Models and algorithms for online stochastic vehicle routing problems

Michael Saint-Guillain

Models and algorithms for online stochastic vehicle routing problems

Michael Saint-Guillain

MODELS AND ALGORITHMS FOR ONLINE STOCHASTIC VEHICLE ROUTING PROBLEMS

MICHAEL SAINT-GUILLAIN

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Applied Science in Engineering (UCLouvain), and Doctor in Computer Sciences (INSA-Lyon)

September 13, 2019

Université catholique de Louvain (UCLouvain),
Institute of Information and Communication Technologies,
Electronics and Applied Mathematics (ICTEAM)
Louvain School of Engineering,
Louvain-la-Neuve, Belgium

Institut National des Sciences Appliquées de Lyon (INSA-Lyon),
Laboratoire d'Informatique en Image
et Systèmes d'information (LIRIS),
Villeurbanne, France

Thesis Committee

Yves Deville (supervisor)	UCLouvain, Belgium
Christine Solnon (supervisor)	INSA-Lyon, France
Charles Pecheur (chairperson)	UCLouvain, Belgium
Romain Billot	IMT-Atlantique, France
Nadia Brauner	Université Grenoble Alpes, France
Sabine Limbourg	ULiège, Belgium

Michael Saint-Guillain: *Models and Algorithms for Online Stochastic Vehicle Routing Problems,* Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Applied Science in Engineering, © September 2019

ABSTRACT

What will be tomorrow's big cities objectives and challenges? Most of the operational problems from the real world are inherently subject to uncertainty, requiring the decision system to compute new decisions dynamically, as random events occur. In this thesis, we aim at tackling an important growing problem in urban context: online dynamic vehicle routing.

Applications of online vehicle routing in the society are manyfold, from intelligent on demand public transportation to sameday delivery services and responsive home healthcare. Given a fleet of vehicles and a set of customers, each being potentially able to request a service at any moment, the current thesis aims at answering the following question. Provided the current state at some moment of the day, which are the best vehicle actions such that the expected number of satisfied requests is maximized by the end of the operational day? How can we minimize the expected average intervention delays of our mobile units?

Naturally, most of the requests remain unknown until they appear, hence being revealed online. We assume a stochastic knowledge on each operational problem we tackle, such as the probability that customer request arise at a given location and a given time of the day. By using techniques from operations research and stochastic programming, we are able to build and solve mathematical models that compute near-optimal anticipative actions, such as preventive vehicle relocations, in order to either minimize the overall expected costs or maximize the quality of service.

Optimization under uncertainty is definitely not a recent issue. Thanks to evolution of both theoretical and technological tools, our ability to face the unknown constantly grows. However, most of the interesting problems remain extremely hard, if not impossible, to solve. There is still a lot of work. Generally speaking, this thesis explores some fundamentals of optimization under uncertainty. By integrating a stochastic component into the models to be optimized, we will see how it is in fact possible to create anticipation.

RÉSUMÉ

Quels seront les objectifs et défis des métropoles de demain ? La plupart des problèmes issus du monde réel sont sujets à l'inconnu, nécessitant de prendre de nouvelles décisions de façon dynamique, à la demande, en fonction des évènements aléatoires qui se réalisent. Dans cette thèse, nous nous attaquons à un problème majeur, du moins en perpectives: la gestion dynamique d'une flotte de véhicules en contexte urbain.

Les applications pratiques des tournées de véhicules à la demande sont nombreuses, incluant les transports publiques intelligents, les services de livraison, les soins et interventions à domicile, *etc.* Étant donnés une flotte de véhicules et un ensemble de clients, chacun pouvant potentiellement et à tout moment émettre une requête nécessitant une intervention, l'objectif de cette thèse est de founir une réponse à la question suivante. Étant donné l'état courant à un moment donné, comment gérer notre flotte de véhicules afin de maximiser l'espérance du nombre total de requêtes satisfaites à la fin de la journée ? Ou encore, comment minimiser l'espérance du délai moyen d'intervention de nos véhicules ?

Bien entendu, la difficulté réside en ce que la plupart des requêtes, avant d'apparaître dynamiquement, ne sont pas connues. Pour chaque problème, nous considérons qu'il nous est fourni une connaissance, sous forme d'information probabiliste, telle que la probabilité qu'une requête apparaisse à un certain endroit, et à un certain moment de la journée. Grâce à des techniques issues de la recherche opérationnelle et de la programmation stochastique, nous sommes en mesure de construire et résoudre des modèles calculant les actions anticipatives les plus adéquates, comme le redéploiement préventif des véhicules, minimisant le coût total espéré, ou encore maximisant la qualité de service.

La question de l'optimisation sous incertitude se pose depuis déjà plusieurs décennies. Grâce aux avancées à la fois théoriques et technologiques, nous sommes chaque jour un peu plus en mesure de palier à l'inconnu. Cependant, la plupart des problèmes intéressants restent extrêmement difficiles à résoudre, si ce n'est impossible. Il reste beaucoup à faire. Cette thèse explore certains concepts fondamentaux de l'optimisation sous incertitude. En intégrant une composante stochastique aux modèles à optimiser, nous verrons ensemble comment il est en effet possible de créer de l'anticipation.

ACKNOWLEDGMENTS

This work could not have been achieved without the following persons. I would like to thank my supervisors Christine Solnon and Yves Deville, for their continuous support during this research, and all the members of the thesis committee: Nadia Brauner, Sabine Limbourg, Romain Billot. I have been impressed by the number of comments they provided. As I retrieved their copies of the manuscript, after my defense, I figured out that they heavily annotated the text, proving both their interested as well as the fact that they actually read the entire document. For that, they propably deserve a medal or something.

I would also like to thank Anthony Papavasiliou for supporting this research at earlier steps. I would like to thank all the people I had the chance to have a collaboration with: Lila B.D. Lievre, Guido Perboli and more recently, Tiago Vaquero. I warmly thank all my teammates from the UCL to Mars 2018 research project, as well as the Mars Society, for all the exciting opportunities and outcomes.

I would like to thank all the folks from my office, especially the third floor, for the friendly (and sometimes, constructive!) atmosphere during those six years. Thank you Quoc Bui Trung, Minh Thanh Kong, John Aoga, Raziel Carvajal Gomez and finally, François Aubry. You worked really hard at maintaining what we set up during all these years, and what kept me motivated until the end. Not to forget Nathan Gurnet (or is it Gourmet?), we still have to conclude our empirical study of the (not so) gastronomical places in Louvain-la-Neuve, and eventually publish it.

I would like to thank everyone at the INGI department, especially the administrative and technical staff, who provided a great place to study and research. Amongst them, Vanessa Maons and Sophie Renard definitely deserve a special mention.

Finally, I would like to thank all my friends that recognize themselves in the complex and sometimes ungrateful process of supporting me, with an honorable mention awarded to Roxanne Florio.

CONTENTS

LIST OF SYMBOLS

PROBLEM GRAPH AND TIME HORIZON

$G = (V, A)$	Complete directed graph
$V = \{0\} \cup W \cup C$	Set of vertices (depot: 0)
$W = \{1, \dots, m\}$	Set of waiting vertices
$W_0 = W \cup \{0\}$	Set of waiting vertices, plus the depot
$C = \{m+1, \dots, m+n\}$	Set of customer vertices
$t_{i,j}$	Travel time along arc $(i, j) \in A$
$d^{\text{TD}}_{v,v'}(t^{\text{arr}})$	Time a vehicle must leave v to reach v' at time t^{arr} (time-dependent travel times)
K	Number of vehicles
Q	Vehicle maximum capacity
$H = \{1, \dots, h\}$	Discrete time horizon
$H_0 = H \cup \{0\}$	Discrete time horizon, plus offline time 0

SCENARIOS AND POTENTIAL REQUESTS

$R = C \times H$	Set of potential requests
$r = (c_r, \Gamma_r)$	A potential request $r \in R$ for location c_r at time Γ_r
$\Gamma_r \in H$	Reveal time of potential request $r \in R$, if it appears
$c_r \in C$	Location (vertex) of potential request $r \in R$
s_r	Service time of potential request $r \in R$
$[e_r, l_r]$	Time window of potential request $r \in R$
$q_r \in Q$	Demand of potential request $r \in R$
p_r	Probability of potential request r
$\xi \subseteq R$	A scenario describing a set of appeared request by the end of horizon H
$p(\xi)$	Probability of scenario ξ
$\xi^t \subseteq \xi$	Subset of requests appearing at time $t \in H$
$\xi^{1..t} = \xi^1 \cup \ldots \cup \xi^t$	Subset of requests appearing up to time t, with $\xi^{1..h} = \xi$

SOLUTIONS

(x, τ)	First-stage solution
$x = \{x_1, ..., x_K\}$	Set of K disjoint sequences of waiting vertices of W
$\tau : W^x \to H$	Assignment of a waiting time τ_w with every waiting vertex $w \in W^x$
$W^x \subseteq W$	Set of waiting vertices visited in first stage solution x
$\underline{on}(w)$	Arrival time unit at waiting vertx w
$\overline{on}(w)$	Last time unit at waiting vertx w
x^t	Action (decision) performed (planned) a time $t \in H_0$
X^t	Set of legal actions at time $t \in H_0$
x^0	First-stage solution (offline decisions)
$x^{i..i+k}$	Sequence of actions $\langle x^i, x^{i+1}, \ldots, x^{i+k} \rangle, i \in H_0, k \geq 0$
$(x^{0..t}, A^t)$	Current state of the second-stage solution at time $t \in H$
$A^t \subseteq \xi^{1..t}$	Set of *accepted* requests up to time t

RECOURSE STRATEGIES

$\mathcal{R}^\infty$	SS-VRPTW-CR recourse strategy with unbounded vehicle capacity
$\mathcal{R}^q$	SS-VRPTW-CR recourse strategy with bounded vehicle capacity
$\mathcal{R}^{q+}$	Improved SS-VRPTW-CR recourse strategy with bounded vehicle capacity
$\mathcal{Q}^{\mathcal{R}}(x, \tau)$	Expected cost under recourse strategy $\mathcal{R}$ and first-stage solution (x, τ)
$\mathcal{Q}^{\mathcal{R}}(x, \tau, \xi)$	Cost under recourse strategy $\mathcal{R}$ and first-stage solution (x, τ), given scenario ξ
$\perp$	The null vertex: $\forall r \in R, \mathrm{w}(r) = \perp \Leftrightarrow r$ is unassigned
$\mathrm{w}(r)$	Waiting vertex of W^x to which $r \in R$ is assigned
π_k	Set of requests assigned to vehicle k
π_w	Set of requests assigned to wait. loc. $w \in W^x$
$\mathrm{fst}(\pi_w)$	Smallest request of π_w according to $<_R$
$\mathrm{fst}(\pi_k)$	Smallest request of π_k according to $<_R$
r^-	Request of π_k which immediately precedes r *w.r.t.* $<_R$
$d^{\min}_{r,w}$	Min. departure time from $w \in W^x$ to handle r ($\mathcal{R}^\infty$, $\mathcal{R}^q$)
$d^{\max}_{r,w}$	Max. departure time from $w \in W^x$ to handle r ($\mathcal{R}^\infty$, $\mathcal{R}^q$)
$d^{\min+}_{r,w}$	Min. departure time from $w \in W^x$ to handle r ($\mathcal{R}^{q+}$)
$d^{\max+}_{r,w}$	Max. departure time from $w \in W^x$ to handle r ($\mathcal{R}^{q+}$)
$d^{\min\mathrm{TD}}_{r,w}$	Time-dependent version of $d^{\min}_{r,w}$
$d^{\max\mathrm{TD}}_{r,w}$	Time-dependent version of $d^{\max}_{r,w}$

INTEGER PROGRAMMING FORMULATIONS

x_{ij}	Binary decision variable; equals 1 iff arc $(i, j) \in V$ is part of the solution (two-index flow formulation)
y_{ik}	Binary decision variable; equals 1 iff vertex $i \in W_0$ is visited by vehicle $k \in \{1, \ldots, K\}$
x_{ijk}	Binary decision variable; equals 1 iff arc $(i, j) \in W_0^2$ is part of route $k \in \{1, \ldots, K\}$
τ_{ijk}	Binary decision variable; equals 1 iff vehicle k waits for $l \in H$ time units at vertex i

MISC.

$[a, b]$	Discrete interval $[a, b] = \{n \in \mathbb{N} : a \leq n \leq b\}$

INTRODUCTION

Online vehicle routing problems aim at modeling and solving real life problems, by considering the dynamic fashion in which pieces of data appear. Such online problems arise in many practical situations, as door-to-door or door-to-hospital transportation of elderly or disabled persons. In many countries, authorities try to set up dial-a-ride services, but escalating operating costs and the complexity of satisfying all customer demands become rapidly unmanageable for solution methods based on human choices (Cordeau and Laporte, 2003). However, such complex dynamic problems need reliable and efficient algorithms that should first be assessed on reference problems, such as the DS-VRPTW we describe later.

Let us consider the problem of managing a team of on-duty doctors, operating at patient home places during nights and week-ends. On-call periods start with all the doctors at a central depot, where each get assigned a taxi cab for efficiency and safety. Patient requests arrive dynamically. We approximate from historical data the probability that a request appears, depending on the location and the moment. Each online request comes with a hard deadline, and one must decide whether an accepted request can be satisfied in time, and how to adapt the routes accordingly. If it cannot be handled in time, the request is rejected and entrusted to an (expensive) external service provider. In such context, which is in fact an online vehicle routing problem, which decisions lead to a minimal expected number of rejected requests?

Another problem is the one of police patrol management in big cities. Amongst the various benchmarks considered in this thesis, we also study the real world problem faced by a particular subset of the police mobile units in Brussels, Belgium. Most of the units working every day for the police department are assigned to minor interventions or safety control during particular events. Our case study concerns a specific team of police units, aimed at taking action on urgent interventions, such as road traffic accidents, violence or alarms. As a consequence, these units spend their time cruising the city, waiting for intervention requests. We hence investigate on the best relocation policies for each of these mobile units, thereby minimising the expected average intervention delay.

THESIS CONTRIBUTIONS

We classify the thesis contributions in three categories: models, algorithms and benchmarks for online vehicle routing problems.

MODELS. We introduce the first two-stage model for an online vehicle routing problem with random customers, in which the requests are time-constrained. Unlike the existing models, our new problem, the SS-VRPTW-CR, stands out by respecting the nonanticipativity principle. This is achieved by modeling the fact that the presence of a customer, that is, the appearance of an online request, is revealed at a random moment instead of depending on the solution itself (*e.g.* when a vehicle visits it). Despite the latter property is well-known and studied in dynamic VRPs, the problem of evaluating the expected quality of a current solution is tricky. The lack of closed form expressions exhorts to rely on sampling methods. Our models and their associated formulae provide an alternative. Unlike sampling methods, our approach comes with strong theoretical guarantees. In addition to cover a whole new set of operational problems, which are inherently two-stage (they do not accept reoptimization during the operations), this also provides a new powerfull tool for reoptimization in dynamic time-constrained VRPs.

ALGORITHMS. We propose both exact and heuristic solution algorithms for the new two-stage stochastic VRP we introduce. In that context, algorithms are also of importance at computing expected costs, and we provide closed-form formulae to compute the necessary values in pseudo-polynomial time. In the context of dynamic online reoptimization, we provide two different novel approaches to the dynamic and stochastic VRP with time windows. The first approach is quite direct, by solving the problem using a sampling method. It is however executed while preserving the nonanticipativity principle, but lacks of strong theoretical guarantees. In the second approach, we show how our two-stage model recently introduced can be easily embedded in the reoptimization process.

BENCHMARKS. Along with the associated studies, this thesis comes with *three new realistic benchmarks* for online dynamic (and stochastic) vehicle routing problems: one in the city of Brussels, Belgium, another in Lyon, France, and finally one benchmark in Turin (Italy). Note that part of the realism of any online vehicle routing problem also depends on the description of the associated travel times and costs. A dataset of higly realistic time dependent travel times in Brussels is provided as well, resulting in a massive use of the Google Maps API. We highlight in Chapter 4 the current lack of real-world based VRP studies and applications in the literature, as well as realistic

benchmarks. By proposing a standard representation of many types of vehicle routing problems (including City VRPs), our framework allows to mitigate that issue by making easier to share, adapt, reuse and even merge data and benchmarks from different sources. Focusing on the proposed framework, the most relevant advantage resides in the possibility to generate diversified operational contexts, customized degree of dynamism, huge sets of instances and tailored classes of benchmark to test any kind of framework.

THESIS ORGANIZATION

The thesis is organized in three parts.

PART I. Provides the minimum technical background that the reader needs in order to fully understand the concepts exploited all along the thesis. In particular, basics of vehicle routing problems as well as optimization under uncertainty are presented in Chapters 1 and 2.

PART II. Focuses on two-stage models for online VRPs, that is, on optimizing the *a priori* decisions. Chapter 5 introduces a new two-stage stochastic problem, corresponding to a new range of applications from real world. Based on paper Saint-Guillain et al., 2017, it hence contributes at bridging the gap between theoretical models and real life applications, in particular in the domain of urban city VRPs. A first benchmark is presented in Chapter 6, whereas solution methods are introduced and experimentally assessed in Chapters 7 and 8. A real case study directly follows in Chapter 9, involving the management of police patrols in the city of Brussels, Belgium.

PART III. Addresses the problem of optimizing online vehicle routing decisions. Chapter 10 introduces a first sampling based solution approach for online VRPs, from publication Saint-Guillain et al., 2015. Thereafter, a hybrid version of the algorithm, based on the theoretical results of Part ii, is theferafter presented and experimentally assessed in Chapter 11. We compare our different online algorithms on various DS-VRPTW benchmarks, as well as our real-world DS-VRP-R problem of police patrol management in Brussels.

Parts are organized in chapters devoted to specific subjects. Each chapter usually ends with a section that contains the local conclusions, as well as potential future directions. A general discussion on our theoretical and empirical results, as well as future research directions, is provided in the *Conclusion and Perspectives* part at the end of the thesis. The thesis is completed with appendices providing additionnal materials, and which can be safely omitted at first reading.

*Such a **note**, appearing on the side, **summarizes** an important concept. It is handy, but always redundant.*

MARGIN NOTES. Margin notes, such as the one showed here aside, will appear throughout the thesis. Such a note always provide a handy, concise and informal summary of a concept discussed locally, usually emphasizing the importance of the latter.

Part I

BACKGROUND

1 VEHICLE ROUTING PROBLEMS

In this chapter, we briefly introduce some of the few theoretical concepts on which the current thesis is based. It should be considered as a naive, incomplete introduction to vehicle routing. We discuss the deterministic case as well as the online variants of the problem. The reader interested in a more extensive and technical introduction may rely on more specific litterature, such as Toth and Vigo, 2014, which we strongly recommand.

1.1 DETERMINISTIC VEHICLE ROUTING PROBLEMS

In the classical, deterministic vehicle routing problem (VRP or CVRP), a set of customers must be serviced by a homogeneous fleet of capacitated vehicles, while reconciling cumulated customers' demands and vehicle capacities. The vehicles are assumed to start and end their operations at a common depot. An optimal solution is then a set of planned vehicle routes, minimizing a predefined cost function, usually defined in terms of total travel distance.

*A classical **VRP** aims at designing a set of optimal vehicle routes for servicing a set of known customers.*

INTRODUCING THE BELGIAN VRP. Suppose you own a food-truck company, based in *La Louvière*, an important[1] city in Belgium. Let us consider the other most important cities in Belgium, and select some (seventeen) of these. Figure 1.1 shows all those cities on a map, including La Louvière. Currently, your small company only counts three vehicles.

Knowing that each vehicle always stays for a full day in a city, you should have enough trucks to cover all these cities in six days. You then look for the best itinaries for your trucks to visit all these cities, so that each vehicle visits at most six cities. Doing so, you will end up with the perfect planning, which you may even repeat from one week to another, while keeping your Sundays free! Figure 1.2 shows an example of such optimal solution, let us say, in terms of total traveled distance.

1 The question whether La Louvière should be considered as a noticeable, important, Belgian place, is somewhat subjective. Historically, it played a significant role for the Belgian economy, during the early stages of the industrial revolution. In our context however, let us just note that the author is attached to this city.

Figure 1.1: A selection of eighteen important cities in Belgium: Anvers, Arlon, Bastogne, Bruges, Bruxelles, Charleroi, Courtrai, Gand, Hasselt, La Louvière, Liège, Louvain, Mons, Malmedy, Namur, Ostende, Rochefort, Tournai.

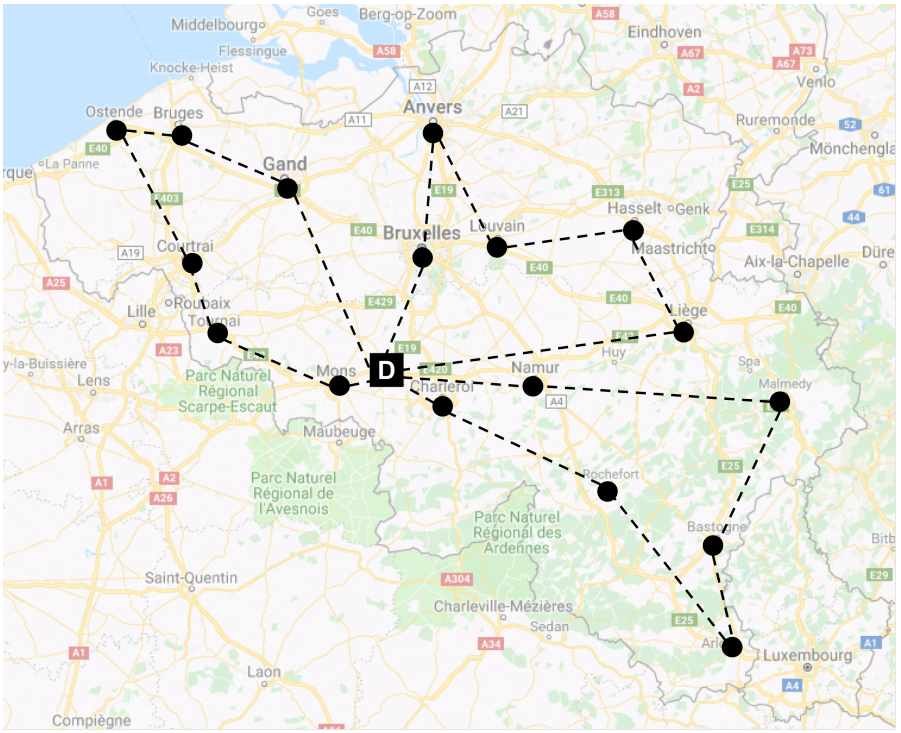


Figure 1.2: A possibly optimal solution to our Belgium VRP problem. We assume customer unit demands. Our depot is located in *La Louvière* and hosts at least three vehicles, each of maximal capacity of six.

