.d2, and TW.f, which are all characterized by a somehow regular time window start time (either fixed in the future or coincident with the request release date), the impact is negligible, and in a few cases, even negative. For TW.h and TW.r, which are characterized by variegate and even randomly dispersed time window start times, the impact is instead very relevant, being equal to 8% or 9% on average and even 10% for SBPA-AS. We can conclude that, when solving a SDDP, it is important to consider the

structure of the time windows to decide whether to allow or forbid PDRs.

Table 6: Impact of preemptive depot returns on %filled per time window type

	reoptimization		SBPA-AS		B&R-AS		average	
TW type	no PDR	PDR	no PDR	PDR	no PDR	PDR	no PDR	PDR
TW.d1	84.94	84.42	85.56	85.21	86.73	86.59	85.75	85.41
TW.d2	94.58	93.88	94.99	95.13	95.46	95.54	95.01	94.81
TW.f	94.78	94.22	95.04	95.11	95.42	95.41	95.08	94.89
TW.h	77.89	86.26	80.22	90.29	83.19	89.24	80.43	88.78
TW.r	80.18	87.72	80.97	90.98	84.71	89.79	81.95	89.63
average	86.47	89.30	87.35	91.49	89.10	91.32	87.64	90.70

To conclude this section, we present an evaluation of our algorithms on the instance set which has been proposed by Ulmer et al. (2019) to evaluates PDRs. The set is made by three groups of 54 instances each, differentiated among them by the *degree of dynamism* (DOD). The DOD represents the percentage of requests that are dynamically revealed over the time horizon over the total number of requests. Three values of DOD have been used, namely, 25%, 50%, and 75%. In all instances, just a single vehicle is available to perform the deliveries.

Preliminary experiments that we performed showed that both the SBPA and B&R algorithms consume too much time and were not effective on these instances. This is due to the fact that these algorithms have not been tailored to deal with instances without time windows and with a single-vehicle. In such a case, when the number of vehicles is so small compared to the number of requests, the best option is usually to start delivering as soon as possible, and hence the reoptimization heuristic represents the best strategy. Of course, one still needs to decide how to route the vehicles along their multiple routes.

The results that we obtained are shown in Table 7, where we compare the reoptimization approach, with and without PDRs, with the approach by Ulmer et al. (2019). The latter approach is called APDR in their paper, and makes use of PDRs. For the reoptimization heuristic, we present the average values of #filled, traveled distance, number of routes, and computing time. The #filled value gives the number of requests that have been served among those that are dynamically revealed and is the objective function used in Ulmer et al. (2019). The number of routes corresponds to the number of times the vehicle returns to the depot. For APDR, we only know the average #filled value.

We first note that the smaller is the DOD, the smaller is the #filled value. This is simply imputed to the fact that there are fewer dynamic requests to serve. Independently from that, the Reoptimization heuristic performs better than APDR for all DOD values and both with and without PDRs. In fact, the best #filled values are obtained when PDRs are not allowed. The use of PDRs also impacts on the average number of routes per vehicle, which increases. This is reasonable because the vehicle performs more returns to the depot. Instead, no relevant effect can be noticed for traveled distance, with just a slight decrease when PDRs are used, and for computing time, which is all very low.

In contrast with what was stated by Ulmer et al. (2019), our results show that PDRs are not useful on this set of instances to improve the number of requests filled. This can be imputed to the fact that there are no time windows. The very positive results of the reoptimization heuristic confirm the strength of the routing algorithm that we developed (see Section 4.5), which achieves good-quality solutions with very short computing times.