1.1.1 *Mathematical formulations*

Here we present a couple of possible formulations of the problem. Although mathematical formulations are often related to exact solution methods (*e.g.* integer linear programming), the current section does not aim at providing an introduction on those methods, nor how to generate optimality proofs. Instead, we examine mathematical formulations in order to discuss some of the inherent properties of the problem, which are of particular interest, no matter the solution approach. The technical background required during the remaining of the section can be obtained in many volumes from the classical literature, such as Wolsey, 1998.

The capacitated vehicle routing problem is often mathematically formulated by using a so-called *two-index flow formulation* (also known as *edge formulation*). Let $V = \{0, \dots, n\}$ be the vertex set of our problem with $C = 1, \dots, n$ customer locations, where vertex 0 identifies the depot. The classical VRP is said to be symmetric. In other words, traveling from a customer i to a customer j contributes equivalently to the objective function, no matter the direction in which the edges are crossed. The edge set of the undirected complete graph, defined on V, is then $E = \{(i, j) \in V \times V : i < j\}$. To each edge is then associated a non-negative cost c_{ij}, representing the cost inquired as a vehicle travels from customer i to customer j, or equivalently from j to i. We are given K identical vehicles, each of maximal capacity Q. Finally, each customer $i \in C$ is associated a demand load q_i.

An integer decision variable x_{ij} is associated to each edge $(i, j) \in E$, indicating in the solution the number of times the corresponding edge is crossed by a vehicle (no matter the direction, *i.e.*, $x_{ij} = x_{ij}$). A possible formulation, inspired from Laporte and Nobert, 1983, is the following integer program:

$$\text{Minimize} \quad \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1.1}$$

subject to

$$\sum_{j \in V \setminus \{i\}} x_{ij} = 2 \qquad \forall\, i \in C \tag{1.2}$$

$$\sum_{i \in C} x_{0i} \leq 2K \tag{1.3}$$

$$\sum_{\substack{i \in S \\ j \in V \setminus S}} x_{ij} \geq 2r(S) \qquad \forall\, S \subseteq C, S \neq \emptyset \tag{1.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in C \tag{1.5}$$

$$x_{0i} \in \{0, 1, 2\} \qquad \forall\, i \in C \tag{1.6}$$

$$x_{ij} = x_{ij} \qquad \forall\, (i, j) \in E \tag{1.7}$$

The linear objective function (1.1) represents the cumulated traveling costs along the vehicle routes. Constraints (1.2)-(1.6) define unambigu-

ously the *solution space* of the capacitated VRP. The flow conservation constraints (1.2), or degree-two constraints, stipulate that every customer must be visited by exactly one vehicle. Constraint (1.3) connects the vehicle routes to the depot, thereby ensuring that at most K vehicles are used. Constraints (1.4) are called *rounded capacity cuts*, where $r(S) = \lceil \sum_{i \in S} q_i / Q \rceil$. They ensure both the prevention of subtours, thus playing the role of *subtour elimination constraints*, as well as the respect of the vehicle capacities. In fact, for any subset S of customers, based on the set $(i, j) : i \in S, j \in V \setminus S$ of edges having exactly one endpoint in S, constraints (1.4) ensure that there are enough vehicles interacting with S.

Constraints (1.2) to (1.4) define a $|V^2|$-dimensional convex polyhedron, in which integrality constraints (1.5)-(1.6) identify the feasible integer points. Note that, in order to allow vehicle routes visiting only one customer, edges connecting the depot can take value 2. Finally, (1.7) reflect the symmetric nature of the problem. They are not redundant, as the objective function as well as constraints (1.2) and (1.6) depend on it.

INTEGER PROGRAMMING. Once translated into an adequate language, such integer (linear) programming formulation can be almost directly solved by common IP solvers (*e.g.* Gurobi, CPlex, *etc.*). The only technical challenge concerns constraints (1.4), which are in fact exponentially many. As a consequence, in practice the complete formulation (1.2)-(1.7) cannot be directly provided to the solver. Instead, a relaxed, incomplete version of the problem is initially provided, consisting in the above formulation without any constraint from (1.4). They are then dynamically added, in a lazy fashion, each time such constraint is found to be violated, until an optimal solution is found to not violate any of those. Such approach, such as the *branch-and-cut* algorithm applying directly to our case, is known as a cutting plane algorithm (Wolsey, 1998).

ASSYMETRIC GENERAL CASE. In fact, it is easy to see that a formulation for the *assymetric VRP* can be obtained by removing (1.7) and applying some minor modifications to the constraints:

$$\text{Minimize} \quad \sum_{i,j \in V} c_{ij} x_{ij} \tag{1.8}$$

subject to

$$\sum_{j \in V \setminus \{i\}} x_{ji} = \sum_{j \in V \setminus \{i\}} x_{ij} = 1 \qquad \forall\, i \in C \tag{1.9}$$

$$\sum_{i \in C} x_{0i} = \sum_{i \in C} x_{i0} \leq K \tag{1.10}$$

$$\sum_{\substack{i \in S \\ j \in V \setminus S}} x_{ij} = \sum_{\substack{i \in S \\ j \in V \setminus S}} x_{ij} \geq r(S) \qquad \forall\, S \subseteq C, S \neq \emptyset \tag{1.11}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in V \tag{1.12}$$

In fact, whereas any instance of a symmetric VRP can be directly modeled as a assymetric VRP by using the above formulation, in practice formulation (1.2)-(1.7) is likely to be more efficiently solved to optimality. This is mainly due to the fact that, for any feasible solution (*i.e.*, admissible integer point) x in (1.2)-(1.7), there are 2^K distinct feasible solutions x' in (1.9)-(1.12). More specifically, there is one such x' per combination of distinct direction that each vehicle can choose in order to travel its route. This is illustrated in Figure 1.3. Hence, whenever it is used in order to formulate an inherently symmetric VRP, formulation (1.9)-(1.12) is said to admit 2^K *symmetries*. The concept of symmetries is an important issue, and well studied field in integer programming (see *e.g.* Margot, 2010).

*The VRP is **assymetric** in general: the travel cost between locations A to B usually differs from that of B to A. Assuming symmetric edges may however simplify the formulation.*

OTHER OBJECTIVE FUNCTIONS. Alternative objectives are often considered, such as minimizing gaz emissions for instance. In some cases, the related objective function (1.8) may hence not remain linear. For instance, the fuel consumption (and therefore the associated CO2 emission) of a vehicle along an arc (i, j) may depend on the current load of a truck as well as the distance along the arc. Note that in the latter case, the VRP becomes *assymetric*, since the direction in which the routes are traveled matters. As a consequence, formulation (1.1)-(1.6) is not valid anymore, as one needs to consider directed edges, or *arcs*, hence resulting in twice as much decision variables x_{ij}.

The scope of our introduction forces us to end here our discussion about mathematical representations of the VRP. The solution space defined by (1.9)-(1.12) is a good starting point yet, as almost all the variants to the VRP are simply generalizations of it, their respective solution space being strictly contained in the convex hull of (1.9)-(1.12). It is however important to note that alternative (in fact, infinitly many) mathematical formulations exists for the VRP, such as the capacity-indexed (Pessoa et al., 2008) or set partitioning formulations (Balinski and Quandt, 1964).

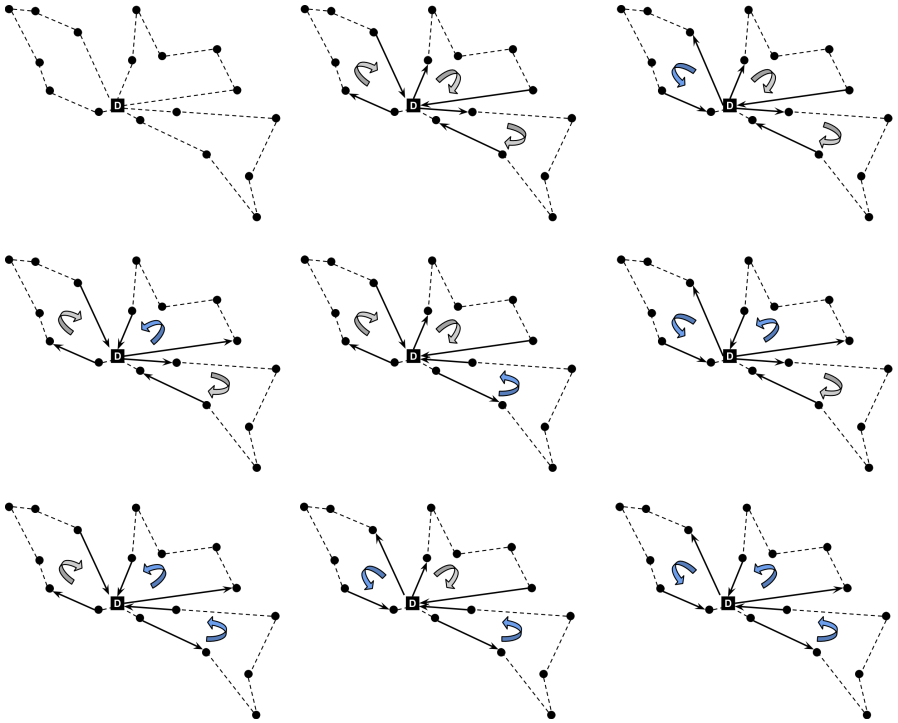


Figure 1.3: Top-left corner: a feasible solution of the symetric VRP formulation (1.2)-(1.7). To that solution correspond $2^3 = 8$ distinct (but equivalent) solutions, whenever a symetric VRP is modeled using the assymetric (but more general) formulation (1.9)-(1.12).

1.1.2 *Time constrained VRPs*

There is no time dimension involved in the classical capacitated VRP, yet in practice many operational contexts may be in fact inherently time constrained. For instance, a delivery company should be able to take into account constraints imposed by their customers, such as time windows indicating the time interval during which each customer accepts to be visited. In the case all the vehicles must terminate their routes at the depot by the end of the day, then a corresponding time window can be associated to the depot, preventing the vehicles from leaving it before the start of the operational day, or similarly, returning after the end of it.

The most natural related theoretical problem is the well known *Vehicle Routing Problem with Time Windows* (VRPTW), a generalization of the assymetric capacicated VRP (1.9)-(1.12). In the VRPTW, a set of customers must be serviced by a homogeneous fleet of capacitated vehicles, while reconciling each customer's time windows and vehicle travel times, as well as cumulated customers' demands and vehicle capacities.

A mathematical *mixed* integer linear programming formulation of the VRPTW can be found in Bard et al., 2002. In fact, the mixed term comes from the necessary continuous (*i.e.* non-integer) time

variables, used in order to keep track of the arrival times at customer vertices. How to model time in mathematical VRPTW formulations is definitely an interesting question, which has been extensively studied in the literature. However, discussing that question in more details is not relevant of the scope of the current thesis. Again, the interested reader should refer to Toth and Vigo, 2014, which provides a really great summary of the various formulations, as well as exact solution methods, for many different VRP variants (including the VRPTW).

We hence conclude here our introduction to deterministic VRPs, that is, with the deterministic VRPTW. In fact, the online VRP variants discussed throughout this thesis are all, in principle, VRPTW instances, in which some piece of data are considered uncertain. Those variants are discussed in the next section.

1.1.3 *Quality of the data and optimality proof*

Optimality always comes with strong ***assumptions****.*

Optimality proof is often considered as the Holy Grail by theoreticians. Solving a VRP by using an exact, optimal method, such as an integer linear programming solver, guarantees that the computed solution is indeed the best possible one. It is only true *with respect to the assumptions* made on the problem itself.

The quality of the data used implicitely creates assumptions, which often reveal unrealistic. For instance, because of the diffilculty and the cost of obtaining realistic driving distance matrices, many VRP studies from the scientific literature make use of Haversine distance instead. Even in works qualified as real case-studies, it often happens that distances are computed as the crow flies. Section 4.2 provides a literature review of the real case-studies conducted so far, up to our knowledge. In the latter review, we focus in particular on the realism of the associated benchmarks.

Consider our Belgium VRP problem again. Solving it optimally actually gives us two different solutions, illustrated in Figure 1.4, whether Haversine or realistic driving distances are used. Our driving distances are computed using *Google Maps Distance Matrix API*. The Haversine *"optimal"* solution reveals 40kms longer than the one obtained by using realistic driving distances, when measured using the latter distances. Naturally, 40kms is actually not a huge difference; but the problem we considered is in fact really small. The difference is however likely to grow as realistically sized problems are tackled. This is especially true in operational contexts where the distances differ significantly, *as it is the case in urban VRPs* for instance.

Consequently, when solving a combinatorial optimization problem, in particular a VRP, one should always wonder about the relevance of the optimality proof. Naturally, optimality proofs are definitly useful in some contexts, such as validating the efficiency of another, less expensive, heuristic solution method. By using small-sized prob-

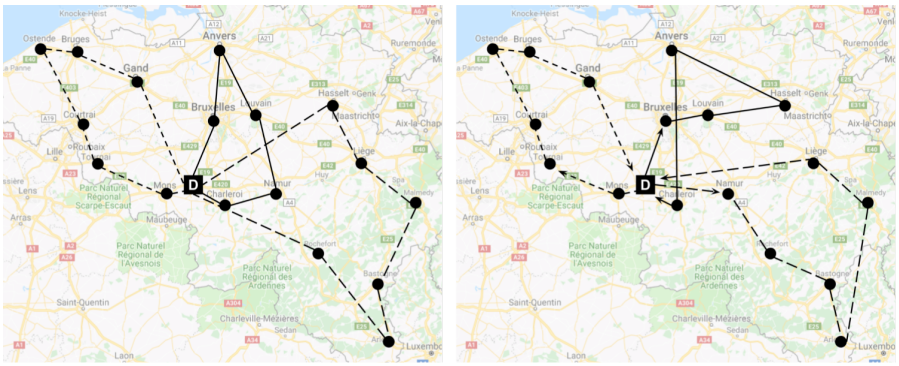


Figure 1.4: Comparison between two optimal solutions to the Belgium VRP. Left: optimal solution obtained when using Haversine distances. Right: optimal solution obtained when using driving distances.

There is no real data, only reasonably realistic assumptions.

lems, which can be solved to optimality within reasonable time and ressources, optimal solutions can in fact be exploited in order to measure optimality gaps. But considering the real, true assumptions, is simply not possible within our complicated world. In particular, real data are extremely hard to obtain. Even in our example, although we are using data that are quite realistic (GMaps assymetric driving distances), optimality may be significantly reconsidered. In fact, our travel distances are fixed, whereas real travel times are inherently time-dependent (and stochastic). The distance between two cities can indeed depend on the time of the day. As a consequence, because we did not take rush hours into account, it is not impossible that the Haversine solution reveals to be the optimal solution for the next week, whereas a solution designed by hand (*e.g.* Figure 1.2) could be that of another week.

However, even provided the best possible time-dependent data, the assumptions remain unrealistic in general, as it fails at taking uncertainty into account. Unless stochasticity is considered, that is, the fact that the data are continuously altered by random events. Taking such considerations into account plunges us into the domain of online stochastic VRPs.

1.2 ONLINE VEHICLE ROUTING

*In **online VRPs**, part of the problem data get revealed during the operations.*

Whereas deterministic VRP(TW)s assume perfect information on input data, in real-world applications some input data may be uncertain when computing a solution. The classical deterministic VRP(TW) assumes that customer demands, as well as all necessary information needed to compute a solution, are known with certainty beforehand. Unlike standard academic formulations, real world applications are usually missing part of the problem data when computing a solution. For instance, only a subset of the customer demands may be known before online execution. Missing demands therefore arrive in a dy-

	Sol. evolution during online execution	*Online requests*	*Information on online requests*
Static and deterministic	No evolution	No	N.A.
Dynamic and deterministic	Online reoptimization	Yes	No information
Dynamic and stochastic	Online reoptimization	Yes	Probabilistic knowledge
Static and stochastic	Recourse strategy	Yes	Probabilistic knowledge

Table 1.1: The four different VRP categories.

namic fashion, while vehicles are on their route. In that context, (part of) the operational decisions must be computed in light of incomplete relevant data. A solution should therefore contain operational decisions that deal with the current state of knowledge, but should ideally also be computed so that it anticipates potential unknown demands.

Dealing with online VRPs requires anticipative and/or preventive actions, computed in light of all the available relevant information.

Albeit the uncertainty can be considered for various attributes of the VRP (e.g. travel times, time windows, etc.), in this thesis we focus on situations where the customer presences are unknown a priori.

1.2.1 *Variants and taxonomy*

Following Pillac et al., 2013, VRPs can be classified in four categories, depending on two dimensions: *solution evolution* and *information quality*. Table 1.1 summarizes the different VRP categories. In order to highlight the difference with dynamic and stochastic problems, to the common appellation of "stochastic VRP (S-VRP)" we prefer the full denomination *"static and stochastic VRP"* (*SS-VRP* for short). The same taxonomy has been recently adopted by Psaraftis et al., 2015.

1.2.1.1 *Static and deterministic VRPs*

Classical VRP(TW)s can be classified as *static and deterministic*, since the solution once computed does not evolve (*i.e.*, is static) and everything is known beforehand. The information quality is said to be deterministic. This corresponds to the VRPs presented in Chapter 1.

1.2.1.2 *Dynamic and deterministic VRPs*

In D-VRP(TW)s, only part of the input is known before online execution, whilst the remaining information is revealed dynamically. Operational decisions must then be computed during execution, i.e.

"online", as the customer requests appear. It should be noted that despite the fact that the demands arrive in a dynamic fashion, the decision system is given no probabilistic knowledge on it. In Branke et al., 2005 for example, no prior knowledge is provided on the potential requests, which are then assumed to be uniformly distributed in the Euclidean plan.

1.2.1.3 *Dynamic and stochastic VRPs*

The DS-VRP(TW)s differ from their deterministic counterpart by the quality of the available information. Besides the fact that only part of the demands are known beforehand, stochastic knowledge about potential unkown requests is available so that more anticipative online operational decisions can be devised. During online execution, the current solution is recomputed at each time step, in light of known demands and current random events. This approach is called *reoptimization*, as already described in Section 2.5. A DS-VRP(TW) can therefore be formulated as a multistage stochastic program (2.11).

Examples of heuristic approaches to the DS-VRPTW can be found in Bent and Van Hentenryck, 2004b, 2007; Ichoua et al., 2006 and more recently in Saint-Guillain et al., 2015. Litterature reviews on both D-VRPs and DS-VRPs can be found in Psaraftis, 1995, Pillac et al., 2013 and recently in Ritzinger et al., 2016 and Psaraftis et al., 2015.

*Both **DS-VRPs** and **SS-VRPs** are online VRPs, for which we assume to be provided a relevant **stochastic knowledge** on the missing data.*

1.2.1.4 *Static and stochastic VRPs*

If the routes can only be adapted by following some predefined scheme, then we are facing a *Static and Stochastic VRP(TW)*, SS-VRP(TW) for short. In the SS-VRP(TW), whenever a bit of information is revealed, the current solution is adapted by applying a *recourse strategy*. The concept of recourse strategy aims at avoiding complex reoptimizations. This will be formally introduced in the next chapter. Based on the probabilistic information, we seek a first stage (also called *a priori*) solution that minimizes its a priori cost, plus the expected sum of penalties caused by the recourse strategy. In order for the evaluation function to remain tractable, the recourse strategy must be efficiently computable, hence simple enough to avoid re-optimization. SS-VRPs should therefore be thought as two-stage stochastic programs of the form (2.12), as described in Section 2.3.

__SS-VRPs__ only differ from __DS-VRPs__ by the assumptions made on the __complexity__ of the allowed __online decisions__: in SS-VRPs, reoptimization is forbidden.

The most famous SS-VRP is probably the SS-VRP-CD introduced by Bertsimas (1992). In Bertsimas, 1992, the customers are known, whereas their demands are revealed online. Two different assumptions are considered, leading to different recourse strategies, as illustrated in Fig. 1.5. In strategy **a**, each demand is assumed to be revealed when the vehicle arrives at the customer place. If the vehicle reaches its maximal capacity, then the first stage solution is adapted by adding a round trip to the depot. In strategy **b**, each demand is revealed when

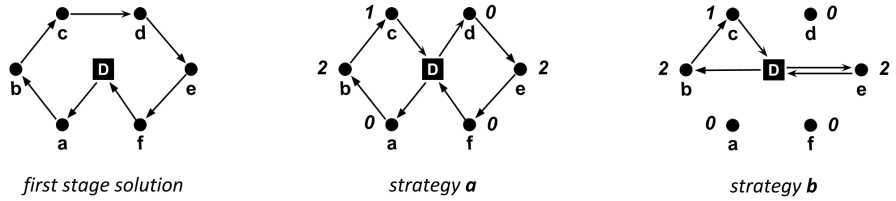


Figure 1.5: Recourse strategies for the SS-VRP with stochastic customers and demands (Bertsimas, 1992). The vehicle has a capacity of 3. The first stage solution states the *a priori* sequence of customer visits. When applying strategy ***a***, the vehicle unloads at the depot after visiting **c**. In strategy ***b***, absent customers (**a**, **d**, **f**) are skipped.

leaving the previous customer, allowing to skip customers having null demands.

In some operational contexts, it may also happen that the solution can't be adapted to restore feasibility. It typically happens when one has to deal with *time windows*: if some random event makes the vehicle arrive lately at a customer request, there is no possible recourse action that could actually undo what has been already done by the vehicle. This is called the *nonanticipativity principle*. The recourse strategy should then at least define how the overall cost is impacted. For instance, a fixed penalty cost can be inquired at each missed request.

2

OPTIMIZATION UNDER UNCERTAINTY

Whereas the terms *stochastic optimization* are sometimes used to refer to a random exploration of some solution space, in this thesis they carry a whole different meaning. We indeed consider optimization problems for which some pieces of data are uncertain, therefore described by random variables. The terms *optimization under uncertainty* tend to become often used as an alternative to *stochastic optimization*, as it avoids any confusion.

In this chapter, we present the basic concepts and notations that we use in the remaining of this document in order to model data uncertainty. We partially rely on Birge and Louveaux's notations in Birge and Louveaux, 2011, which we simplified in order to meet the scope of our discussion. The scientific litterature counts tens, maybe hundreds, of books dedicated to the vehicle routing problem (VRP). For instance, the interested reader can find in Toth and Vigo, 2014 a nice and quite recent introduction to the VRP and many of its variants, as well as a rather complete description of the existing common models and algorithms. In this thesis, we only assume the reader to be familiar with the basics of a the classical, deterministic VRP. Henceforth, the next section introduces the main concepts of stochastic programming, by extending the classical VRP to the concept of uncertainty. We will follow the example of a simple stochastic vehicle routing problem, gradually reformulated, generalized, or adapted, in order to illustrate those theoretical concepts.

2.1 INTRODUCTORY EXAMPLE: THE COURIER DELIVERY PROBLEM

Classical toy examples of stochastic programs include the *Farmer's Problem* and the *News Vendor Problem*, which are very well described in Birge and Louveaux, 2011, among others. Since this thesis is rather focused on routing problems, the motivating example described here is what we call the *Courier Delivery Problem*. Formally speaking, this stochastic problem is a direct application of the *(Static and) Stochastic Vehicle Routing Problem with random Customer*, or *SS-VRP-C* in short, a generalization of the Probabilistic Traveling Salesperson Problem (Jaillet and Odoni, 1988). The taxonomy of the different online VRPs is explained in Section 1.2.

The *Courier Delivery Problem* (CDP) consists in designing routes for the delivery of courier (or parcels). Amongst the known (finite) set of possible delivery locations, some will actually require deliveries whereas the others not. For practical reasons however, the itinary of

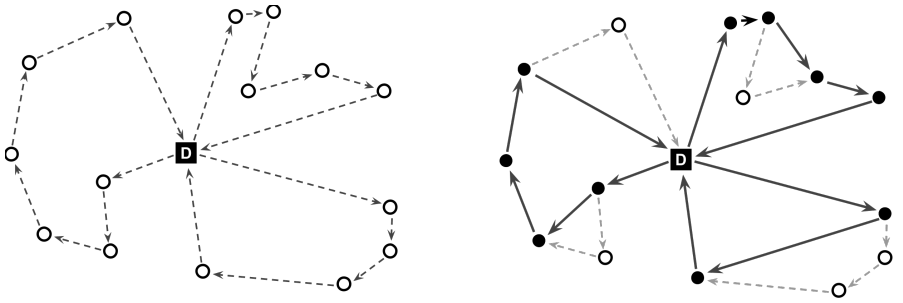


Figure 2.1: Illustrative example of the Courier Delivery Problem with three vehicles. Left: itinaries designed a priori. Right: final itinaries, when skipping the addresses for which no parcel has to be delivered (empty circles).

each delivery person cannot be recomputed everyday, but must instead be predefined, efficiently, once at all. Each deliverer then follows its itinary, *i.e.* a sequence of addresses, by skipping those that reveal to not require a service. We assume for that purpose the deliverer to be provided with an ordered pile of parcels, accorded to its itinary. Each time he (or she) completes a delivery, the deliverer henceforth simply retrieves the location of the next parcel to be delivered, if any. Figure 2.1 illustrates the operational process.

Fortunately, we assume the delivery company to be operating for long enough for us to count with accurate historical data. We can hence safely estimate the probability for each address to require a delivery. Based on that stochastic knowledge, the CDP therefore aims at designing a set of itinaries, such that each address is *a priori* part of exactly one itinary, and which minimizes the expected total distance to be traveled.

2.2 PROBABILITY SPACE, RANDOM VARIABLES AND SCENARIOS

Uncertainty is represented in terms of random experiments, with outcomes denoted by $\omega \in \Omega$, where Ω is the set of all outcomes. Only subsets of Ω, called events, are actually relevant to our problem and we denote $\mathscr{A}$ the collection of all these events. Namely, $\mathscr{A}$ is problem-dependent. Finally, to each event $A \in \mathscr{A}$ is assigned a probability $\mathrm{P}(A)$, such that $0 \leq \mathrm{P} \leq 1$, $\mathrm{P}(\emptyset) = 0$, $\mathrm{P}(\Omega) = 1$ and $\mathrm{P}(A_1 \cup A_2) = \mathrm{P}(A_1) + \mathrm{P}(A_2)$ if $A_1 \cap A_2 = \emptyset$. The triplet $(\Omega, \mathscr{A}, \mathrm{P})$ is called the *probability space* of our problem. In addition, the probability space must actually satisfy a number of conditions, that are not in the scope of this document (see *e.g.* Chung, 2001).

All the problem variables that are influenced by events in $\mathscr{A}$ are called *random variables*. Each random variable (or vector) $\boldsymbol{\xi}$ is given a cumulative probability distribution $\mathrm{P}(\boldsymbol{\xi} \leq x) = \mathrm{P}\left(\{\omega \in \Omega : \boldsymbol{\xi}(\omega) \leq x\}\right)$.

Random variables can be either discrete or continuous. In this thesis, we focus on the case where a random variable can take a countable,

usually finite, number of different values. We further define $\mathcal{S}_{\boldsymbol{\xi}} \subset \mathbb{R}^N$ as being the *support* of $\boldsymbol{\xi}$, *i.e.*, the smallest closed subset in $\mathbb{R}^N$ such that $\mathrm{P}(\mathcal{S}_{\boldsymbol{\xi}}) = 1$. Namely, $\mathcal{S}_{\boldsymbol{\xi}}$ is the set of distinct realizations of $\boldsymbol{\xi}$ having a positive probability; it is the set of all possible values $\xi_1, \ldots, \xi_{|\mathcal{S}_{\boldsymbol{\xi}}|}$ that can be assigned to the random variable $\boldsymbol{\xi}$. Generally speaking, if $\boldsymbol{\xi}$ is a random vector, then $\mathcal{S}_{\boldsymbol{\xi}}$ is the set of all its possible *scenarios*. In the Courier Delivery Problem, the uncertainty is on the set of locations requiring a delivery, which is in fact described by a random vector of binary variables, of known (estimated) probability distributions. We naturally define the probability distribution $p(\cdot)$ of a discrete random variable, or vector, $\boldsymbol{\xi}$ as:

*Provided a vector of random variables $\boldsymbol{\xi}$, a **scenario** ξ represents a possible realizations of the random vector.*

$$p(\xi_i) = \mathrm{P}(\boldsymbol{\xi} = \xi_i) = \mathrm{P}\left(\{\omega \in \Omega \,:\, \xi(\omega) = \xi_i\}\right) \quad \text{s.t.} \sum_{\xi_i \in \mathcal{S}_{\boldsymbol{\xi}}} p(\xi_i) = 1.$$

Thus, $p(\xi_i)$ is the probability that a specific scenario $\xi_i \in \mathcal{S}_{\boldsymbol{\xi}}$ realizes. Note that ξ_i and $\xi(\omega)$ both represent a scenario, that is, a possible realization of the random variable $\boldsymbol{\xi}$. Depending on the semantic context, both notations will be alternatively used through this thesis. In fact, notation $\xi_i \in \mathcal{S}_{\boldsymbol{\xi}}$ refers to an enumerative context, whereas $\xi(\omega)$ refers more generally to any possible value the random variable can take, consequently to the outcome ω of the random experiment.

Finally, let $f(x, \boldsymbol{\xi})$ be a function of some (deterministic) variable $x \in \mathbb{R}^N$ and of random variable $\boldsymbol{\xi}$. The expected value of the random function is defined as:

$$\mu = \mathrm{E}\big[f(x, \boldsymbol{\xi})\big] = \sum_{\xi_i \in \mathcal{S}_{\boldsymbol{\xi}}} p(\xi_i) \cdot f(x, \xi_i).$$

In the CDP, f may compute the value of some key performance indicator, such as the total traveled distance. The computation of its expectation is then a key part (in fact, the objective function) of the CDP, and will be discussed in Section 2.3.

2.3 TWO-STAGE STOCHASTIC PROGRAMS

One way to model uncertainty in a combinatorial optimization problem is to model it as a *two-stage stochastic program*. The idea is to divide the problem in two phases, called *stages*. In the first stage, realizations of the random variables are not known yet, but some decisions must already be taken, *a priori*; one thus tries to make the best decisions while relying on the available stochastic knowledge only, that is, on the probability distributions of the random variables. In other words, at first stage one simply bets on one particular (set of) scenario(s), and designs the best corresponding solution. Illustration on the left of Figure 2.1 shows an example of first-stage solution to the CDP, involving three deliverers. At the second-stage, all the random variables get assigned to particular realizations consecutive to the outcome,

*At first stage, a solution is designed **a priori**. At second stage, the problem data is updated, as the missing information are revealed by the **random events**.*

hence revealing the scenario we have to deal with. In the CDP, this corresponds to a subset of addresses.

According to the a priori decisions as well as the revealed scenario, further decisions, called *recourse actions*, must then be applied accordingly. The recourse actions can be a set of alternative decisions, aiming at correcting the previous ones which are not compatible anymore with the scenario. It can however also be a complementary set of decisions, or simply involve the computation of linear penalties (or rewards).

*The **recourse actions** aim at taking complementary decisions, or adapting the initial ones, in reaction to the outcomes.*

More formally, let us define $\mathcal{Q}(x, \xi(\omega)) = \mathcal{Q}(x, \omega)$, the deterministic *second-stage value function*. It is the total cost of the recourse actions against first-stage decisions x, when the scenario reveals to be $\boldsymbol{\xi} = \xi(\omega)$, which is of course only known after the random experiment. A two-stage stochastic program is the problem of finding an optimal first-stage solution, which minimizes its total first-stage cost plus the expected total cost of the second-stage recourse actions:

$$\begin{aligned} \min_{x} \quad & z = c^T x + \mathrm{E}\mathcal{Q}(x, \boldsymbol{\xi}) \\ \text{s.t.} \quad & x \in X, \end{aligned} \tag{2.1}$$

*A **two-stage stochastic program** optimizes the cost of the **a priori decisions**, plus the expected cost of the **recourse actions**.*

where X is the first-stage solution space satisfying the problem dependent constraints, and $c^T x$ is the first-stage solution cost. Suppose now that the set $Y(x, \omega)$ specifies which recourse actions are allowed, depending on the random event ω and consequently to first-stage decisions x. Namely, Y defines the second-stage solution space. Furthermore, let $c'(x, \omega)$ be a vector that defines the recourse action costs. Generally speaking, the second-stage value function can be further defined as:

$$\mathcal{Q}(x, \omega) = \min\{c'(x, \omega)^T y : y \in Y(x, \omega)\}$$

Decision vector y thus fully describes the second-stage solution, corresponding to first-stage decisions x and outcome ω. Illustration on the right of Figure 2.1 shows an example of such final solution, computed based on the a priori itiniaries (left) and following the revealed subset of addresses requiring a delivery (filled circles).

Two-stage stochastic programs can be further characterized by the shape of Y, relatively to X, and provide theoretical tools when it shows to have some desired property (*e.g.* two-stage stochastic program with fixed, complete recourse). Such mathematical analysis however falls beyond the scope of this thesis. The interested reader should refer to Birge and Louveaux, 2011 for a more detailed introduction.

2.3.1 *Deterministic equivalent program*

Depending on the problem, and especially on the operational assumptions and constraints, it may be more practical to consider function

$\mathcal{Q}(x, \omega)$ as a black box. In some situations, it even happens to this second-stage value function to be computable in polynomial time. Similarly, one step further and its expected value may also be considered as a black box:

$$\mathscr{Q}(x) = \mathrm{E}\mathcal{Q}(x, \boldsymbol{\xi})$$

called the *expected second-stage value function*, which totally captures (and hides) the stochastic dimension of the problem. The problem's operational context may henceforth justify the reformulation of (2.1) as a *deterministic equivalent program* (Wets, 1974), or DEP:

*The **expected second-stage value function** $\mathscr{Q}(x)$ may acts as a blackbox function, computing the expected recourse costs of any a priori solution x, leading to the formulation of a **determistic equivalent program**.*

$$\begin{aligned} \min_{x} \quad & z = c^T x + \mathscr{Q}(x) \\ \text{s.t.} \quad & x \in X \end{aligned} \tag{2.2}$$

Back to our example, the Courier Delivery Problem, the second-stage value function $\mathcal{Q}(x, \omega)$ simply computes, based on the a priori itinaries x, the total distance saved at skipping useless locations, when the subset of the ones requiring a delivery reveals to be $\xi(\omega)$. We immediatly notice that this simple recourse is in fact efficiently computable, and that the complexity is linear in the number of locations. Furthermore, in the CDP case, both the deterministic and the expected second-stage value functions return negative (expected) values. In fact, in the case of the CDP, mathematical programs (2.1) and (2.2) aim at minimizing the distance of the first-stage itinaries, minus the expected distance to be saved at second stage.

2.3.2 *Sampling approximations*

It clearly appears that program (2.1) can also be trivially computed by enumerating all the scenarios

$$\mathscr{Q}(x) = \sum_{\xi_i \in \mathcal{S}_\xi} p(\xi_i) \mathcal{Q}(x, \xi_i) \tag{2.3}$$

Naturally, the exponential size of $\mathcal{S}_\xi$ is a huge problem in practice.

SAMPLE AVERAGE APPROXIMATION. Most existing methods for solving two-stage (and multistage) problems are approximations, which consider a subset only of the scenarios. In the deterministic equivalent program (2.2) of a two-stage stochastic program, where we saw that $\mathscr{Q}(x) = \mathrm{E}\mathcal{Q}(x, \boldsymbol{\xi}) = \sum_{\xi_k \in \mathcal{S}_\xi} p(\xi_k) \mathcal{Q}(x, \xi_k)$. In the DEP formulation, the expected second stage value function $\mathscr{Q}$ is considered as a black box. The most natural approach, known as *Scenario Sampling Approximation* (SAA, Ahmed and Shapiro, 2002), uses Monte Carlo sampling in order to generate a limited pool of scenarios $\widetilde{\mathcal{S}}_\xi$ out of the random variable distributions. Thanks to the scenario pool, the expected second-stage value of a first-stage solution x can be approxi-

mated by measuring its average performance against each scenario of the pool:

$$\mathscr{Q}(x) \approx \frac{1}{|\widetilde{\mathcal{S}}_\xi|} \sum_{\xi_i \in \widetilde{\mathcal{S}}_\xi} \mathcal{Q}(x, \xi_i) \tag{2.4}$$

The approximation however comes with no guarantee in general, and its average accuracy directly depends on the size of $\widetilde{\mathcal{S}}_\xi$.

SCENARIO REDUCTION METHODS. Instead of generating randomly a predefined number of scenarios, so-called *scenario reduction* (or *scenario aggregation*) techniques look for a scenario subset, and a related probability measure, such that the subset is minimal and close enough to the initial distribution (Dupacova et al., 2003). This permits to restrict significantly the amount of scenarios considered in a multi-stage stochastic program, while providing guarantees on any optimal solutions to the problem.

Scenario reduction techniques are necessarily coupled with a solution method, such as the Progressive Hedging algorithm of Helgason and Wallace, 1991.

2.3.3 *Recourse strategy*

The Courier Delivery Problem (CDP) also exhibits an important property: the first-stage decisions (the a priori delivery itinaries) can only be altered by skipping locations which do not require any delivery. That, in fact, significantly reduces the range of $Y(x, \omega)$, the allowed recourse actions at second stage. In particular, it implies that there exists a function $f : X \times \mathcal{S}_\xi \to Y$, computable in linear time, determining unambiguously, and for each first-stage solution x, the final set of itinaries y according to the scenario revealed. In other words, the existence of f implies $|Y(x, \omega)| = 1$, for x and ω fixed. Such function is called a *recourse strategy*.

Provided a first-stage solution and a scenario realization, a ***recourse strategy*** *is a function that returns a unique second-stage solution.*

The recourse strategy sometimes allows an efficient computation of the expected second-stage value function $\mathscr{Q}(x)$. Suppose, in our introductory problem, that we know the distance $d_{i,j}$ between each pair of locations. Also assume, for simplicity of notation, the a priori itinary of a deliverer k to be $\{0, 1, 2, \ldots, n, 0\}$, where the depot is represented by 0. Let now $h(r)$ be the expected remaining distance from location r, if r requires a delivery, with necessarily $h(n+1) = 0$ and $h(n) = d_{n,0}$. The expected length of the kth itinary is thus given by $h^k(0)$, with:

$$h^k(r) = \sum_{r'=r+1}^{n+1} \prod_{i=r+1}^{r'-1} (1 - p_i) \cdot p_{r'} \cdot (d_{r,r'} + h(r')),$$

where p_i is the (estimated) probability that location i requires a delivery, with $p_0 = 1$, $d_{i,n+1} = d_{i,0}$. Assuming r to be part of the second-stage itinary (*i.e.* it requires a delivery), the recursion considers the

next delivery to be r', namely when the locations $\{r+1,\dots,r'-1\}$ in between are skipped.

An important consequence of the above statement is that, thanks to the recourse strategy, it provides a $\mathcal{O}(N^2)$ algorithm to compute $\mathrm{E}\mathcal{Q}(x,\boldsymbol{\xi})$ for the CDP:

$$(CDP) \qquad \mathcal{Q}(x) = \mathrm{E}\mathcal{Q}(x,\boldsymbol{\xi}) = \sum_k h^k(x),$$

whereas the expectation involves in fact 2^N scenarios when having N locations in total.

*A **recourse strategy** function, if computable in polynomial time, may lead to an expected second-stage value function that is **efficiently computable**.*

2.4 ROBUSTNESS VERSUS FLEXIBILITY

Models and methods for optimization under uncertainty aim at finding solutions that behave well under real life fluctuating data. However, what does *'behave well'* mean for a solution, within that context? As noticed in Gendreau et al., 1996b, we highlight two (very) different properties a solution can embed in order to achieve good performances in average, when performing in a stochastic context: *robustness* and *flexibility*.

A particular solution is usually feasible in a subset of the scenarios (that we hope to be significant!) but infeasible in all the remaining ones. Whenever the system outcomes a scenario which is infeasible with respect to the current solution, so-called *recourse actions* must then be applied in order to restore feasibility. By evaluating all (or a relevant subset of) the scenarios, one can then determine (or approximate) the expected recourse cost. The cheaper its expected recourse cost is, the more *flexible* the solution is. The associated models are called *stochastic programs with recourse* (SPR), and are of the form (2.1).

2.4.1 *Chance Constrained Programs*

On the other hand, a solution is said to be *robust* if it provides guaranties on the reliability of the solution, over the set of possible scenarios. Formally, a solution is said to be robust if it is guaranteed to stay feasible for at least a given subset of the scenarios. The probability of a failure, due to uncertainty, is then constrained to remain under a given level α. Such a solution does not take into account the cost of recourse actions in case of failure. Programs we obtain by considering such a model are called *chance constrained programs* (CCP), and may be formulated as suggested in King and Wallace, 2012:

$$\begin{aligned} (CCP) \qquad \min_x \quad & z = c^T x \\ \text{s.t.} \quad & \sum_{\xi_i \in S(x)} p(\xi_i) \geq \alpha \\ & \mathcal{W}(x) = \{\xi(\omega) : x \in Y(x,\omega)\}, \end{aligned} \tag{2.5}$$

*A **chance constrained program** looks for the best first-stage solution having at least probability $\boldsymbol{\alpha}$ to remain feasible at second stage.*

where $\mathcal{W}(x) = \{\xi(\omega) : x \in Y(x,\omega)\}$ is thus the set of scenarios in which the first-stage solution x remains a feasible second-stage solution.