Table 7: Comparison on Ulmer et al. (2019) single-vehicle instances

		reoptimization no PDR				reoptimization PDR				Ulmer et al. (2019)
DOD	inst.	#filled	dist.	routes	time	#filled	dist.	routes	time	#filled
25%	54	1.87	235.5	1.2	3.3	1.84	233.5	1.5	3.5	1.38
50%	54	5.97	341.7	1.8	0.4	5.34	336.0	2.4	1.3	4.55
75%	54	13.81	401.1	2.2	1.0	12.16	390.2	3.2	0.9	11.39
average		7.22	326.1	1.7	1.6	6.45	319.9	2.3	1.9	5.77

5.3 Comparison with Voccia et al. (2019)

In this section, we compare our algorithms with those of Voccia et al. (2019) on their entire set of homogeneous instances with $\lambda = 0.2$. This is composed of 4050 instances, all characterized by a fleet of three vehicles. The results that we obtained are presented in Table 8. In "Offline", we show the results obtained on the static variant of the problem in which all information is assumed to be known in advance. This variant is solved with a unique call to the routing optimize function of Section 4.5. We then show the results obtained with the most relevant configurations of Reoptimization, SBPA, and B&R algorithms, in our implementations and in the implementation by Voccia et al. (2019) as well. For Voccia et al. (2019), we show the results of their Reoptimization and SBPA-RS algorithms (which they call with and without sampling, respectively), both of which do not use PDR. Among our

methods, we test Reoptimization with and without PDR, SBPA with RS and AS functions and no PDR, and B&R-AS with and without PDR.

For each algorithm, we present the values already discussed in the previous tests, in addition to the number of events (i.e., the number of times the algorithm is invoked) and the time per event (computed as time/events). For Voccia et al. (2019), we only know the average %filled and time per event values, which have been taken from their papers. Their algorithms were implemented in Python and tested on a computing cluster equipped with a combination of 2.6 GHz and 2.9 GHz processors running CentOS 6.3, which can be considered similar to the computer we used for our tests.

The Offline algorithm shows that the best possible %filled value achievable (with our heuristic) corresponds to more than 86%. This indicates that the more information we have about the problem, the better results we can obtain. Among the Reoptimization methods with no PDR, our algorithm is faster than the one by Voccia et al. (2019) and can serve many more requests (53.29% vs. 36.81%). This can be imputed to the efficiency of our routing optimize function and probably also to a difference in the code implementations. The Reoptimization method with PDR obtains a remarkable %filled of 60%, which proves, as noted in the previous sections, that this simple approach is good enough on instances with small fleet sizes.

For what concerns SBPA-RS, our method is faster and more effective than the one by Voccia et al. (2019) (1.9 vs. 95.2 seconds of computing time per event, and 51.53% vs. 41.99% served requests). Further improvements are obtained by the SBPA that adopts the new AS consensus function, which increases the %filled and slightly decreases the time per event. This is obtained at the expense of a reasonable increase in the traveled distance and in the number of routes.

The best results are obtained, once more, by the B&R algorithms. When PDRs are not allowed, B&R-AS can serve 54.26% of the requests, which is about 1% better than Reoptimization and our SBPA-RS. When they are allowed, the %filled increases to slightly more than 60%. The number of events faced by the B&R methods is about one-third of those faced by the SBPA ones. However, their time is larger (about three seconds per event vs. less than two) because of the increased complexity of the procedures invoked at each event. Nevertheless, the computing times per event remain very low and perfectly compatible with a real-world use of the methods.

The positive results that we obtained for the case with three vehicles can also be confirmed for other fleet sizes, as graphically depicted in Figure 1. We executed all algorithms given in the legend of the figure on the entire set of instances by varying the number of vehicles as shown on the x-axis. The

Table 8: Comparison with Voccia et al. (2019) (three vehicles)

algorithm	PDR	%filled	dist.	time	routes	events	time/events
Offline	-	86.51	1289.3	23.2	14.9	1.0	23.2
Reoptimization	NoPDR	53.29	1214.7	0.2	12.3	87.0	0.0
Reoptimization	PDR	60.02	1318.6	0.4	13.7	138.0	0.0
Reopt. (Voccia et al. 2019)	NoPDR	36.81	-	-	-	-	3.3
SBPA-RS (Voccia et al. 2019)	NoPDR	41.99	-	-	-	-	95.2
SBPA-RS	NoPDR	51.53	1084.0	865.7	11.5	461.8	1.9
SBPA-AS	NoPDR	53.27	1236.6	766.5	13.4	453.7	1.7
B&R-AS	NoPDR	54.26	1236.2	335.2	11.7	113.8	3.1
B&R-AS	PDR	60.16	1304.4	377.6	13.4	116.9	3.3