It is not hard to think at operational contexts justifying a chance constrained formulation of the Courier Delivery Problem. Suppose that we want to privilegiate some customers, by delivering their parcels as soon as possible. An obvious naive approach is to design the a priori itinaries by simply scheduling the most privilegied ones first. Such first-stage decisions may however end up in disastrous second-stage solutions, leading the deliverers to often work terribly late. Instead, a chance constraint model can be used in order to guarantee that, for instance, no deliverer will work longer than a total duration D, with at least some probability α:

$$
\begin{aligned}
(CCP : CDP) \qquad \min_{x} \quad & z = c^T x \\
\text{s.t.} \quad & \sum_{\xi_i \in S(x)} p(\xi_i) \geq \alpha \\
& \mathcal{W}(x) = \{\xi(\omega) : h^k(x,\omega) \leq D, \forall k\},
\end{aligned}
\tag{2.6}
$$

where delivering privilegiate customers after regular ones is therefore penalized in c. Again, whereas computing the probability mass of the set $\mathcal{W}(x)$ in (2.6) involves 2^N scenario evaluations in general, in contrary the assumptions made in the case of the CDP (*i.e.*, the recourse strategy) allow to compute the robustness of a solution x in polynomial time. Define $\mathrm{P}\{h^k(r,\boldsymbol{\xi}) \leq d\}$ the probability that the remaining distance, from location r in itinary k, is of at most d. Hence,

$$
\sum_{\xi_i \in S(x)} p(\xi_i) = \prod_{k} \mathrm{P}\{h^k(0,\boldsymbol{\xi}) \leq D\}
$$

since all the k random variables $h^k(r,\boldsymbol{\xi})$ are independent, with the recursion

$$
\mathrm{P}\{h^k(r,\boldsymbol{\xi}) \leq d\} = \sum_{r'=r+1}^{n+1} \prod_{i=r+1}^{r'-1} (1-p_i) \cdot p_{r'} \cdot \mathrm{P}\{h^k(r,\boldsymbol{\xi}) \leq d - d_{r,r'}\}
$$

having the base cases $\mathrm{P}\{h^k(n,\boldsymbol{\xi}) \leq d\} = 1$ if $d_{n,0} \leq d$, zero otherwise, and $\mathrm{P}\{h^k(n+1,\boldsymbol{\xi}) \leq d\} = 1$, naturally.

2.4.2 *Mixed formulations*

Considering either pure two-stage stochastic or pure chance constrained programs may sometimes reveals as not perfectly appriopriate, depending on the context and performance requirements. In addition to imposing a chance constraint on the durations of the CDP

itinaries, similarly to (2.6), the company may also want to optimize an expected quality of service $\mathscr{Q}(x)$:

$$(CDP') \qquad \min_{x} \quad z = \mathscr{Q}(x) \tag{2.7}$$
$$\text{s.t.} \quad \sum_{\xi_i \in S(x)} p(\xi_i) \geq \alpha$$
$$\mathcal{W}(x) = \{\xi(\omega) : h^k(x,\omega) \leq D, \forall k\},$$

where $\mathscr{Q}(x)$ could be the average expected delivery time of a premium customer, for instance. Chance constraints could then be too restrictive, if an adequate and sound α level cannot be determined a priori, or if our courier delivery company is willing to improve the expected customer satisfaction, at the expense of paying occasional extra working hours. A bi-objective formulation such as

$$(CDP'') \qquad \min_{x} \quad \Big(\mathscr{Q}(x)\ ,\ 1 - r(x)\Big) \tag{2.8}$$

provides a useful tradeoff between expected quality of service and expected robustness $r(x) = \sum_{\xi_i \in S(x)} p(\xi_i)$ of the solution. A set of Pareto efficient solutions could further be computed based on program (2.8), explaining the relation between these two key performance indicators.

2.4.3 *Application: Robust operations management on Mars*

We already presented the Courier Delivery Problem as an introductory application example, for both two-stage stochastic and chance constrained formulations, in the context of vehicle routing. In Chapters 5 and 9, we present a new particular two-stage stochastic VRP and a related real world case study. Pure scheduling problems, such as the jobshop scheduling problem, are well-known for their similarities with the vehicle routing problem (see *e.g.* Beck et al., 2003). In this section, we present a concrete real world application using a stochastic formulation of a jobshop scheduling problem, mixing two-stage modeling with probabilistic robustness.

Project development realized about 30% of the world gross product in 2010, whereas we estimate at *at least* 20% of it (6% of the world gross product) the waste in poor project management (Turner et al., 2010). We see hundreds of books published, every year, on the subject of project management, development process and methodologies. Yet, whereas the technology and mathematical tools now finally enable for a more formal approach of the problem, there are still very few related studies in the literature. According to Herroelen and Leus, 2005, most of the existing studies have solely been done on machine scheduling. In fact, the scheduling theory takes its historical roots in production planning problems. The operator is usually considered as

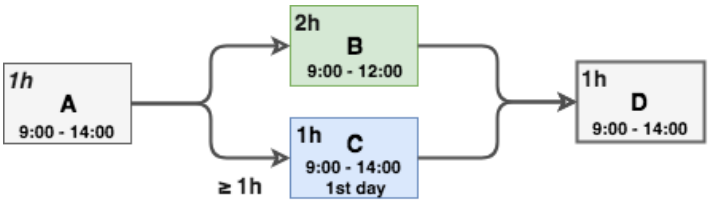


Figure 2.2: The ABCD problem. Each job has a processing time of 1 or 2 hours and a time window spanning either the entire work day (9am to 2pm) or part of it (9am to 12am). Job C must wait at least one hour after completion of A to start and must be completed during the first day.

a machine, whereas in project management it may rather represents a team member: developer, technician, scientist, *etc*.

Most of the projects are organized in tasks, or jobs, which are constrained by deadlines, conditionals, *etc*. Hence, most operations management issues can be naturally stated as scheduling problems. But solving the associated scheduling problem reveals to be of really poor interest, as soon as we take into account all the context variability, the uncertainties, that did actually justify the study of new, more flexible, development methodologies such as *Agile* (Martin, 2002). Indeed, the *Waterfall* methodology is only suited in an ideal, often unrealistic, context where the project input data (*e.g.* development time, requirements, ...) are perfectly known and remain fixed.

Let us consider the following simple scheduling problem. Four jobs $\{A, B, C, D\}$ must be scheduled to be conducted by one operator. Each job can have one or several time windows, precedence constraints and minimum transition time constraints, the latter stipulating a minimum delay between the completion time of a job and the beginning of another. We assume jobs to be atomic, so that the processing of a job cannot be split on several time windows. The horizon comprises two working days of five hours each, from 9am to 2pm. Figure 2.2 provides the remaining details of the problem, which we will refer to as the ABCD problem.

A common goal for a deterministic scheduling problem is to minimize the completion time of the entire project, which is achieved here by the sequence solution $s = \langle A, B, C, D \rangle$. It is in fact the only optimal solution: A starts at 9am, then job B from 10 to 12, followed by job C from 12 to 1pm and finally job D from 1 to 2pm. Now suppose that A's processing time is uncertain. If A reveals to require slightly more than one hour, then job B will not be completed within its time window, and is reported to the second day. Solution s is no longer feasible, as job C is not completed during the first day. Assuming the mean processing time of A to be of 1 hour, it has a significant probability to exceed it. No matter how confident we are about A's processing time and the degree of its uncertainty (*e.g.* standard deviation), a non null probability for A's processing time to reveal not to

be exactly 1 hour leads a high probability of failure (usually $\approx 50\%$). A reliable scheduler should rather suggest the more robust solution $s = \langle A, C, B, D \rangle$.

In Appendix A, based on Saint-Guillain, 2019, we compare both deterministic and mixed robust stochastic approaches to the problem of scheduling a set of scientific tasks under processing time uncertainty. While dealing with strict time windows and minimum transition time constraints, we provide closed-form expressions to compute the exact probability that a given solution remains feasible. Experiments, taking uncertainty on the stochastic knowledge itself into account, are conducted on real instances involving the constraints faced and objectives pursued during a recent two-week Mars analog mission in the desert of Utah, USA. The results reveal that, even when using very bad approximations of probability distributions, solutions computed from the stochastic models we introduce, significantly outperform the ones obtained from a classical deterministic formulation, while preserving most of the solution's quality.

2.5 MULTISTAGE STOCHASTIC PROGRAMS

Once again, we rely on our introductory Courier Delivery Problem, this time in order to motivate the concept of *multistage programming*. Suppose that our delivery company proposes an express pickup and delivery service, offering to customers who call during office hours to ship a parcel to some address, with the guarantee that it will arrive at destination the day following the call. This typically refers to an online version of the Pickup and Delivery Problem, a well known variant of the VRP. A possible, simpler, approach is to separate the problem in two different subproblems: parcel collection and parcel delivery.

The vehicle fleet is then split in both the collection vehicles, responsible for collecting the parcels, and the delivery vehicles, responsible for delivering during the next day all the parcels that have been collected (the day before). In order to meet the guaranteed quality of service, each parcel must be collected the same day as the request occured, otherwise it will not be delivered the day after.

For the need of the discussion, we focus on the collection problem online, which is in fact an *online problem*. It requires to react and commit new decisions as random events occur dynamically, during the execution of the operations. A random event here is naturally a customer that suddenly calls, asking for the collection of a parcel. Online operational decisions must be computed in order to determine which vehicle to send and when, so that some objective function (*e.g.* fuel consumption) is optimized by the end of the day.

*In an **online** (i.e. **dynamic**) problem, new information arrive as the solution is currently executed, often requiring new decisions.*

2.5.1 *Operational periods, scenarios and decision variables*

Let $H = \{1, \dots, h\}$ represent our operational time horizon, discretized in h time units, or periods. At each period t, some piece of information gets revealed, in a dynamic fashion. We note $\omega^t \in \Omega^t$ the corresponding outcome, with Ω^t the subset of possible outcomes corresponding to the random experiment at time $t \in H$.

A decision x^t (which generally corresponds to a vector of decisions) must then be taken accordingly. Then comes the next period $t+1$, with the corresponding outcome ω^{t+1}, and so on until we reach the end of the horizon.

$$\begin{array}{ccccccccc} t=0 & \rightarrow & t=1 & \rightarrow & \cdots & \rightarrow & t=h-1 & \rightarrow & t=h \\ \emptyset & & \omega^1 & & & & \omega^{h-1} & & \omega^h \\ & & \downarrow & & & & \downarrow & & \downarrow \\ x^0 & & x^1 & & & & x^{h-1} & & x^h \end{array}$$

*At the end of the operational horizon, the costs of all the intermediate a priori and online decisions are summed to obtain the total **final cost**.*

Here period $t = 0$ corresponds to the first-stage period, when decision x^0 must be taken offline, that is, before any outcome get revealed (*i.e.* before the beginning of the operations). Online decision $x^{t>0}$ must be chosen in the set $X^t = X(\{x^0, \dots, x^{t-1}\}, \omega^t)$ defining the set of legal recourse actions at period t, which generally depends on previous actions $\{x^0, \dots, x^{t-1}\}$ and the outcomes just revealed. The final total operational cost of the day is then obtained by summing the cost the a priori decision and the costs of the recourse actions:

$$z = c^0 x^0 + c^1 x^1 + \ldots + c^h x^h,$$

where $c^t = c(\{x^0, \dots, x^{t-1}\}, \omega^t)^T$ defines the recourse action costs at a given period, which may also depend on past actions and realizations. We note X^0 and c^0 the initial first-stage solution space and cost vectors, respectively, which are therefore known a priori.

EXAMPLE. In the online collection problem, ω^t naturally corresponds to the set of collection requests being revealed at time t. Decision variable x^0 may simply correspond to the number of vehicles involved in our online parcel collection problem, and c^0 the daily wage of a driver, in which case $X^0 \subseteq \mathbb{Z}$ and $c^0 \in \mathbb{R}$. A vehicle leaves the depot at $t = 1$, and is only allowed to move towards an unvisited customer that already revealed a request, if any; otherwise it simply waits at its current location. Each variable x^t, associated to the online periods $t \geq 1$, could then correspond to the next destination of the vehicle, or $\emptyset$ if no decision can be determined, either because the vehicle is currently moving at time t or if there is no remaining unvisited request at that moment.

2.5.2 *Optimal decisions*

*An (a priori) optimal solution to a **multistage stochastic program** minimizes its **expected total final cost**.*

In such an operational context, that involves multiple realization and decision periods, the optimal decision at first-stage is the one obtained by solving the multistage stochastic program (2.9):

$$\min_{x^0 \in X^0} c^0 x^0 + \mathrm{E}\Big[\min_{x^1 \in X^1} c^1 x^1 + \mathrm{E}\Big[\dots \mathrm{E}\Big[c^{h-1} x^{h-1} + \min_{x^{h-1} \in X^{h-1}} \mathrm{E}\mathcal{Q}(x^{h-1}, \boldsymbol{\xi}^h) \Big] \dots \Big] \Big] \tag{2.9}$$

The random variable (or vector $\boldsymbol{\xi}^h$) describes the random events that may occur at period h. More explicitely, by further expanding the expected value function $\mathrm{E}\mathcal{Q}(x^{h-1}, \boldsymbol{\xi}^h)$ of the final period, we obtain:

$$\begin{aligned} &\min_{x^0 \in X^0} c^0 x^0 + \mathrm{E}\Big[\dots \mathrm{E}\Big[\min_{x^{h-1} \in X^{h-1}} c^{h-1} x^{h-1} + \sum_{\xi_i^h \in \mathcal{S}_\xi^h} p(\xi_i) \mathcal{Q}(x^{h-1}, \xi_i) \Big] \dots \Big] \\ = &\min_{x^0 \in X^0} c^0 x^0 + \mathrm{E}\Big[\dots \mathrm{E}\Big[\min_{x^{h-1} \in X^{h-1}} c^{h-1} x^{h-1} + \sum_{\xi_i^h \in \mathcal{S}_\xi^h} p(\xi_i) \min_{x^h \in X^h} c^h x^h \Big] \dots \Big] \end{aligned} \tag{2.10}$$

where $\mathcal{S}_\xi^t$ is the set of possible realizations at period t. Consequently,

$$\mathcal{S}_\xi = \mathcal{S}_\xi^1 \times \mathcal{S}_\xi^2 \times \dots \times \mathcal{S}_\xi^h$$

is the set of all the scenarios that can possibly realize over the entire horizon, the support of $\boldsymbol{\xi} = (\boldsymbol{\xi}^1, \dots, \boldsymbol{\xi}^h)$.

EXAMPLE. In our online collection problem, $\mathcal{S}_\xi^t$ is the set of subsets of locations at which a request (for collecting a parcel) may appear at time unit t. An element of $\mathcal{S}_\xi$, a scenario, then describes a possible sequence of requests, each coming with its location and reveal time. If we let $c^t = c(\{x^0, \dots, x^{t-1}\}, \omega^t)^T$ be simply the travel distance from the location of the vehicle at time t, according to $\{x^0, \dots, x^{t-1}\}$, to any other location x^t could take the value of, then (2.9) minimizes the expectation of the total distance traveled.

2.5.3 *Scenario tree*

By further examining the structure of the stochastic program (2.10), we notice the nested expectations, and henceforth the nested summations

$$\dots \sum_{\xi_i^1 \in \mathcal{S}_\xi^1} p(\xi_i^1) \Big[\dots \sum_{\xi_i^2 \in \mathcal{S}_\xi^2} p(\xi_i^2 | \xi_i^1) \Big[\dots \sum_{\xi_i^h \in \mathcal{S}_\xi^h} p(\xi_i^h | \xi_i^1, .., \xi_i^h)[\dots] \Big] \Big],$$

from which a tree structure clearly appears. It is well known as the *scenario tree*. Each path of the tree constitutes a possible scenario realization. This is illustrated in Figure 2.3. Leaf nodes at $t = h$

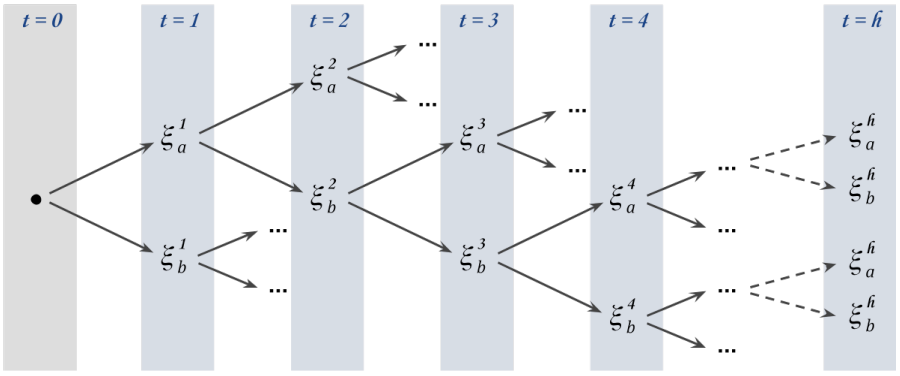


Figure 2.3: Tree structure of the problem. For simplicity, the random variable has only two possible outcomes (denoted a and b) at each period, leading to 2^{h-t} leaf nodes. Each leaf node ξ_i^h corresponds to a specific scenario.

correspond to all the possible scenario realizations, each composed of a path sequence $\xi_i^1, \ldots, \xi_i^h$ of outcomes. In general, solving (2.9) amounts, in the worst case, at following all these exponentially many different paths.

2.5.4 *Anytime formulation*

*At a **current time** period t, the current **decision** x^t is only impacted by the previous fixed decisions, and the random variables $\xi^{t+1}, \ldots, \xi^h$ that did not realize yet.*

The multistage stochastic program we formulate in (2.9) involves solving the entire problem offline, that is, from period $t = 0$. However, a noticeable property of the model is that, at a current period $t \geq 1$, the decision x^t is to be determined as previous outcomes $\omega^1, \ldots, \omega^t$ have already been observed. In the scenario tree, this corresponds to a specific node ξ_i^t, where in addition to the subtree rooted at ξ_i^t, we only need to care about the partial path from $t = 0$ to ξ_i^t.

In particular, the online problems we formulate as multistage stochastic programs of the form (2.9) do not only involve the selection of the best a priori decision x^0, but also online decisions at any time $t \geq 1$. From a purely syntactical point of view however, in order to better emphasis the online nature of the problem, (2.9) may then be reformulated as:

$$\min_{x^t \in X^t} \sum_{t'=0}^{t} c^{t'} x^{t'} + \mathrm{E}\Big[\min_{x^{t+1} \in X^{t+1}} \ldots + \mathrm{E}\Big[c^{h-1} x^{h-1} + \min_{x^{h-1} \in X^{h-1}} \mathrm{E}\mathcal{Q}(x^{h-1}, \xi^h) \Big] \ldots \Big], \tag{2.11}$$

which can be stated as the pointwise formulation of the online decision problem, for any current period $0 \leq t < h$, with $x^0, \ldots, x^{t-1}$ being known. For $t = h$, the problem naturally reduces to a classical deterministic optimization problem.

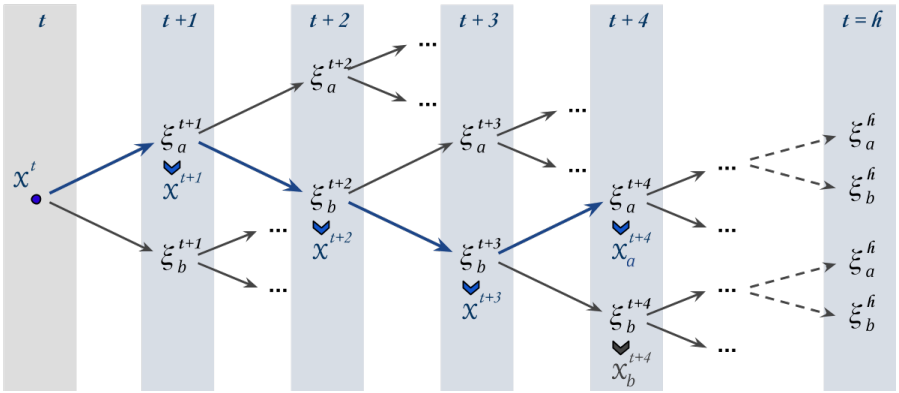


Figure 2.4: Illustration of the implicit nonanticipativity constraints on the scenario tree structure. We see that decisions x_a^{t+4} and x_b^{t+4} must necessarily share the same ancestors: they both are determined whilst considering previous decisions $x^t, \dots, x^{t+3}$ as fixed.

2.5.5 *Nonanticipativity constraints*

__Nonanticipativity constraints__ are essential to the time consistency of the recourse actions; they state that in any case, an online decision can be __determined in light of the past and current random outcomes__ only, whereas the future events remain unknown.

The multistage stochastic program (2.11) actually differs from the two-stage stochastic problem defined by:

$$\min_{x^t \in X^t} \sum_{t'=0}^{t} c^{t'} x^{t'} + \mathrm{E}\mathcal{Q}\big(x^t, (\boldsymbol{\xi}^t, \dots, \boldsymbol{\xi}^h)\big) \tag{2.12}$$

, which is in fact purely a two-stage stochastic program, equivalent to (2.1) when $t = 0$. Indeed, the nested shape of the expectations in eq. (2.11) implicitely enforces the so-called *nonanticipativity constraints* (Birge and Louveaux, 2011; Shapiro et al., 2009). Note that, especially in the older literature, these are also sometime called *implementability constraints*. At each time $t'' > t$, decision $x^{t''}$ should minimize the nested expectations over random variables $\boldsymbol{\xi}^{t''+1}, \dots, \boldsymbol{\xi}^h$, whereas at that point all the decision $x^0, \dots, x^{t''-1}$, *thus belonging to the same branch in the scenario tree*, must be considered as fixed. This is illustrated in Figure 2.4. In the multistage program defined by (2.11), an optimal decision x_a^{t+4} is associated to the node corresponding to realization ξ_a^{t+4}. In particular, the sequence of optimal decisions $x^t, \dots, x^{t+3}$ that leads to the decision x_a^{t+4} must necessarily be the same as for decision x_b^{t+4}, with possibly $x_a^{t+4} \neq x_b^{t+4}$, whereas x_b^{t+4} is also optimal but under a different context (namely, in the context of ξ_b^{t+4}). On the contrary, the two-stage version (2.12) of the problem actually relaxes

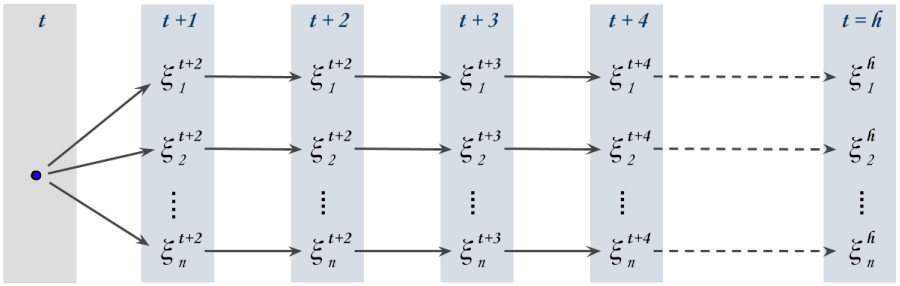


Figure 2.5: Tree structure of the problem, when scenarios are considered separately. Here $n = |\mathcal{S}_{\xi}^{t+1..h}|$. If we suppose that, similarly to Figure 2.3, the random variable of each period has only two possible realizations, we then have $n = 2^{h-t}$ possible scenarios.

the nonanticipativity constraints. By further developing (2.12), we obtain the following equivalent two-stage programs:

$$\begin{aligned} &\min_{x^t \in X^t} \sum_{t'=0}^{t} c^{t'} x^{t'} + \sum_{\xi_i \in \mathcal{S}_{\xi}^{t+1..h}} p(\xi_i)\ \mathcal{Q}(x^t, \xi_i^{t+1..h}) \\ &= \min_{x^t} \sum_{t'=0}^{t} c^{t'} x^{t'} + \sum_{\xi_i \in \mathcal{S}_{\xi}^{t+1..h}} p(\xi_i) \min_{x^{t+1},\ldots,x^h} \left\{ c^{t+1} x^{t+1} + \ldots + c^h x^h \right\} \\ &\quad \text{s.t.} \quad t \le t' \le h: \quad x^{t'} \in X\big(\{x^0, \ldots, x^{t'-1}\}, (\omega^1, \ldots, \omega^h)\big) \end{aligned} \tag{2.13}$$

where $\mathcal{S}_{\xi}^{t+1..h} = \mathcal{S}_{\xi}^{t+1} \times \ldots \times \mathcal{S}_{\xi}^{h}$ is then the set of all possible realizations, over the remaining periods, of random variables $\boldsymbol{\xi}^{t+1}, \ldots, \boldsymbol{\xi}^h$.

We remark that in program (2.12), and henceforth (2.13), the scenarios are considered in a totally independent way, and so are the associated recourse actions $x^{t+1}, \ldots, x^h$, instead of being bound altogether by the nonanticipavity constraints. This is illustrated in Figure 2.5. On the contrary, in (2.11) the scenarios are considered in a nested, tree-like, manner. As a consequence, an optimal online decision x^t for (2.12) is not necessarily optimal for (2.11), and may in fact have a higher expected value when considered in the light of the nonanticipativity constraints, which are essential to the time consistency of the recourse actions.

Nonanticipativity constraints *are implicitely part of any scenario tree.*

By explicitely adding the nonanticipativity constraints to (2.12), we hence obtain a valid *two-stage equivalent program* (2EP) to the multistage stochastic program (2.11):

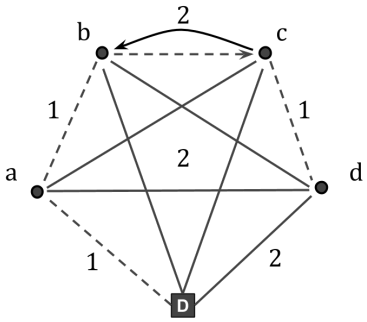
$$\min_{x^t \in X^t} \sum_{t'=0}^{t} c^{t'} x^{t'} + \sum_{\xi_k \in \mathcal{S}_{\xi}^{t+1..h}} p(\xi_k) \min_{x_k^{t+1},\ldots,x_k^h} \left\{ c^{t+1} x_k^{t+1} + \ldots + c^h x_k^h \right\} \tag{2.14}$$

$$\text{s.t.} \quad t < t' \le h,\ \forall \xi_i \in \mathcal{S}_{\xi}^{t+1..t'}:$$

$$x_i^{t'} \in X\big(\{x^0, \ldots, x^t, x_i^{t+1}, \ldots, x_i^{t'-1}\}, (\omega_i^1, \ldots, \omega_i^{t'})\big) \tag{2.15}$$

$$\forall \xi_j \in \mathcal{S}_{\xi}^{t+1..t'}: \xi_i = \xi_j \quad \Rightarrow \quad t < t'' \le t':\ x_i^{t''} = x_j^{t''} \tag{2.16}$$

where $x_i^{t''}$ is the decision associated to node $\xi_i^{t''}$ in the scenario tree, that is, the optimal decision computed in light of the stochastic knowl-



t :	0	1	2	3	4	...	h
ξ_1^t :	∅	a,d	b	c	∅	...	∅
ξ_2^t :	∅	a,d	c	b	∅	...	∅

Figure 2.6: A simple example of the impact of nonanticipativity constraints on the online parcel collection problem. The graph shows the (integer) travel times between vertices, which are always either 1 (*dashed*) or 2 (*bold*) periods and symmetric, except between vertices b and c. Finally, for simplicity we assume that there are only two, equiprobable, scenarios. The two scenarios ξ_1 and ξ_2 only differ on the moment at which requests b and c arrive.

edge about random variables $\boldsymbol{\xi}^{t''+1}, \ldots, \boldsymbol{\xi}^{h}$, and conditionally that random variables $\boldsymbol{\xi}^{t+1}, \ldots, \boldsymbol{\xi}^{t'}$ realize as $\xi_i = \left(\xi_i^{t+1}, \ldots, \xi_i^{t'}\right)$. For each subsequence of realizations $\xi_i^{t+1}, \ldots, \xi_i^{t'}$, constraints (2.15) ensure that the associated decisions are consistent with the previous decisions and the scenario in which they take place. Constraints (2.16) express the nonanticipativity property, by stating that if two subsequences of realizations $\xi_i = \left(\xi_i^{t+1}, \ldots, \xi_i^{t'}\right)$ and $\xi_j = \left(\xi_j^{t+1}, \ldots, \xi_j^{t'}\right)$ are identical, thus meaning that they both belong to the same branch in the scenario tree, then they must share the same identical decisions $\left(x_i^{t+1}, \ldots, x_i^{t'}\right) = \left(x_j^{t+1}, \ldots, x_j^{t'}\right)$.

EXAMPLE. Nonanticipativity constraints may also be of importance in the case of the online parcel collection problem. We are given one unique vehicle starting at the depot, and a set of customer vertices. Each customer may or not reveal a request (for collecting a parcel) at some (discrete) time unit $t \in H$ during the operational horizon $H = \{1, \ldots, h\}$. The vehicle leaves the depot at $t = 1$, and is only allowed to visit each customer once, and only if it has revealed a request. The vehicle simply waits at its current location if there is no such destination yet. The operational online decision x^t, for a period $t \geq 1$, then amounts at choosing the next destination amongst the set of unvisited vertices for which a request revealed at a time $t' \leq t$, if any. Figure 2.6 provides additional information. At time $t = 1$, the vehicle must leave the depot and has the choice between requests a and d, for its first destination. In light of the two equiprobable scenarios, what should be the first destination of the vehicle, in order to minimize the expected total travel time?

We will compare the two formulations (2.11) and (2.12) and show that, when taking the nonanticipativity constraints into account, the two-stage formulation (2.12) can lead to a suboptimal decision x^t.

According to (2.12), the two scenarios will be entirely known at time $t = 2$. In other words, if the vehicle travels first to a, at the time it arrives there, all the scenarios will be revealed up to $t = h$. We hence know at time $t = 1$ that, at the moment we will have to decide x^2, we will certainly be aware of either *1) a request appears instantly at b and another will appear at $t = 3$ for c* (scenario ξ_1), or *2) things will happen in the other way round*, according to the oppposite scenario ξ_2. In the second scenario, the information telling us that a request will appear soon at b allows, through $X^2 = X\big(\{x^0, x^1\}, (\omega^0, \ldots, \omega^h)\big)$ in (2.13), the vehicle to *anticipate* the next request, leaving a to reach b at $t = 3$. The total travel time is therefore 6 in both scenarios, when decision x^t is to travel to a first, henceforth being optimal with an expected cost of 6 (which is in fact the length of the shortest Hamiltonian tour).

It is however based on a very strong assumption, which is not consistent with the description of our problem. This anticipativity assumption, namely the relaxation of the nonanticipativity constraints from the initial problem, tells us that everything will be revealed at time $t = 2$. It results in an unrealistically low expected cost, based on (recourse) actions that over-anticipate the future. In reality, the true expected cost of the decision is of 7.5, as we explain now.

Let us now recalculate the expected cost (when the nonanticipativity constraints are enforced) of the decision *travel to a first*. Everything goes fine under scenario 1. Under scenario 2 however, the only request revealed at time $t = 2$ when leaving a is c, so that the vehicle has no other choice than moving towards c. It will then travel $t = 3$ from c to b, and from b to d and the depot, leading to a distance of 9. The real expected cost of action a, under (2.11) or (2.14)-(2.16), is therefore $\frac{1}{2} \cdot 6 + \frac{1}{2} \cdot 9 = 7.5$. An expected better decision is to travel to d first. By the time the vehicle reaches d, both b and c have revealed their requests, so that the shortest remaining tour from d ($\rightarrow c \rightarrow b \rightarrow a \rightarrow D$) is valid in both scenarios. As a consequence, (2.11) and (2.14)-(2.16) tell us to visit d first, with a true expected cost of 7. Instead, based on a wrong expected cost of 6 (2.12) recommends to visit a first, whereas the true expected cost of that action is of 7.5.

2.5.6 *Nonanticipativity and recourse strategies*

We already pointed out in Section 2.3 that recourse strategies can be of great practical interest, as they may allow an efficient exact computation of a first-stage solution's expected cost, at least when the strategy corresponds to the real operational context. Yet, it was only introduced in the context of two-stage stochastic programs.

A recourse strategy can also be devised for and applied to a multistage stochastic program of the form (2.9) and (2.11). Let us first introduce the *deterministic equivalent program* of (2.11):

$$\begin{aligned} \min_{x^t} \quad & z = c^0 x^0 + \ldots + c^t x^t + \mathcal{Q}(x^t) \\ \text{s.t.} \quad & x^t \in X^t \end{aligned} \tag{2.17}$$

where $\mathcal{Q}(x^t)$ is the *expected multistage value function*:

$$\mathcal{Q}(x^t) = \mathrm{E}\Big[\min_{x^{t+1} \in X^{t+1}} \ldots + \mathrm{E}\Big[c^{h-1}x^{h-1} + \min_{x^{h-1} \in X^{h-1}} \mathrm{E}\mathcal{Q}(x^{h-1}, \boldsymbol{\xi}^h)\Big] \ldots\Big],$$

which represents the expected total cost of the optimal decisions $x^{t+1}, \ldots, x^h$ that are to be commited subsequently to x^t.

In case the operational context of the problem involves a recourse strategy that explicitely forbids reoptimization, that is, if $\forall t' > t : |X^{t'}| = 1$, then $\mathcal{Q}\big(x^t, (\omega^{t+1}, \ldots, \omega^h)\big)$ is computable in linear time in h. That actually consitutes a particular degenerated multistage program, which is rather inherently two-stage, since there is actually no real online decision involved. In any case, it then may be that $\mathrm{E}\mathcal{Q}\big(x^t, (\boldsymbol{\xi}^t, \ldots, \boldsymbol{\xi}^h)\big)$ is then efficiently computable, in *(pseudo-)* polynomial time. Now suppose that the recourse strategy, as defined by $X^{t+1}, \ldots, X^h$, respects the nonanticipativity constraints. If so, $\mathcal{Q}(x^t)$ is then efficiently computable as well, since the recourse strategy imposes $\mathcal{Q}(x^t) = \mathrm{E}\mathcal{Q}\big(x^t, (\boldsymbol{\xi}^t, \ldots, \boldsymbol{\xi}^h)\big)$.

2.6 RECOURSE STRATEGIES VERSUS ONLINE REOPTIMIZATION

As highlighted in Van Hentenryck and Bent, 2009, two-stage stochastic programming considers uncertainty in the data (e.g. in the customer demands or travel times) in order to compute robust and/or flexible a priori solution rather than focus on operational decisions.

Some problems are inherently two-stages, such as the Farmer's Problem (Birge and Louveaux, 2011), whereas for other problems the distinction between two-stage or multistage may be quite ambiguous. In fact, online vehicle routing problems are rather inherently multistage. Yet, by imposing a *recourse strategy* to our introductory Courier Delivery Problem (CDP), we formulated it in a two-stage fashion, hence putting all the emphasis on the a priori decisions, from which the subsequent online actions will strictly follow. The choice for such a two-stage model, and furthermore a recourse strategy, may result from contextual constraints. In the CDP, it may simply happen that there is no possibility for the company to equip all the vehicles with the necessary location and communication devices. It may also be consecutive of even more practical considerations, as for instance the preference of the deliverers for a fixed, predefined and well-known, itinary.

In contrast, a multistage stochastic program defines a *reoptimization* process, involving online decisions. It considers uncertainty on the variables, as for instance which request to serve next in a VRP according to the current realizations. Operational decisions are at therefore at the core of the reoptimization process. Instead of relying only on an a priori plan and a recourse strategy to solve stochastic problem instances, in multistage programming one considers the current state of the solution together with information about the future outcomes to decide, step by step and after each random variable realization, which action to consider. If for some practical or operational reasons the problem naturally admits a recourse strategy, that is, does actually not involve online decisions (*e.g.* CDP), then the multistage is degenerated and should in fact be formulated as a two-stage instead.

2.6.1 *Recourse strategies for inherently multistage problems*

Nonetheless, for inherently multistage problems, thus involving non-trivial online decisions (such as the online collection problem), imposing an appropriate recourse strategy can be beneficial. Let $X^t_{\mathcal{R}} = X_{\mathcal{R}}\big(\{x^0, \dots, x^{t-1}\}, (\omega^1, \dots, \omega^t)\big)$ be the set of online decisions allowed by the recourse strategy $\mathcal{R}$ at time t, depending on previous decisions and the current scenario. Let also $\mathscr{Q}^{\mathcal{R}}(x^t)$ be the expected cost of a decision x^t when applying the subsequent actions as prescribed by the recourse strategy. In order to be a valid recourse strategy, we must have $X^t_{\mathcal{R}} \subseteq X^t$ and $|X^t_{\mathcal{R}}| = 1$, at any period t and under any possible scenario. If, in addition, the recourse strategy is shown to respect the nonanticipativity constraints 2.16, then for any online decision x^t we have $\mathscr{Q}(x^t) \leq \mathscr{Q}^{\mathcal{R}}(x^t)$. In other words, *if $\mathscr{Q}^{\mathcal{R}}(x^t)$ can be efficiently computed, a guarantee on the true expected quality of any online decision can be efficiently computed too*.

*A **nonanticipative recourse strategy** provides a useful **lower bound** on the **expected quality** of an online decision.*

In Chapter 5, we present several recourse strategies for an online routing problem variant that generalizes our online collection problem, among others. We also show how to efficiently compute the associated expected costs $\mathscr{Q}^{\mathcal{R}}(x)$ while preserving the nonanticipativity property.

3

SCOPE OF THE THESIS

This short chapter formally states the scope of the thesis, in terms of operational contexts, by defining the input data common to all the problems we will consider. Provided these input data, as well as necessary assumptions on the nature of the operational decisions, the current thesis can be unambiguously divided in two different parts: Part ii and Part iii.

PROBLEM INPUT DATA

Recall that online vehicle routing problems aim at modeling and solving real life problems, by considering uncertainty on the data. In both of the abovementioned cases (the SS-VRPTW and the DS-VRPTW), the problem input, the data, are actually exactly the same. Only the operational assumptions differ, and consequently, the way the problem is solved.

In this thesis, we focus on cases where the customer presence, or similarly their online requests, is unknown a priori. However, an online request comes with a quality of service, usually in terms of intervention delay, which must be guaranteed, using time windows for instance.

Finally, we assume to be provided with some probabilistic knowledge on the missing data. In fact, the probability distributions can be in many situations approximated from historical data.

INPUT DATA. We consider a complete directed graph $G = (V, A)$ and a discrete time horizon $H = [1, h]$, where the interval $[a, b]$ denotes the set of all integer values i such that $a \leq i \leq b$. We note $H_0 = H \cup \{0\}$. To each arc $(i, j) \in A$ is associated a travel time $t_{i,j} \in \mathbb{R}_+$, with $t_{i,j} \neq t_{j,i}$ in general. The set of vertices $V = \{0\} \cup W \cup C$ is composed of a depot 0, a set of m waiting locations $W = [1, m]$, and a set of n customer vertices $C = [m + 1, m + n]$. We note $W_0 = W \cup \{0\}$ and $C_0 = C \cup \{0\}$. The fleet is composed of K identical vehicles of maximum capacity Q.

We consider the set $R = C \times H$ of potential requests, such that an element $r = (c, \Gamma) \in R$ represents a potential request which reveals to appear or not, at time $\Gamma \in H$ for customer $c \in C$. A deterministic demand $q_r \in [1, Q]$, a deterministic service duration $s_r \in H$, and a deterministic time window $[e_r, l_r]$ with $\Gamma \leq e_r \leq l_r \leq h$ are associated to each potential request r. We note p_r the probability that r appears on vertex c at time Γ and assume independence between request probabilities. When $\Gamma = 0$, r is known before the online execution and

it is said to be *offline*. When $\Gamma > 0$, r is revealed during the online execution at time Γ and it is said to be *online*. Although our formalism imposes $\Gamma \geq 1$ for all potential requests, in practice a request may be known with probability 1, leading to a deterministic request. Also, p_r can be equal to zero for a specific request r. Finally, two or more different customers in C can share the same geographical location, making it possible to consider different types of requests in terms of deterministic attributes.

To simplify notations, a request $r = (c, \Gamma)$ may be written in place of its own vertex c. For instance, the distance $t_{v,c}$ may also be written as $t_{v,r}$. Furthermore, we use Γ_r to denote the reveal time of a request $r \in R$ and c_r for its customer vertex.

SCENARIO REALIZATION. Let $\xi \subseteq R$ be the set of requests that are found to appear by the end of the horizon H. The set ξ is also called a *scenario*. We note $\xi^t \subseteq \xi$, the set of requests appearing at time $t \in H$, *i.e.*, $\xi^t = \{r \in \xi : \Gamma_r = t\}$. We note $\xi^{1..t} = \xi^1 \cup \ldots \cup \xi^t$, the set of requests appearing up to time t, with $\xi^{1..h} = \xi$. A sequence $\langle x^i, x^{i+1}, \ldots, x^{i+k} \rangle, k \geq 0$ is noted $x^{i..i+k}$, and the concatenation of two sequences $x^{i..j}$ and $x^{j+1..k}, i \leq j < k$ is noted $x^{i..j}.x^{j+1..k}$.

OPERATIONAL ASSUMPTIONS: ONLINE DECISIONS

In order to handle new customers who appear dynamically, the current solution must be adapted, hence computing online decisions, as such random events occur. Depending on the operational context, we distinguish two fundamentally different assumptions.

- If the routes can only be adapted by following some predefined scheme, then we are facing a *Static and Stochastic VRP(TW)*: SS-VRP(TW).
- If the currently unexecuted part of the solution can be arbitrarily redesigned, then we are facing a *Dynamic and Stochastic VRP(TW)*: DS-VRP(TW).

In a SS-VRPTW, whenever a piece of information is revealed, the current solution is adapted by applying a *recourse strategy*, as described in sections 2.3. Based on the probabilistic information, we seek a first stage (also called *a priori*) solution that minimizes its a priori cost, plus the expected sum of penalties caused by the recourse strategy. In our case, penalties are expressed in terms of missed requests or intervention delays. In order for the evaluation function to remain tractable, the recourse strategy must be efficiently computable. In other words, it must be simple enough to avoid re-optimization.

In the DS-VRPTW case, the solution is adapted by re-optimizing the new current problem while fixing the executed partial routes. Naturally, as the new problem is NP-hard in general, approximation

approaches such as Approximate Dynamic Programming (see *e.g.* Maxwell et al., 2010; Schmid, 2012) or (meta-)heuristics (see for example Bent and Van Hentenryck, 2007; Ichoua et al., 2006) are preferred.

In the next chapter, we provide a literature review of both cases. Part ii of the thesis is devoted to the study of the SS-VRPTW. Part iii is then devoted to the study of the DS-VRPTW.

4

LITERATURE REVIEW

This chapter provides a specific, non-exhaustive, literature review on the subject of online VRPs, whenever the uncertainty is related to the customer (or request) presence (or appearance). The static and stochastic case is discussed in Section 4.1, whereas a review on the DS-VRPTW is provided in Section 4.2. Finally, the current chapter concludes with a literature review on an important subject, although still rarely considered: the availability of real world benchmarks, for the DS-VRPTW as well as other online VRPs related to real city logistic problems.

4.1 STATIC AND STOCHASTIC VRP'S

In this section we do not consider dynamic VRPs and rather focus on existing studies that have been carried on *static and stochastic VRPs*. Specific literature reviews on the SS-VRP may be found in Bertsimas and Simchi-Levi, 1996; Campbell and Thomas, 2008a; Cordeau et al., 2007; Gendreau et al., 1996b and more recently in Berhan et al., 2014; Gendreau et al., 2016; Kovacs et al., 2014; Toth and Vigo, 2014. According to Pillac et al., 2013, the most studied cases in SS-VRPs are:

- Stochastic customers (SS-VRP-C), where customer presences are described by random variables;
- Stochastic demands (SS-VRP-D), where all customers are present but their demands are random variables; see for instance Christiansen and Lysgaard, 2007; Dror et al., 1989; Gauvin et al., 2014; Laporte et al., 2002; Mendoza and Castanier, 2011; Morales, 2006; Secomandi, 2000; Secomandi and Margot, 2009;
- Stochastic times (SS-VRP-T), where either travel and/or service times are random variables; see for instance Kenyon and Morton, 2003; Laporte et al., 1992; Li et al., 2010; Verweij et al., 2003.