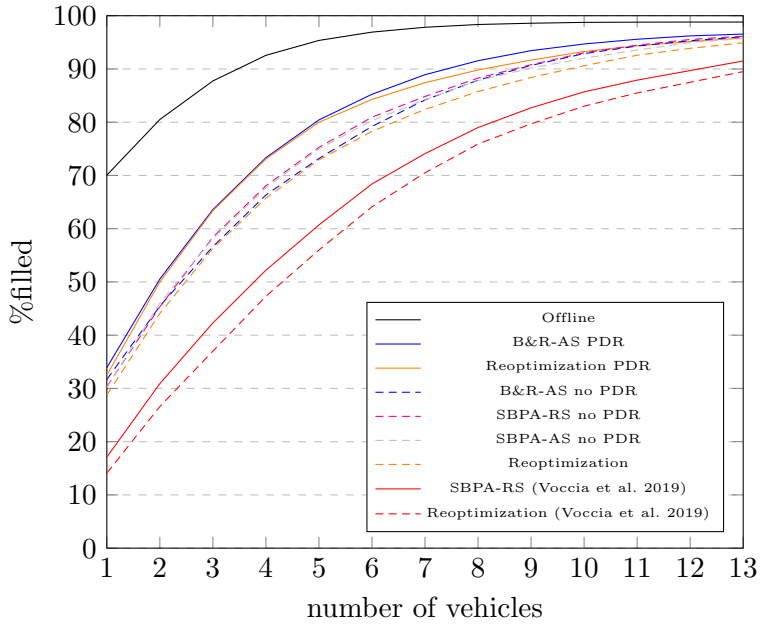
y-axis shows the average %filled value achieved on the different runs. The Offline algorithm is, quite obviously, the one obtaining the best performance, and the second and third ones are the B&R and Reoptimization methods with PDR. These two methods have almost coincident performance for small fleet sizes (up to 4 vehicles) but then diverge because B&R performs better. Then, they asymptotically converge when the number of vehicles becomes very large. Below these methods, we can find the ones that do not allow PDR, among which the B&R-AS is the best one.

6 Conclusions and Future Research

In this paper, we studied dynamic vehicle routing problems where stochastic customers request deliveries with strict and close time windows, and the aim is to maximize served requests and minimize traveled distances. This type of problem is known in the literature as the same-day delivery problem and is of great relevance because it models a number of real-world applications, including the delivery of online purchases. We developed a good set of solution algorithms, ranging from a simple reoptimization heuristic to a sophisticated branch-and-regret in which sampled scenarios are used to anticipate decisions. We tested the algorithms on a large set of benchmark instances from the literature, obtaining very favorable comparisons with both Voccia et al. (2019), on the basic problem variant, and Ulmer et al. (2019), on the problem variant where preemptive returns of the vehicles to the depot are allowed. Notably, on a large benchmark set of 4050 instances from Voccia et al. (2019), we raised the rate of served requests from about 42% to more than 60% and, at the same time, we decreased the computing time per event from about 95 seconds to just 3.

These good results have been obtained by developing dedicated algo-

Figure 1: Percentage of requests filled as the number of vehicles increases



rithms that employ features from the literature as well as new techniques, and by performing a careful calibration of the parameter settings. Overall, we found out that a simple Reoptimization heuristic is good enough to provide efficient solutions for cases where the number of vehicles is small and thus departing as soon as possible is usually the best option. In such cases, it is still important to devote a good effort to optimize the vehicle routes, for which we found convenient to adopt a classical adaptive large neighborhood search (Ropke and Pisinger 2006).

To obtain better results, it is important to make use of stochastic information. As suggested by Bent and Van Hentenryck (2004), we made use of consensus functions to select the best set of routes when both real and sample requests are taken into account. We found out that the consensus functions have an important impact on the performance of the algorithms and that there is still relevant research to be done in this field. Indeed, a new consensus function that we proposed, based on assignment similarity, is the one that allowed us to obtain the best results on average.

The best performance has obtained by branch-and-regret algorithms. Naive implementations of these algorithms may be very time-consuming, as a large number of alternatives must be taken into considerations and optimized. We found out that a good management of the events might consistently decrease the computing time and, at the same time, still allow to get very high rates of served requests.

In the literature on dynamic stochastic vehicle routing problems, a small problem variation might make it challenging to devise a fair comparison among different algorithms. This difficulty is increased by the fact that instances are complicated, as they typically contain not only the realized requests but also the sample stochastically generated scenarios. In our work, we pursued a fair comparison among different methods, either new or from the literature, and we tried to foster new comparative research by making instances publicly available. These classes of problems are very relevant as they model many emerging real-world applications, and we believe they should be studied in detail in the next future.

There are, indeed, several interesting future research directions to follow. In terms of methodology, we believe that there is still room for improvements in branch-and-regret algorithms by developing new branching rules, alternative consensus functions, and new mechanisms that make better use of the information from the scenarios. In addition, we believe good results could be obtained by a deep study of immediate request acceptance policies, as in Klapp et al. (2020) and Marlin and Thomas (2020), so as to assign as soon as possible a request to a third-party logistic operator. This could decrease

waiting times for the customers, but at the possible expense of an increase in the overall delivery costs.

In terms of optimization problems, as our algorithms are already equipped to solve dynamic pickup and delivery problems, it would be interesting to study their performance on different emerging problem variants. Among these, we would like to cite one-to-many-to-one problems, as in Bruck and Iori (2017), where, in addition to the delivery of merchandise, one has to collect further merchandise to be brought back to the depot. This case could involve stochastic customers, stochastic demands and capacitated vehicles. Multi-pickup and delivery problems with time windows (see, e.g., Naccache et al. 2018 and Aziez et al. 2020) too represent an emerging variant with relevant applications. In these problems, a request is composed of several pickups of different items, followed by a single delivery at the customer location. Stochastic aspects might hence concern both customer and pickup locations.

Finally, we mention the class of meal delivery problems (see, e.g., Ulmer et al. 2021), where the customers require food from restaurants and the aim is to deliver it promptly by considering the time in which it will be ready. There are thus two sources of uncertainty in these problems: the customers, which are unknown until they place an order, and the food release dates at the pickup locations.

Acknowledgments

Jean-François Côté acknowledges support by the Canadian Natural Sciences and Engineering Research Council (NSERC) under grant 2015-04893. Thiago Alves de Queiroz acknowledges support by the National Council for Scientific and Technological Development (CNPq) under grant 311185/2020-7 and the State of Goiás Research Foundation (FAPEG). Manuel Iori acknowledges support from the University of Modena and Reggio Emilia under grant FAR 2018. We thank Compute Canada for providing high-performance computing facilities.

References

Archetti, C., Feillet, D., Mor, A., and Speranza, M. (2020). Dynamic traveling salesman problem with stochastic release dates. *European Journal of Operational Research*, 280(3):832–844.

Arda, Y., Crama, Y., Kronus, D., Pironet, T., and Van Hentenryck, P. (2014). Multi-period vehicle loading with stochastic release dates. *EURO Journal on Transportation and Logistics*, 3(2):93–119.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2012). A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, 199(1):103–112.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2014). An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173.

Aziez, I., Côté, J.-F., and Coelho, L. C. (2020). Exact algorithms for the multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 284(3):906–919.

Battarra, M., Monaci, M., and Vigo, D. (2009). An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem. *Computers & Operations Research*, 36:3041–3050.

Bent, R. W. and Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(2):977–987.

Bent, R. W. and Van Hentenryck, P. (2007). Waiting and relocation strategies in online stochastic vehicle routing. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1816–1821.