Since the SS-VRPTW-CR belongs to the first category, we focus this review on customer presence uncertainty only.

4.1.1 *SS-TSP-C*

The Traveling Salesman Problem (TSP) is a particular case of the VRP with only one uncapacitated vehicle. The first study on SS-VRP is due to Bartholdi III et al., 1983, who considered *a priori* solutions to daily food delivery. Jaillet, 1985 formally introduced the TSP with stochastic

Customers (SS-TSP-C), a.k.a. the probabilistic TSP (PTSP) or TSPSC in the literature, and provided mathematical formulations and a number of properties and bounds of the problem (see also Jaillet, 1988). In particular, he showed that an optimal solution for the deterministic problem may be arbitrarily bad in case of uncertainty. Laporte et al., 1994 developed the first exact solution method for the SS-TSP-C, using the integer L-shaped method for two-stage stochastic programs proposed in Laporte and Louveaux, 1993 to solve instances up to 50 customers. Heuristics for the SS-TSP-C have then been proposed by Jezequel, 1985, Rossi and Gavioli, 1987, Bertsimas, 1988, Bertsimas and Howell, 1993, Bertsimas et al., 1995, Bianchi et al., 2005 and Bianchi and Campbell, 2007 as well as meta-heuristics such as simulated annealing (Bowler et al., 2003) or ant colony optimization (Bianchi et al., 2002). Braun and Buhmann, 2002 proposed a method based on learning theory to approximate SS-TSP-C. A Pickup and Delivery Traveling Salesman Problem with stochastic Customers is considered by Beraldi et al., 2005, as an extension of the SS-TSP-C in which each pickup and delivery request materializes with a given probability.

Particularly close to the SS-VRPTW-CR is the SS-TSP-C with Deadlines introduced by Campbell and Thomas, 2008b. Unlike the SS-VRPTW-CR, authors assume that customer presences are not revealed at some random moment during the operations, but all at once at the beginning of the day. However, Campbell and Thomas showed that deadlines are particularly challenging when considered in a stochastic context, and proposed two recourse strategies to address deadline violations. More recently, Voccia et al., 2013 extended their work to handle time windows. They propose two recourse strategies: first, customers are visited even if their deadlines are violated; second, a customer is skipped if its deadline cannot be respected. Weyland et al., 2013 proposed heuristics for the latter problem based on general-purpose computing on graphics processing units. A recent literature review on the SS-TSP-C may be found in Henchiri et al., 2014.

4.1.2 *SS-VRP-C*

The first SS-VRP-C has been studied by Jezequel, 1985, Jaillet, 1987 and Jaillet and Odoni, 1988 as a generalization of the SS-TSP-C. The problem is defined on a graph that includes a depot and a solution exploits a fleet of capacitated vehicles. Waters, 1989 considered general integer demands and compared different heuristics. Bertsimas, 1992 considered a VRP with stochastic Customers and Demands (SS-VRP-CD). A customer demand is assumed to be revealed either when the vehicle leaves the previous customer or when it arrives at the customer's own location. Two different recourse strategies are proposed, as illustrated in Figure 1.5. For both strategies, closed-form mathematical expressions are provided to compute the expected total distance, given a

first stage solution. Gendreau et al., 1995 and Séguin, 1994 developed the first exact algorithm for solving the SS-VRP-CD for instances up to 70 customers, by means of an integer L-shaped method. Gendreau et al., 1996a later proposed a tabu search to efficiently approximate the solution. Experimentations are reported on instances with up to 46 customers. Gounaris et al., 2014 later developed an adaptive memory programming metaheuristic for the SS-VRP-C and assessed it on benchmarks with up to 483 customers and 38 vehicles.

A variant of the SS-VRPTW-C, the Courier Delivery Problem with Uncertainty, is considered in Sungur and Ren, 2010. Potential customers have deterministic soft time windows but are present probabilistically, with uncertain service times. Vehicles are uncapacitated and share a common hard deadline for returning to the depot. The objective is to construct an *a priori* solution, to be used every day as a basis which is adapted to daily customer requests. Unlike the SS-VRPTW-CR, the set of customers is revealed at the beginning of the operations. A scenario-based heuristic coupled with tabu search is used in order to maximize the coverage of customers, minimize the total time, earliness and lateness penalty, and finally maximize the similarity between the a priori solution and the actual daily routes.

4.1.3 *Other related SS-VRPs*

In the study of Heilporn et al., 2011, uncertainty is not explicitly focused on the customers' presence. They introduced the Dial-a-Ride Problem (DARP) with stochastic customer delays. The DARP is a generalization of the VRPTW that distinguishes between pickup and delivery locations and involves customer ride time constraints. Each customer is present at its pickup location with a stochastic delay. A customer is then skipped if it is absent when the vehicle visits the corresponding location, involving the cost of fulfilling the request by an alternative service (*e.g.*, a taxi). In a sense, stochastic delays imply that each request is revealed at some uncertain time during the planning horizon. That study is thus related to our problem, although in the SS-VRPTW-CR only a subset of the requests are actually revealed. Similarly, Ho and Haugland, 2011 studied a probabilistic DARP where *a priori* routes are modified by removing absent customers at the beginning of the day, and proposed local search based heuristics.

Other interesting studies are those of Novoa et al., 2006 and Lei et al., 2011, both focused on stochastic demands. In addition to a set-partitioning-based model for the SS-VRP with stochastic Demands (SS-VRP-D), Novoa et al., 2006 introduced an extended recourse strategy in which vehicles are allowed to serve customers from a failed route, before returning to the depot or in a new route after returning back to the depot. In the study of Lei et al., 2011, the authors consider the VRPTW with stochastic demands (SS-VRPTW-D). The interesting

difference with the SS-VRP-D particular case is that, because of the deterministic time windows, a failure on a route (which implies a round trip to the depot) may also imply failures on the remaining vertices on the route.

4.2 DYNAMIC AND STOCHASTIC VRP'S

The first D-VRP is proposed in Wilson and Colvin, 1977, which introduces a single vehicle Dynamic Dial-a-Ride Problem (D-DARP) in which customer requests appear dynamically. Then, Psaraftis, 1980 introduced the concept of immediate requests that must be serviced as soon as possible, implying a replanning of the current vehicle route. Complete reviews on D-VRP may be found in Pillac et al., 2013; Psaraftis, 1995. In this section, we more particularly focus on Dynamic and Stochastic VRPs (DS-VRPs).

The survey provided in Pillac et al., 2013 classifies approaches for stochastic D-VRP in two categories, either based on *stochastic modeling* or on *sampling*. Stochastic modeling approaches formally capture the stochastic nature of the problem, so that solutions are computed in the light of an overall stochastic context. Such holistic approaches usually require strong assumptions and efficient computation of complex expected values. Sampling approaches try to capture stochastic knowledge by sampling scenarios, so that they tend to be more focused on local stochastic evidences. Their local decisions however allow sample-based methods to scale up to larger problem instances, even under challenging timing constraints. One usually needs to find a good compromise between having a high number of scenarios, providing a better representation of the real distributions, and a more restricted number of these leading to less computational effort.

An original solution approach is proposed in Bent and Van Hentenryck, 2004b for the DS-VRPTW, by introducing the Multiple Scenario Approach (MSA). A key element of MSA is an adaptive memory that stores a pool of solutions. Each solution is computed by considering a particular scenario which is optimized for a few seconds. The pool is continuously populated and filtered such that all solutions are consistent with the current system state. Another important element of MSA is the *ranking function* used to make operational decisions involving idle vehicles. The authors designed 3 algorithms for that purpose:

- *Consensus* Bent and Van Hentenryck, 2004b,c selects the request that appears the most frequently as the next serviced request in the solution pool.
- *Expectation* Bent and Van Hentenryck, 2004a,c samples a set of scenarios and selects the next request to be serviced by considering its average cost on the sampled set of scenarios. Algorithm 1 (Van Hentenryck et al., 2009) depicts how it chooses the next

Algorithm 1: The ChooseRequest-ε Expectation Algorithm

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

action a^t to perform. It requires an optimization for each action $a^t \in A(\xi^t)$ and each scenario $s \in S$ (lines 3-4), which is computationally very expensive, even with a heuristic approach.

- *Regret* Bent and Van Hentenryck, 2004a; Bent et al., 2005 approximates the expectation algorithm by recognizing that, given a solution sol_s^* to a particular scenario s, it is possible to compute a good approximation of the local loss inquired by performing another action than the next planned one in sol_s^*.

Experiments, on online vehicle routing as well as online packet scheduling, showed that the reget approach outperforms consensus and expectation in most of the cases.

Quite similar to the consensus algorithm is the Dynamic Sample Scenario Hedging Heuristic introduced by Hvattum et al., 2006 for the stochastic VRP. Also, Ichoua et al., 2006 designed a Tabu Search heuristic for the DS-VRPTW and introduced a vehicle-waiting strategy computed on a future request probability threshold in the near region. Finally, Bent and Van Hentenryck, 2007 extends MSA with waiting and relocation strategies so that the vehicles are now able to relocate to promising but unrequested yet vertices. As the performances of MSA has been demonstrated in several studies Bent and Van Hentenryck, 2007; Flatberg et al., 2007; Pillac et al., 2012; Schilde et al., 2011, it is still considered as a state-of-the-art method for dealing with DS-VRPTW.

Other studies of particular interest for this thesis are Ghiani et al., 2009, on the dynamic and stochastic pickup and delivery problem, and Schilde et al., 2011, on the DS-DARP. Both consider local search based algorithms. Instead of a solution pool, they exploit one single solution that minimizes the expected cost over a set of scenarios. However, in order to limit computational effort, only near future requests are sampled within each scenario. Although the approach of Schilde et al., 2011 is similar to the one of Ghiani et al., 2009, the set of scenarios considered is reduced to one scenario. Although these papers show some similarities with the approach we propose, they do not provide any mathematical motivation and analysis of their methods.

CONCLUSION. The dynamic and stochastic VRPTW is tackled in Part iii of the current thesis. Throughout the latter part, we will insist

on the following property, which is common to most of sampling based solution approaches. Approaches based on sampling, such as MSA, usually approximate a multistage stochastic problem by relaxing the nonanticipativity constraints, hence solving (usually also approximately) a two-stage problem of the form (2.12). On the contrary, all the solution methods proposed in Part iii aim at preserving those constraints.

4.3 BENCHMARKS FOR CITY LOGISTICS

Taniguchi and Thompson, 2002 define City Logistics as "*the process for totally optimizing the logistics and transport activities by private companies with the support of advanced information systems in urban areas considering the traffic environment, its congestion, safety, and energy savings within the framework of a market economy*". As extensively reported in Cattaruzza et al., 2017, urban transportation has huge economic, social and environmental consequences. As the public and private demands for good distribution in cities are recently exploding, public authorities and enterprises require more intelligent policies. In the meanwhile, urbanization keeps growing and city inhabitants claim for improvements in the traffic of their area.

The four main stakeholders involved in urban vehicle routing are: *shippers* (retailers), *carriers* (transporters), *residents* (customers) and *administrators* (representing both city and environmental concerns). In City Logistics, the different categories of stakeholders all play an important role in urban applications, but they are rarely considered all together, leading the search to some local optimum (Kim et al., 2015). Furthermore, stakeholders usually have conflicting objectives (cooperation, competition, coopetition, collaboration), and external concerns may be of interest. For instance, the customers might be inclined to wait a little bit more if it may reduce their environmental impact.

In such settings, the transportation community has been recently devoting significant efforts to propose efficient and innovative approaches to address many types of City Logistics problems. On the other hand, a standard framework for simulating and studying the impact of optimization in City Logistics is currently missing, limiting the possibility to validate in real settings the technology transfer to industry. In particular, as highlighted by Kim et al., 2015 and observed in Section 4.3.1, there is an obvious lack of available realistic benchmark data set for the City Vehicle Routing Problem (VRP). The instances in the literature are based on the generalization of classical instances, often not created for urban applications, or on artificial data, i.e., data not coming from any historical or empirical datasets. The validation of models and methods becomes more difficult, being the results not directly compared to real or realistic settings. Even when real data

become available, some other issues come from the availability of a finite dataset, the necessity to anonymize them or to mix real data with empirical distributions.

In our opinion, that latter issue mainly comes from the following current limitations:

1. **unavailability of full data:** given a urban area, gathering the real data associated with all four stakeholders usually requires to much time and/or expertise to be actually implemented;

2. **difficulty of combining/reusing existing data:** whereas existing studies may provide realistic data involving one or more stakeholders, there is still no trivial way to combine such data from different sources.

4.3.1 *Literature review*

We focus our review on realistic City Vehicle Routing Problem (VRP) related case studies one can find in the literature up to these days. The purpose of this section is primarily to identify the scope of city VRP applications already addressed by the scientific community and secondly, the benchmarks that are still currently available in that field. This review is initially based on the excellent work Kim et al., 2015, which we restrain in order to focus on real case studies, in particular, those for which the benchmarks are still available.

The first vehicle routing case study based on real world data in urban area is due to Dulac et al., 1980, and later Chapleau et al., 1985 who addressed the problem of collecting students and routing school bus in the city of Drummondville (Canada). They proposed a heuristic method to optimize instances involving up to 99 real bus stops. In Vigo, 1996, a real-world problem of distributing pharmaceutical products in downtown Bologna (Italy) is considered and modelled as an asymmetric Capacitated Vehicle Routing Problem (CVRP). The test instances, involving up to 70 customers, are still available on the author's web page. In Chang and Yen, 2012, a case study involving a multiple Travelling Salesman Problem (TSP) with time windows is considered in the city of Taipei (Taiwan).

Variability in vehicle travel times has been studied from several aspects. In Fleischmann et al., 2004a, a Dynamic Pickup and Delivery problem is studied under dynamic travel time information. They empirically showed the importance of real-time traffic information using real-life data from a urban traffic management center in Berlin. Liao and Hu, 2011 later also considered a dynamic VRP under real-time information in Taichung City, Taiwan. Sifa et al., 2011 proposed a model for a pickup and delivery problem with time-dependent fuzzy travel times and applied it to a real-world instance in China. Hu et al., 2009 addressed a (deterministic) time-dependent VRP faced by a

food wholesaler in Beijing, China. Donati et al., 2008 also proposes a real-world case study of time-dependent VRP in Veneto (Italy). Kim et al., 2005 worked on shortest paths under non-stationary stochastic travel times, where each road segment can be either uncongested or congested at a discrete moment of the time horizon. Using real time-dependent travel time statistics on a urban road network in Detroit, they showed that significant costs savings could be achieved when considering such stochastic knowledge. In particular, in Kong et al., 2012 the authors worked on shortest paths under dynamic travel times in a downtown area in Shanghai. The raw data (1 month of taxi GPS data, 3 months of bus GPS data in Shanghai during the year 2007) are still currently available on demand to the authors.

Quak and Koster, 2009 studied the effect of governmental restrictions, such as time windows (access time regulations) and vehicle restrictions in urban areas, on the financial and environmental performances. Their research is based on a multiple a real-world case study of several Dutch retailers, which provided all organizational, flow and cost data. The impact of access time windows has also been studied by Muñuzuri et al., 2013, through a case study in Seville (Spain).

Multi-level vehicle routing problems, such as the Two-Echelon VRP (2E-VRP) introduced by Perboli et al., 2011, have also received recent interest in urban context. In Escuín et al., 2012, the authors present a real-world case study involving a time-dependent VRPTW in Zaragoza (Spain), in which customers can be either directly delivered from the classical depot or by mean of green vehicles by using intermediate urban depots.

The problem of trash collection has been studied by Santos et al., 2008, which they model as a Capacitated Arc Routing Problem (CARP) with additional constraints, such as one-way streets, demands along the arcs, prohibited turns, etc. An analysis is conducted provided real data from the city of Coimbra (Portugal). A very similar problem has also been addressed by Bautista et al., 2008 and applied to the municipality of Sant Boi de Llobregat, within the metropolitan area of Barcelona (Spain). Perrier et al., 2008 model urban snow plowing operations as a Chinese postman problem and apply it to real data from the city of Dieppe (Canada).

Vehicle breakdowns have been considered in Minis et al., 2012. Whenever a breakdown occurs, the re-planning problem is modeled as a modified Team Orienteering Problem (TOP) over the remaining customers and vehicles. They validate their heuristic method on real data from a food distribution company in Attica (Greece). Note that their benchmark is still available on demand.

Environmental-friendly decision systems in the context of (City) VRPs are increasingly studied nowadays under various names: Green Vehicle Routing Problem (GVRP), Pollution-Routing Problem (PRP), Emissions Vehicle Routing Problem (EVRP), etc. Alternative fuel vehi-

cles are considered in Erdogan and Miller-Hooks, 2012 by means of the GVRP. The objective is to minimize the total routing cost given limited refueling stations. Numerical experiments are conducted based on data from a medical textile supply company in Virginia. These data are still available under demand. In Bektas and Laporte, 2011, a realistically generated benchmark is used in order to illustrate the PRP over a set of cities in United Kingdoms. The benchmark is still available. Faulin et al., 2011; Ubeda et al., 2011 addressed real-world delivery case-studies of food delivery in Spain while also taking environmental costs into account. In Huang et al., 2012, the study a green VRP with simultaneous pickups and deliveries. Using data from a European company, they empirically show that fuel consumption and carbon emission can be significantly improved while keeping the traditional distance-minimizing objective reasonably low. A number of similar studies (e.g., Kuo, 2010; Kuo and Wang, 2011; Kwon et al., 2013; Saberi and Verbas, 2012; Xiao et al., 2012; Yang et al., 2013) have been conducted on artificially generated benchmarks, many of them based on Solomon's instances. More recently, Maggioni et al., 2014 provided a realistic benchmark for the Multi-path TSP with stochastic travel costs (Tadei et al., 2014) applied to electric and hybrid vehicles in freight distribution. In particular, Maggioni et al., 2014 propose a first example of instance generator. However, it was in a prototype form and strictly dependent on the application, lacking a global vision.

Matis, 2010 addresses a VRP in a urban area that has the particularity of involving hundreds of thousands of customers. This particular VRP sometime called Street Routing Problem, and usually solved by means of heuristic clustering methods coupled with standard VRP/TSP approaches. In these studies, the author experiments on several real-world postal delivery case studies in the Slovak Republic. A similar mail distribution problem, together with a vehicle and crew scheduling problem, was earlier addressed by Hollis et al., 2006 and applied to smaller instances (up to 339 locations) obtained from the Australia Post distribution service in Melbourne.

Another potential application related to City VRP is the Personal Rapid Transit (PRT) problem described in Schüpbach and Zenklusen, 2013. In a urban area, small automated vehicles transport passengers on demand on dedicated tracks. Whereas the idea is not recent (Fichter, 1964), there is still no computational validation based on realistic data (*e.g.*, real-world inspired origin-destination matrix, statistical knowledge, etc.).

4.3.2 *Conclusion*

Our literature review highlights that the City VRP contributions contain very few real-world based applications and the results are normally based on academic (and unrealistic) datasets. Up to our knowl-

edge, among those studies that are based on realistic data, the corresponding benchmarks are still currently available for only 6 papers: Bektas and Laporte, 2011; Erdogan and Miller-Hooks, 2012; Kong et al., 2012; Maggioni et al., 2014; Minis et al., 2012; Vigo, 1996. Moreover, these applications lack a global vision and they are scarcely repeatable in a different context. Appendix section B.1 proposes a way to overcome these issues, by introducing a simulation-optimization framework, specific to urban VRPs. Recently, the DATA2MOVE initiative started to collect data from different sources for Logistics and Supply Chain applications, but the project is still at an early stage (Woensel, 2017). A lack emerging from the literature is the identification of the main sources types, how to mix and how to interface them with one or more simulation and optimization modules in order to give flexible solutions to the stakeholders and the users. Amongst the numerous VRP classes (e.g., GVRP, CARP, VRPTW, 2E-VRP, *etc.*) addressed in the City VRP literature, most of these can actually be modelled using the proposed framework, facilitating the share, reuse and merge of existing benchmarks.

In Appendix B.2, we show how our framework can be exploited to realize a concrete case study of online freight collection in a realistic urban context, in the city of Turin (Italy). In particular, the study considers the integration of different deliveries modes (*i.e.*, cargo bikes and lockers), reflecting the current practices in the city, which are devoted to the adoption of green delivery options.

Part II

TWO-STAGE VRPTWS: OPTIMIZING A PRIORI DECISIONS

5

THE STATIC AND STOCHASTIC VRPTW WITH RANDOM CUSTOMERS AND REVEAL TIMES

This chapter is based on Saint-Guillain et al., 2017. Before the beginning of the day, probability distributions on customer data are used to compute a first-stage solution that optimizes an expected cost. Customer data are revealed online, while the solution is executed, and a recourse strategy is applied on the first-stage solution to quickly adapt it. Existing SS-VRP variants usually make a strong assumption on the time at which a stochastic customer reveals its data (*e.g.* when a vehicle arrives at the corresponding location). Instead of reoptimizing online, a so-called recourse strategy defines the way the requests are handled, whenever they appear.

In the recent review of Gendreau et al., 2016, the authors argue for new recourse strategies: with the increasing use of ICT, customer information is likely to be revealed on a very frequent basis. In this context, the chronological order in which this information is transmitted no longer matches the planned sequences of customers on the vehicle routes. In particular, the authors consider as paradoxical the fact that the existing literature on SS-VRPs with random Customers (SS-VRP-C) assumes full knowledge on the presence of customers at the beginning of the operational period.

We introduce a new SS-VRP, called *Static and Stochastic VRPTW with both random Customers and Reveal times*. The SS-VRPTW-CR considers stochastic customers with time windows and does not make any assumption on their reveal times, which are stochastic as well. Based on customer request probabilities, we look for an a priori solution composed of preventive vehicle routes, minimizing the expected number of unsatisfied customer requests at the end of the day. A route describes a sequence of strategic vehicle relocations, from which nearby requests can be rapidly reached.