Bruck, P. B. and Iori, M. (2017). Non-elementary formulations for single vehicle routing problems with pickups and deliveries. *Operations Research*, 65(6):1597–1614.

Cattaruzza, D., Absi, N., and Feillet, D. (2016). The multi-trip vehicle routing problem with time windows and release dates. *Transportation Science*, 50(2):676–693.

Cattaruzza, D., Absi, N., and Feillet, D. (2018). Vehicle routing problems with multiple trips. *Annals of Operations Research*, 271:127–159.

Cattaruzza, D., Absi, N., Feillet, D., and Vidal, T. (2014). A memetic algorithm for the multi trip vehicle routing problem. *European Journal of Operational Research*, 236(3):833–848.

Dayarian, I., Savelsbergh, M., and Clarke, J.-P. (2020). Same-day delivery with drone resupply. *Transportation Science*, 54(1):229–249.

Fleischmann, B. (1990). *The vehicle routing problem with multiple use of vehicles*. PhD thesis, Fachbereich Wirtschaftswissenschaften, Universität Hamburg.

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.

Ghiani, G., Manni, E., and Thomas, B. W. (2012). A comparison of anticipatory algorithms for the dynamic and stochastic traveling salesman problem. *Transportation Science*, 46(3):374–387.

Haneveld, W. K. K. and van der Vlerk, M. H. (1999). Stochastic integer programming: General models and algorithms. *Annals of Operations Research*, 85:39–57.

Hvattum, L. M., Løkketangen, A., and Laporte, G. (2006). Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science*, 40(4):421–438.

Hvattum, L. M., Løkketangen, A., and Laporte, G. (2007). Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Networks*, 49(4):330–340.

Klapp, M. A., Erera, A. L., and Toriello, A. (2018). The dynamic dispatch waves problem for same-day delivery. *European Journal of Operational Research*, 271(2):519–534.

Klapp, M. A., Erera, A. L., and Toriello, A. (2020). Request acceptance in same-day delivery. *Transportation Research Part E: Logistics and Transportation Review*, 143:102083.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Lin, I. I. and Mahmassani, H. S. (2002). Can online grocers deliver? some logistics considerations. *Transportation Research Record*, 1817:17–24.

Løkketangen, A. and Woodruff, D. L. (1996). Progressive hedging and tabu search applied to mixed integer (0,1) multi-stage stochastic programming. *Journal of Heuristics*, 2:111–128.

Marlin, W. U. and Thomas, B. W. (2020). Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. *European Journal of Operational Research*, 285:183–195.

Mingozzi, A., Roberti, R., and Toth, P. (2013). An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing*, 25(2):193–207.

Mitrović-Minić, S. and Laporte, G. (2004). Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655.

Naccache, S., Côté, J.-F., and Coelho, L. C. (2018). The multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 269(1):353–362.

Paradiso, R., Roberti, R., Laganà, D., and Dullaert, W. (2020). An exact solution framework for multitrip vehicle-routing problems with time windows. *Operations Research*, 66(1):180–198.

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11.

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.

Ritzinger, U., Puchinger, J., and Hartl, R. F. (2016). A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231.

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.

Song, Y., Ulmer, M. W., Thomas, B. W., and Stein, W. W. (2020). Building trust in home services – stochastic team-orienteering with consistency constraints. *Transportation Science*, 54(3):565–853.

Ulmer, M. W. (2020). Dynamic pricing and routing for same-day delivery. *Transportation Science*, 54(4):855–1152.

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., and Thomas, B. W. (2020). On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, 9(2):100008.

Ulmer, M. W., Thomas, B. W., Campbell, A. M., and Woyak, N. (2021). The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. *Transportation Science*, 55(1):75–100.

Ulmer, M. W., Thomas, B. W., and Mattfeld, D. C. (2019). Preemptive depot returns for dynamic same-day delivery. *EURO Journal on Transportation and Logistics*, 8(4):327–361.

Voccia, S. A., Campbell, A. M., and Thomas, B. W. (2019). The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184.