*The **SS-VRPTW-CR** does not make any assumption on the moment at which random events occur.*

CONTRIBUTIONS. Up to our knowledge, previous static and stochastic VRP's studies all assume that requests are revealed at the beginning of the day, and all fail at capturing the following property: besides the stochasticity on request presence, the moment at which a request is received is stochastic as well. The SS-VRPTW-CR is the first one that actually captures this *nonanticipativity* property. However, the recourse strategy previously proposed does not take capacity constraints into account. A first contribution is to introduce a new recourse strategy that handles these constraints. We also introduce an improved recourse strategy that optimizes routes by skipping some useless parts.

*Previous models assume customer requests to be revealed at the beginning of the day. In that context, the SS-VRPTW-CR is the first model to enforce the **nonanticipativity** property.*

We introduce closed-form expressions to efficiently compute expected costs for these two new recourse strategies.

ORGANIZATION. In Section 5.1, we formally define the general SS-VRPTW-CR. In Section 5.2, we extend the recourse strategy introduced by Saint-Guillain et al., 2017 to deal with capacitated vehicles. In Section 5.3, an alternative recourse strategy involving cleverer vehicle operations is presented. The remaining chapters of the current thesis part introduce solutions methods, as well a case studies.

5.1 PROBLEM DESCRIPTION: THE SS-VRPTW-CR

The problem input data have already been introduced in Chapter 3. Our results in the context of the SS-VRPTW-CR require integer operational times. Consequently, for what concerns the SS-VRPTW and in particular within the current chapter, we assume $t_{i,j} = \lceil t_{i,j} \rceil$ and $s_{i,j} = \lceil s_{i,j} \rceil$. In practice, the current limitation can be avoided by defining a time horizon, which is necessarily discrete, being accurate enough.

5.1.1 *First-stage solution.*

The first-stage solution is computed offline, before the beginning of the time horizon. It consists of a set of K vehicle routes visiting a subset of the m waiting vertices, together with time variables denoted by τ indicating how long a vehicle should wait on each vertex. More specifically, we denote by (x, τ) a first-stage solution to the SS-VRPTW-CR, where:

- $x = \{x_1, ..., x_K\}$ defines a set of K disjoint sequences of waiting vertices of W. Each sequence $x_k = \langle w_{m_1}, ..., w_{m_k} \rangle$ is such that x_k starts and ends with 0, *i.e.*, $w_{m_1} = w_{m_k} = 0$, and each vertex of W occurs at most once in x. We note $W^x \subseteq W$, the set of waiting vertices visited in x.
- $\tau : W^x \to H$ associates a waiting time τ_w with every waiting vertex $w \in W^x$ (H is the discrete time horizon).
- for each sequence $x_k = \langle w_{m_1}, ..., w_{m_k} \rangle$, the vehicle is back at the depot before the end h of the time horizon, *i.e.*,

$$\sum_{i=1}^{k-1} t_{w_{m_i}, w_{m_{i+1}}} + \sum_{i=2}^{k-1} \tau_{w_{m_i}} \leq h$$

In other words, x defines a solution to a *Team Orienteering Problem* (TOP, see Chao et al., 1996) to which each visited location is assigned a waiting time by τ.

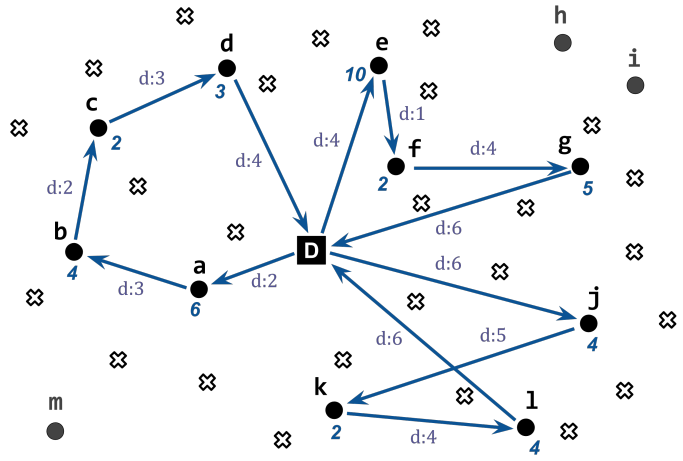


Figure 5.1: Example of first stage solution with $K = 3$ vehicles. The depot, waiting vertices and customer vertices are represented by a square, circles and crosses, respectively. Arrows represent the three vehicle routes: $x_1 = \langle D, a, b, c, d, D\rangle$, $x_2 = \langle D, e, f, g, D\rangle$ and $x_3 = \langle D, j, k, l, D\rangle$. Integers at waiting locations indicate waiting times defined by τ. Waiting vertices h, i and m are not part of the first stage solution and $W^x = \{a, b, c, d, e, f, g, j, l, k\}$. For vehicle 1, we have $\overline{on}(D) = 1$, $\underline{on}(a) = 3$, $\overline{on}(a) = 9$, $\underline{on}(b) = 12$, $\overline{on}(b) = 16$, etc.

Given a first-stage solution (x, τ), we define $on(w) = [\underline{on}(w), \overline{on}(w)]$ for each vertex $w \in W^x$ such that $\underline{on}(w)$ (resp. $\overline{on}(w)$) is the arrival (resp. departure) time at w. In a sequence $\langle w_{m_1}, ..., w_{m_k}\rangle$ in x, we then have $\underline{on}(w_{m_i}) = \overline{on}(w_{m_{i-1}}) + t_{w_{m_{i-1}}, w_{m_i}}$ and $\overline{on}(w_{m_i}) = \underline{on}(w_{m_i}) + \tau_{w_{m_i}}$ for $i \in [2, k]$ and assume that $\overline{on}(w_{m_1}) = 1$. Figure 5.1 illustrates an example of a first-stage solution for a basic SS-VRPTW-CR instance.

5.1.2 *Recourse strategy and second-stage solution.*

A recourse strategy $\mathcal{R}$ states how a second-stage solution is gradually constructed as requests are dynamically revealed. In this paragraph, we define the properties of a recourse strategy. Two recourse strategies are given in Sections 5.2 and 5.3.

A second-stage solution is incrementally constructed for each time unit by following the skeleton provided by the first-stage solution (x, τ). At a given time t of the horizon, we note $(x^{0..t}, A^t)$, the current state of the second-stage solution:

- $x^{0..t}$ defines a set of vertex sequences describing the route operations performed up to time t. Unlike x, we define $x^{0..t}$ on a graph that also includes the customer vertices. Sequences of $x^{0..t}$ must satisfy the time window and capacity constraints imposed by the VRPTW.

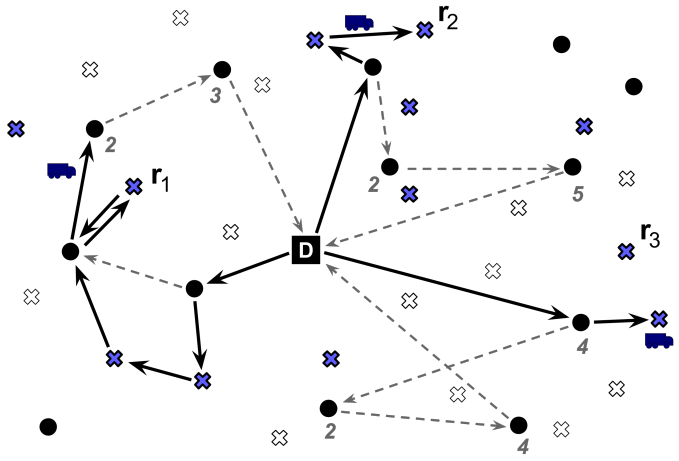


Figure 5.2: Example of partial second stage solution (plain arrows). Filled crosses are accepted requests. Some accepted requests, such as r_1, have been satisfied (or the vehicle is currently traveling towards the location, *e.g.*, r_2), while some others are not yet satisfied (*e.g.*, r_3).

- $A^t \subseteq \xi^{1..t}$ is the set of *accepted* requests up to time t. Requests of $\xi^{1..t}$ that do not belong to A^t are said to be *rejected*.

*All accepted online requests **must** eventually be served.*

We distinguish between requests that are accepted and those that are both accepted and satisfied. Up to a time t, an accepted request is said to be *satisfied* if it is visited in $x^{0..t}$ by a vehicle. Accepted requests that are not yet satisfied must be *guaranteed to be eventually satisfied* according to their time windows. Figure 5.2 illustrates an example of a second-stage solution being partially constructed at some moment of the time horizon.

Before starting the operations (at time 0), x^0 is a set of K sequences that only contain vertex 0, and $A^0 = \emptyset$. At each time unit $t \in H$, given a first-stage solution (x, τ), a previous state (x^{t-1}, A^{t-1}) of the second-stage solution, and a set ξ^t of requests appearing at time t, the new state $(x^{0..t}, A^t)$ is obtained by applying a specific recourse strategy $\mathcal{R}$:

$$(x^{0..t}, A^t) = \mathcal{R}\big((x, \tau), (x^{t-1}, A^{t-1}), \xi^t\big). \tag{5.1}$$

As explained in Chapter 2, a necessary property of a recourse strategy is that it avoids reoptimization. We consider that $\mathcal{R}$ avoids reoptimization if the computation of $(x^{0..t}, A^t)$ is achieved in polynomial, often linear, time.

We note $\text{cost}(\mathcal{R}, x, \tau, \xi) = |\xi \setminus A^h|$, the final cost of a second-stage solution with respect to a scenario ξ, given a first-stage solution (x, τ) and under a recourse strategy $\mathcal{R}$. This cost is the number of requests that are rejected at the end h of the time horizon.

5.1.3 *Optimal first-stage solution*

Given strategy $\mathcal{R}$, an optimal first-stage solution (x, τ) to the SS-VRPTW-CR minimizes the expected cost of the second-stage solution:

$$\text{(SS-VRPTW-CR)} \quad \underset{x,\tau}{\text{Minimize}} \ \mathcal{Q}^{\mathcal{R}}(x, \tau) \tag{5.2}$$

$$\text{s.t.} \quad (x, \tau) \text{ is a first-stage solution.} \tag{5.3}$$

*The SS-VRPTW-CR minimizes the **expected** amount of **rejected requests**.*

The objective function $\mathcal{Q}^{\mathcal{R}}(x, \tau)$, which is nonlinear in general, is the *expected number of rejected requests*, *i.e.*, requests that fail to be visited under recourse strategy $\mathcal{R}$ and first-stage solution (x, τ):

$$\mathcal{Q}^{\mathcal{R}}(x, \tau) = \sum_{\xi_i \subseteq R} p(\xi_i) \ \text{cost}(\mathcal{R}, x, \tau, \xi_i) \tag{5.4}$$

Since we assume independence between requests, we have $p(\xi_i) = \prod_{r \in \xi_i} p_r \cdot \prod_{r \in R \setminus \xi_i} (1 - p_r)$. Note that $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ actually represents an expected quality of service, which does not take travel costs into account. In fact, in most practical applications that could be formulated as an SS-VRPTW-CR, quality of service prevails whenever the number of vehicles is fixed, as travel costs are usually negligible compared with the labor cost of mobilized mobile units.

Formulation (5.2)-(5.3) states the problem in general terms, hiding two non-trivial issues. Given a recourse strategy $\mathcal{R}$, finding a computationally tractable way to evaluate $\mathcal{Q}^{\mathcal{R}}$ constitutes the first challenge. We address it Sections 5.2 and 5.3., based on two new recourse strategies we propose. The second problem naturally concerns the minimization problem, or how to deal with the solution space. This is addressed in Chapters 7 and 8.

NON-REJECT MODELS. Depending on the operational context, situations may arise in which one cannot reject online requests. The objective function should then be adapted, for example, by minimizing the expected average intervention delay. In Chapter 9, we show how our theoretical results for the SS-VRPTW-CR can easily be adapted and exploited in such a context, and apply it to a real case study.

5.2 RECOURSE STRATEGY WITH BOUNDED CAPACITY

In this section we introduce the recourse strategy $\mathcal{R}^q$, which is a generalization of the recourse strategy $\mathcal{R}^\infty$ introduced by Saint-Guillain et al., 2017: $\mathcal{R}^\infty$ only considers uncapacitated vehicles (*i.e.*, $Q = \infty$), whereas $\mathcal{R}^q$ considers capacitated vehicles in the context of integer customer demands. We first describe $\mathcal{R}^q$ and then describe the closed-form expression that allows us to compute the expected cost of the second-stage solution obtained when applying $\mathcal{R}^q$ to a first-stage solution.

5.2.1 *Description of $\mathcal{R}^q$*

Informally, the recourse strategy $\mathcal{R}^q$ accepts a request revealed at time t if the assigned vehicle is able to adapt its first-stage tour to visit the customer, given its set of previously accepted requests. Time window and capacity constraints should be respected, and already accepted requests should not be perturbed.

Ideally, whenever a request appears and prior to determine whether it can be accepted, a vehicle should be selected to minimize objective function (5.2). Furthermore, if several requests appear at the same time unit and amongst the subset of these that are possibly acceptable, some may not contribute optimally to (5.2). Given a set of accepted requests, the order in which they are visited also plays a critical role. Unfortunately, none of these decisions can be made optimally without reducing to a NP-hard problem. In order for $\mathcal{R}^q$ to remains efficiently computable, they are necessarily made heuristically.

In what we propose, these decisions are made beforehand. Before computing the recourse strategy and in order to avoid reoptimization, the set R of potential requests is ordered. Each potential request $r \in R$ is also preassigned to exactly one planned waiting vertex in W^x, and therefore one vehicle, based on geographical considerations.

5.2.1.1 *Request ordering*

In order to avoid reoptimization, the set R of potential requests must be ordered. This ordering is defined before computing first-stage solutions. Different orders may be considered, without loss of generality, provided that the order is total, strict, and consistent with the reveal time order; *i.e.*, $\forall r_1, r_2 \in R$, if the reveal time of r_1 is smaller than the reveal time of r_2 ($\Gamma_{r_1} < \Gamma_{r_2}$), then r_1 must be smaller than r_2 in the request order.

In this paper, we order R by increasing reveal time first, end of time window second, and lexicographic order to break further ties. More precisely, we consider the order $<_R$ such that $\forall \{r_1, r_2\} \subseteq R$, $r_1 <_R r_2$ iff $\Gamma_{r_1} < \Gamma_{r_2}$ or ($\Gamma_{r_1} = \Gamma_{r_2}$ and $l_{r_1} < l_{r_2}$) or ($\Gamma_{r_1} = \Gamma_{r_2}$, $l_{r_1} = l_{r_2}$ and r_1 is smaller than r_2 according to the lexicographic order defined over $C \times H$).

5.2.1.2 *Request assignment according to a first-stage solution*

Given a first-stage solution (x, τ), we assign each request of R either to a waiting vertex visited in x or to $\perp$ to denote that r is not assigned. We note $\mathrm{w} : R \to W^x \cup \{\perp\}$ this assignment. It is computed for each first-stage solution (x, τ) before the application of the recourse strategy. To compute this assignment, for each request r, we first compute the set W_r^x of waiting vertices which are feasible for r:

$$W_r^x = \{w \in W^x : d_{r,w}^{\min} \leq d_{r,w}^{\max}\}$$

where $d^{\min}_{r,w}$ and $d^{\max}_{r,w}$ are defined as follows. Time

$$d^{\min}_{r,w} = \max\{\underline{on}(w), \Gamma_r, e_r - t_{w,r}\}$$

is the earliest time at which the vehicle can possibly depart from waiting vertex w in order to satisfy request r. Indeed, a vehicle cannot handle r before (1) the vehicle is on w, (2) r is revealed, and (3) the beginning e_r of the time window minus the time $t_{w,r}$ needed to go from w to r. Time

$$d^{\max}_{r,w} = \min\{l_r - t_{w,r},\ \overline{on}(w) - S_{r,w}\}$$

is the latest time at which a vehicle can handle r (which also involves a service time s_r) from waiting vertex w and still leave r it at time $t \leq \overline{on}(w)$. Here $S_{r,w} = t_{w,r} + s_r + t_{r,w}$. Given the set W^x_r of feasible waiting vertices for r, we define the waiting vertex $\mathrm{w}(r)$ associated with r as follows:

- If $W^x_r = \emptyset$, then $\mathrm{w}(r) = \perp$: r is always rejected as it has no feasible waiting vertex;
- Otherwise, $\mathrm{w}(r)$ is set to the closest feasible vertex of W^x_r, in terms of travel time $t_{r,\mathrm{w}(r)}$. Further ties are broken with respect to vertex number.

Once finished, the request assignment ends up with a partition

$$\{\pi_\perp, \pi_1, ..., \pi_K\}$$

of R, where π_k is the set of requests assigned to the waiting vertices visited by vehicle k and $\pi_\perp$ is the set of unassigned requests (such that $\mathrm{w}(r) = \perp$). We note π_w, the set of requests assigned to a waiting vertex $w \in W^x$. We note $\mathrm{fst}(\pi_w)$ and $\mathrm{fst}(\pi_k)$, the first requests of π_w and π_k, respectively, according to the order $<_R$. For each request $r \in \pi_k$ such that $r \neq \mathrm{fst}(\pi_k)$, we note r^-, the request of π_k that immediately precedes r according to the order $<_R$. Remember that all these notations are specific to a first-stage solution (x, τ):

5.2.1.3 *Using $\mathcal{R}^q$ to adapt a first-stage solution at time t*

At each time step t, the recourse strategy is applied to compute the second-stage solution (x^t, A^t), given the first-stage solution (x, τ), the second-stage solution (x^{t-1}, A^{t-1}) at the end of time $t-1$, and the incoming requests ξ^t. Recall that A^{t-1} is likely to contain some requests that have been accepted but are not yet satisfied.

AVAILABILITY TIME $available(r)$. Besides vehicle operations, a key point of the recourse strategy is to decide whether each request $r \in \xi^t$ that is found to appear at time $t = \Gamma_r$ will be accepted or not. Let k be the vehicle associated with r, *i.e.*, $r \in \pi^k$. The decision to accept

or reject r depends on the time at which k will be available to serve r. By available, we mean that it has finished serving all its accepted requests that precede r (according to $<_R$) and has reached waiting vertex $\mathrm{w}(r)$. This time is denoted by $available(r)$. It is defined only when we know all accepted requests that are assigned to $\mathrm{w}(r)$ and that must be served before r. As the requests assigned to $\mathrm{w}(r)$ are ordered by increasing reveal time, we know all these accepted requests for sure when $t \geq \Gamma_{r^-}$. In this case, when vehicles exclusively serve customer requests by performing round trips from corresponding waiting locations, $available(r)$ is recursively defined by:

$$available(r) = \begin{cases} \underline{on}(\mathrm{w}(r)) & \text{if } r = \mathrm{fst}(\pi_{\mathrm{w}(r)}) \\ available(r^-) & \text{if } r \neq \mathrm{fst}(\pi_{\mathrm{w}(r)}) \\ & \text{and } r^- \notin A^t \\ \max\{e_{r^-}, available(r^-) + t_{\mathrm{w}(r),r^-}\} & \\ \quad + s_{r^-} + t_{r^-,\mathrm{w}(r)} & \text{otherwise.} \end{cases}$$

REQUEST NOTIFICATIONS. A^t is the set of requests accepted up to time t. It is initialized with A^{t-1} as all previously accepted requests must still be accepted at time t. Then incoming requests $r \in \xi^t$ are considered in increasing order with respect to $<_R$. r is either accepted (added to A^t) or rejected (not added to A^t). A request r is accepted if the vehicle assigned to it is available on time and its capacity is not exceeded, *i.e.*, if

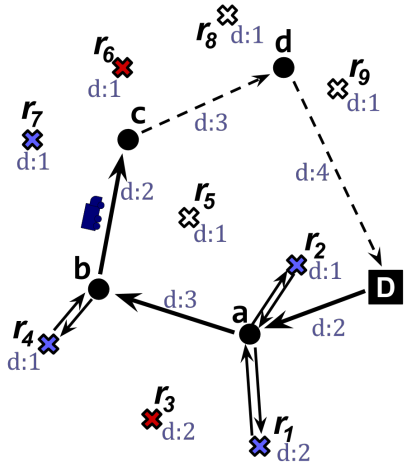
$$\mathrm{w}(r) \neq \perp \ \wedge \ available(r) \leq d^{\max}_{r,\mathrm{w}(r)} \ \wedge \ q_r + \sum_{r' \in \pi_k \cap A^t} q_{r'} \leq Q \quad (5.5)$$

VEHICLE OPERATIONS. Once A^t has been computed, vehicle operations for time unit t must be decided. Each vehicle operates independently from all other vehicles. If vehicle k is traveling between a waiting vertex and a customer vertex, or if it is serving a request, then its operation remains unchanged. Otherwise, its operations are defined in Algorithm 2. Figure 5.3 shows an example of a second-stage solution at time $t = 17$, from an operational point of view.

In the following section 5.2.2, we show how to efficiently compute the exact expected cost incurred by recourse strategy $\mathcal{R}^q$. In Section 5.2.3, we show how the resulting equations, when specialized to the special case of the SS-VRP-C, naturally reduce to the ones proposed in Bertsimas, 1992.

5.2.2 *Expected cost of second-stage solutions under $\mathcal{R}^q$*

Given a recourse strategy $\mathcal{R}$ and a first-stage solution (x, τ) to the SS-VRPTW-CR, a naive approach for computing $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ would be



r	w(r)	Γ_r	$[e_r, l_r]$	$d^{min}_{r,a}$	$d^{max}_{r,a}$	status	$av(r)$	$sat(r)$
a	*Arrival time : **3***		*Departure time : **9***					
1	a	2	2 , 6	3	4	*satisf.*	3	5
2	a	2	2 , 8	3	7	*satisf.*	7	8
3	a	5	5 , 15	5	5	*rejected*	9	*n.a.*
b	*Arrival time : **12***		*Departure time : **16***					
4	b	10	13, 16	13	13	*satisf.*	12	14
5	b	13	13 , 17	13	14	*absent !*	15	*n.a.*
c	*Arrival time : **18***		*Departure time : **20***					
6	c	16	16 , 20	18	18	*accepted*	18	19
7	c	17	17 , 19	18	18	*rejected*	20	*n.a.*
d	*Arrival time : **23***		*Departure time : **26***					
8	d	18	18 , 25	23	24	*unknown*	23	**?**
9	d	20	20 , 30	23	24	*unknown*	**?**	**?**

current time: t = 17

Figure 5.3: Example of second-stage solution at time $t = 17$, under strategy $\mathcal{R}^q$. A filled cross represents a request that appeared, an empty one a request that is either still unknown (*e.g.*, r_8) or revealed as being absent (*i.e.*, did not appear, *e.g.*, r_5). Here $\pi_k = \langle r_a, r_1, \ldots, r_9 \rangle$ is the sequence of requests assigned to the vehicle, according to (x, τ). We assume $q_r = s_r = 0, \forall r \in R$. $sat(r)$ represents, for a request r, the time at which r gets satisfied. Function $available(r)$ is here designated by $av(r)$ for short.

Algorithm 2: Operations (TRAVEL, SERVE or WAIT) of vehicle k, at current time t. Let w be the waiting vertex the vehicle is currently assigned to, $\mathrm{s}(w)$ the waiting vertex (or the depot) that follows w in x.

```
1 if t = on(w) then TRAVEL from w to s(w);
2 else
3     Let P = {r ∈ π_w | c_r ∉ x^{0..t} ∧ (r ∈ A^t ∨ t < Γ_r)}, the set of
      requests of π_w not yet satisfied, either accepted or not yet
      revealed;
4     if P = ∅ then TRAVEL to s(w);
5     else
6         r^next ← smallest element of P according to <_R;
7         if t < d^min_{r^next,w} then WAIT until t + 1 ;
8         else TRAVEL to r^next, SERVE it, and TRAVEL back to w;
```

to literally follow equation (5.4), therefore using the strategy described by $\mathcal{R}$ in order to confront (x, τ) with each and every possible scenario $\xi \subseteq R$. Because there are an exponential number of scenarios with respect to $|R|$, this naive approach is not affordable in practice. In this section, we show how the expected number of rejected requests under the recourse strategy $\mathcal{R}^q$ may be computed in $\mathcal{O}(nh^2Q)$ using closed-form expressions.

Let us recall that we assume that request probabilities are independent of each other; *i.e.*, for any couple of requests $r, r' \in R$, the probability $p_{r \wedge r'}$ that both requests will appear is given by $p_{r \wedge r'} = p_r \cdot p_{r'}$.

$\mathscr{Q}^{\mathcal{R}^q}(x, \tau)$ is equal to the expected number of rejected requests, which in turn is equal to the expected number of requests that are found to appear minus the expected number of accepted requests. Under the independence hypothesis, the expected number of revealed requests is given by the sum of all request probabilities, whereas the expected number of accepted requests is equal to the cumulative sum, for every request r, of the probability that it belongs to A^h, *i.e.*,

$$\mathscr{Q}^{\mathcal{R}^q}(x, \tau) = \sum_{r \in R} p_r - \sum_{r \in R} \mathrm{P}\{r \in A^h\} = \sum_{r \in R} \left(p_r - \mathrm{P}\{r \in A^h\}\right) \quad (5.6)$$

The probability $\mathrm{P}\{r \in A^h\}$ is computed by considering every feasible time $t \in [d^{\min}_{r,w}, d^{\max}_{r,w}]$ and every possible load configuration $q \in [0, Q - q_r]$ that satisfies r:

$$\mathrm{P}\{r \in A^h\} = \sum_{t=d^{\min}_{r,w}}^{d^{\max}_{r,w}} \sum_{q=0}^{Q-q_r} g_1(r, t, q). \quad (5.7)$$

$g_1(r, t, q)$ is the probability that r has appeared and that vehicle k leaves $\mathrm{w}(r)$ at time t with load q to serve r, *i.e.*,

$$g_1(r, t, q) \equiv \mathrm{P}\{r \in \xi^{\Gamma_r}, \mathrm{departureTime}(r) = t \text{ and } \mathrm{load}(k, t) = q\}$$

where load(k,t) is the load of vehicle $k \in [1,K]$ at time $t \in H$, and departureTime$(r) = \max\{\mathit{available}(r), d^{\min}_{r,\mathrm{w}(r)}\}$ is the time at which it actually leaves the waiting vertex $\mathrm{w}(r)$ in order to serve r (the vehicle may have to wait if $\mathit{available}(r)$ is smaller than the earliest time for leaving $\mathrm{w}(r)$ to serve r).

5.2.2.1 *Computation of probability $g_1(r,t,q)$*

Recall that π_k is the set of potential requests on route $k \in [1,K]$, ordered by $<_R$. The base case for computing g_1 is concerned with the very first potential request on the entire route, $r = \mathrm{fst}(\pi_k)$, which must be considered as soon as vehicle k arrives at $w = \mathrm{w}(r)$, that is, at time $\underline{on}(w)$, except if $\underline{on}(w) < d^{\min}_{r,w}$:

$$\text{if } r = \mathrm{fst}(\pi_k) \text{ then}$$
$$g_1(r,t,q) = \begin{cases} p_r & \text{if} \quad t = \max\{\underline{on}(w), d^{\min}_{r,w}\} \quad \wedge \quad q = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

For any $q \geq 1$, $g_1(r,t,q)$ is equal to zero as vehicle k necessarily carries an empty load when considering the first request r.

The more general case of a request r which is not the first request of a waiting vertex $w \in W^x$, (*i.e.*, $w \neq \mathrm{fst}(\pi_w)$), depends on the time and load configuration at which vehicle k is available for r. Although $\mathit{available}(r)$ and load(k,t) are both deterministic when we know the set $A^{\Gamma_{r^-}}$ of previously accepted requests, this is not true anymore when computing probability $g_1(r,t,q)$. As a consequence, $g_1(r,t,q)$ depends on the probability $f(r,t,q)$ that vehicle k is available for r at time t with load q:

$$f(r,t,q) \equiv \mathrm{P}\{\mathit{finishToServe}(r^-) = t \text{ and } \mathrm{load}(k,t) = q\}.$$

Note that for any such request $r \in R : r \neq \mathrm{fst}(\pi_{\mathrm{w}(r)})$, the time $\mathit{finishToServe}(r^-)$ is equivalent to $\mathit{available}(r)$. On the contrary, we will see that this is not the case for a request that is the first of its waiting vertex. The computation of f is detailed below. Given this probability f, the general case for computing g_1 is:

$$\text{if } r \neq \mathrm{fst}(\pi_{\mathrm{w}(r)}) \text{ then}$$
$$g_1(r,t,q) = \begin{cases} p_r \cdot f(r,t,q) & \text{if} \quad t > d^{\min}_{r,\mathrm{w}(r)} \\ p_r \cdot \sum_{t'=\underline{on}(\mathrm{w}(r))}^{d^{\min}_{r,\mathrm{w}(r)}} f(r,t',q) & \text{if} \quad t = d^{\min}_{r,\mathrm{w}(r)} \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

Indeed, if $t > d^{\min}_{r,\mathrm{w}(r)}$, then vehicle k leaves $\mathrm{w}(r)$ to serve r as soon as it becomes available. If $t < d^{\min}_{r,\mathrm{w}(r)}$, the probability that vehicle k leaves $\mathrm{w}(r)$ at time t is null since $d^{\min}_{r,\mathrm{w}(r)}$ is the earliest time for

serving r from $\mathrm{w}(r)$. Finally, at time $t = d^{\min}_{r,\mathrm{w}(r)}$, we must consider the possibility that vehicle k has been waiting to serve r since an earlier time $\underline{on}(\mathrm{w}(r)) \leq t' < d^{\min}_{r,\mathrm{w}(r)}$. In this case, the probability that vehicle k leaves $\mathrm{w}(r)$ to serve r at time t is p_r times the probability that vehicle k has actually been available from a time $\underline{on}(\mathrm{w}(r)) \leq t' \leq d^{\min}_{r,\mathrm{w}(r)}$.

We complete the computation of g_1 with the particular case of a request r which is not the first of the route (*i.e.*, $r \neq \mathrm{fst}(\pi_k)$) but is the first assigned to the waiting vertex associated with r (*i.e.*, $r = \mathrm{fst}(\pi_{\mathrm{w}(r)})$). As the arrival time on $\mathrm{w}(r)$ is fixed by the first-stage solution, departureTime(r) is necessarily $\max(\underline{on}(\mathrm{w}(r)), d^{\min}_{r,\mathrm{w}(r)})$. In particular, time *finishToServe*(r^-) is no longer equivalent to *available*(r). Unlike departureTime(r), load(k,t) is not deterministic but rather depends on what happened previously. More precisely, load(k,t) depends on the load carried by vehicle k when it has finished serving r^- at the previous waiting location $\mathrm{w}(r^-)$. For every first request of a waiting vertex, but not the first of the route ($r = \mathrm{fst}(\pi_{\mathrm{w}(r)})$ and $r \neq \mathrm{fst}(\pi_k)$), we then have:

$$g_1(r,t,q) = \begin{cases} p_r \cdot \sum_{t'=\underline{on}(\mathrm{w}(r^-))}^{\overline{on}(\mathrm{w}(r^-))} f(r,t',q) & \text{if } t = \max(\underline{on}(\mathrm{w}(r)), d^{\min}_{r,\mathrm{w}(r)}) \\ 0 & \text{otherwise ,} \end{cases} \tag{5.10}$$

where we see that all possible time units for vehicle k to serve r^- belong to $\left[\underline{on}(\mathrm{w}(r^-)), \overline{on}(\mathrm{w}(r^-))\right]$.

5.2.2.2 *Computation of probability $f(r,t,q)$*

Let us now define how to compute $f(r,t,q)$, the probability that vehicle k becomes available for r at time t with load q. This depends on what happened to the previous request r^-. We have to consider three cases: (a) r^- appeared and was satisfied, (b) r^- appeared but was rejected, and (c) r^- did not appear. Let us introduce our last probability $g_2(r,t,q)$, which is the probability that a request r did not appear ($r \notin \xi^{\Gamma_r}$) and is discarded at time t while the associated vehicle carries load q. We note discardedTime$(r) = \max\{\mathit{available}(r), \Gamma_r\}$, the time at which the vehicle becomes available for r whereas r does not appear:

$$g_2(r,t,q) \equiv \mathrm{P}\{r \notin \xi^{\Gamma_r}, \mathrm{discardedTime}(r) = t \text{ and } \mathrm{load}(k,t) = q\}$$

The computation of g_2 is detailed below. Given g_2, we compute f as follows:

$$\begin{aligned} f(r,t,q) = {} & g_1(r^-, t - S_{r^-,\mathrm{w}(r^-)}, q - q_{r^-}) \cdot \delta(r^-, t - S_{r^-}, q - q_{r^-}) \\ & + \; g_1(r^-,t,q) \cdot \big(1 - \delta(r^-,t,q)\big) \; + \; g_2(r^-,t,q) \end{aligned} \tag{5.11}$$

where the indicator function $\delta(r,t,q)$ returns 1 if and only if request r is satisfiable from vertex $\mathrm{w}(r)$ at time t with load q; *i.e.*, $\delta(r,t,q) = 1$ if

$t \leq d^{\max}_{r,\mathrm{w}(r)}$ and $q + q_r \leq Q$, whereas $\delta(r,t,q) = 0$ otherwise. The first term in the summation of the right hand side of equation (5.11) gives the probability that request r^- actually appeared and was satisfied (case *a*). In such a case, departureTime(r^-) must be the current time t minus the delay S_{r^-} needed to serve r^-. The second and third terms of equation (5.11) add the probability that the vehicle was available at time t but that request r^- did not consume any operational time. There are only two possible reasons for that: either r^- actually appeared but was not satisfiable (case *b*, corresponding to the second term) or r^- did not appear at all (case *c*, corresponding to the third term). Note that $f(r,t,q)$ must be defined only when r is not the first potential request of a waiting location.

5.2.2.3 *Computation of probability $g_2(r,t,q)$*

This probability is computed recursively, as for g_1. For the very first request of the route of vehicle k, we have:

if $r = \mathrm{fst}(\pi_k)$ then

$$g_2(r,t,q) = \begin{cases} 1 - p_r, & \text{if} \quad t = \max(\underline{on}(\mathrm{w}(r)), \Gamma_r) \quad \wedge \quad q = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.12}$$

The general case of a request which is not the first of its waiting vertex is quite similar to the one of function g_1. We just consider the probability $1 - p_r$ that r is found not to appear and replace $d^{\min}_{r,\mathrm{w}(r)}$ by the reveal time Γ_r:

if $r \neq \mathrm{fst}(\pi_{\mathrm{w}(r)})$ then

$$g_2(r,t,q) = \begin{cases} (1-p_r) \cdot f(r^-,t,q) & \text{if } t > \max(\underline{on}(\mathrm{w}(r)), \Gamma_r) \\ (1-p_r) \cdot \sum_{t'=\underline{on}(\mathrm{w}(r))}^{\max(\underline{on}(\mathrm{w}(r)),\Gamma_r)} f(r^-,t',q) & \text{if } t = \max(\underline{on}(\mathrm{w}(r)), \Gamma_r) \\ 0 & \text{otherwise.} \end{cases} \tag{5.13}$$

Finally, for the first request of a waiting location that is not the first of its route, we have:

if $r = \mathrm{fst}(\pi_{\mathrm{w}(r)})$ and $r \neq \mathrm{fst}(\pi_k)$ then

$$g_2(r,t,q) = \begin{cases} (1-p_r) \cdot \sum_{t'=\underline{on}(\mathrm{w}(r^-))}^{\overline{on}(\mathrm{w}(r^-))} f(r^-,t',q) & \text{if } t = \max(\underline{on}(\mathrm{w}(r)), \Gamma_r) \\ 0 & \text{otherwise.} \end{cases} \tag{5.14}$$

5.2.2.4 *Computational complexity.*

The complexity of computing $\mathscr{Q}^{\mathcal{R}^q}(x,\tau)$ is equivalent to that of filling up K matrices of size $|\pi_k| \times h \times Q$ containing all the $g_1(r,t,q)$ proba-

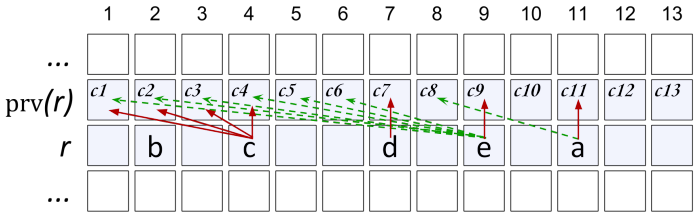


Figure 5.4: Cell dependencies in a $f(r,t,q)$-matrix. The q dimension is omitted for simplicity, hence assuming zero demands. We assume $\underline{on}(w) = 1$, $\Gamma_r = 4$, $d^{\min}_{r,w} = 6$ and $S_r = 3$. Cell **a** represents $f(r,11)$, whereas cell ***c8*** represents $f(r^-,9)$. Since $t \geq \Gamma_r$ and $t \geq d^{\min}_{r,w}$ at cell **a**, $f(r,11)$ depends directly and exclusively on cells ***c8*** for its first term of eq. (5.11), and ***c11*** for the second and third ones. Since cell **b** has $t < \Gamma_r$ and $t < d^{\min}_{r,w}$, it does not depend on another cell as its probability must be zero. Cell **c** has $t = \Gamma_r$ and $t < d^{\min}_{r,w}$, so the first term of eq. (5.11) is zero and the second and third depend on probabilities at cells ***c1*** to ***c4***. Cell **d** has $\Gamma_r \leq t < d^{\min}_{r,w}$, so it only depends on cell ***c7***. Finally, since cell **e** has $t = d^{\min}_{r,w}$, first term of eq. (5.11) depends on cells ***c1*** to ***c6***, whereas $t \geq \Gamma_r$ makes second and third terms depend on cell ***c9*** only.

bilities. In particular, once the probabilities in cells $(r^-, 1 \cdots t, 1 \cdots q)$ are known, the cell (r,t,q) such that $r \neq \text{fst}(\pi_w)$ can be computed in $\mathcal{O}(h)$ according to equation (5.9). Given n customer vertices and a time horizon of length h, there are at most $|R| = nh \geq \sum_{k=1}^{K} |\pi_k|$ potential requests in total, leading to an overall worst case complexity of $\mathcal{O}(nh^2Q)$.

*The expected cost under $\mathcal{Q}^{\mathcal{R}^q}$ is computable in **pseudo-polynomial** time.*

5.2.2.5 *Incremental computation.*

Since we are interested in computing $\mathrm{P}\{r \in A^h\}$ for each request r separately, by following the definition of g_1 and f, the probability of satisfying r only depends on the g_1 and g_2 probabilities associated with r^-. As a consequence, two similar first-stage solutions are likely to share equivalent subsets of probabilities. This is of particular interest when considering local search based methods generating (first-stage) solutions in sequence, where each new solution is usually quite similar to the previous one. In fact, for every two similar solutions, subsets of equivalent probabilities can easily be deduced, hence allowing an incremental update of the expected cost. This does not change the time complexity, as in the worst case (*i.e.*, when the first waiting vertex of each sequence in x has been changed), all probabilities must be recomputed. However, this greatly improves the efficiency in practice.

Figure 5.4 gives a visual representation of how the f-probabilities of a request r depend on those of its predecessor r^-. Using the same visual representation, Figure 5.5 shows how, within the first stage sequence π_k of potential requests from the example of Figure 5.3, the f-probabilities of each requests depend on each others.

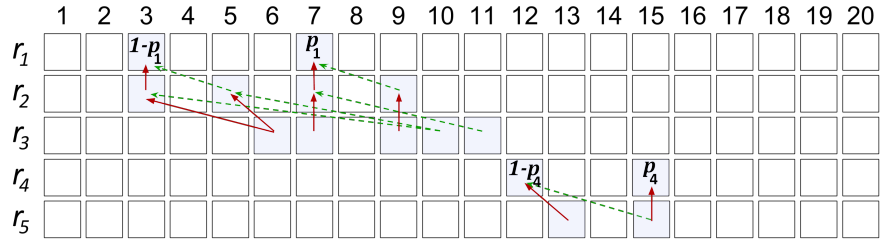


Figure 5.5: Dependencies between f-probabilities, when considering the first stage sequence π_a of potential requests from example of Figure 5.3. The q dimension is omitted for simplicity, hence assuming zero demands. Since r_1 is the very first request of π_k^a, there is a $1 - p_1$ probability that the vehicle gets rid of r_1 at time $\max(\underline{on}(a), \Gamma_{r_1}) = 3$, hence $f(r_1, 3) = 1 - p_1$. There is p_1 probability that r_1 appears, in which case the vehicle finishes with it at time $\max(\underline{on}(w), d^{\min}_{r_1,w}) + S_1$ that is, $f(r_1, 7) = p_1$. Then all the f-probabilities for requests in π_k^a are computed incrementally, starting from $f(r_1, 3)$ and $f(r_1, 7)$. For instance, $f(r_2, 7) = p_1 \cdot (1 - p_2)$ and $f(r_2, 9) = p_1 \cdot p_2$. By using an adequate data structure while filling up such a sparse matrix, substantial savings can be made on the computational effort.

5.2.3 *Relation with SS-VRP-C*

As presented in Section 1.2, the SS-VRP-C (Bertsimas, 1992) differs by having stochastic binary demands (which represent the random customer presence) and no time window. In this case, the goal is to minimize the expected distance traveled, provided that when a vehicle reaches its maximal capacity, it unloads by making a round trip to the depot. In order to compute the expected length of a first-stage solution that visits all customers, a key point is to compute the probability distribution of the vehicle's current load when reaching a customer. In fact, this is directly related to the probability that the vehicle makes a round trip to the depot to unload, which is denoted by the function "$f(m, r)$" in Bertsimas, 1992.

The existing closed-form expression for Bertsimas' SS-VRP-C can be derived from our equations!

Here we highlight the relation between SS-VRPTW-CR and SS-VRP-C by showing how Bertsimas's "$f(m, r)$" equation can be derived from equation (5.11) when time windows are not taken into account.

Since there is no time window consideration, we can state that $\Gamma_r = d^{\min}_{r,w} = 1$ and $d^{\max}_{r,w} = +\infty$ for any request r. Also, each demand q_r is equal to 1. Consequently, the δ-function used in the computation of the f probabilities depends only on q and is equal to 1 if $q \leq Q$. Therefore, the f probabilities are defined by:

$$f(r, t, q) = g_1(r^-, t - S_{r^-}, q - 1) \; + \; g_2(r^-, t, q).$$

Now let $f'(r, q) = \sum_{t \in H} f(r, t, q)$. As $f(r, t, q)$ is the probability that the vehicle is available for r at time t with load q, $f'(r, q)$ is the probability that the vehicle is available for r with load q during the day. It is also true that $f'(r, q)$ gives the probability that exactly q requests among

the $r_1, ..., r^-$ potential ones actually appear (with a unit demand). We have:

$$f'(r,q) = \sum_{t \in H} f(r,t,q) = \sum_{t \in H} g_1(r^-, t - S_{r^-}, q-1) \; + \; \sum_{t \in H} g_2(r^-, t, q)$$

As we are interested in $f'(r,q)$, not the travel distance, we can assume that all potential requests are assigned to the same waiting vertex. Then either $r = \text{fst}(\pi_k)$ or $r \neq \text{fst}(\pi_{\text{w}(r)})$. If $r = \text{fst}(\pi_k)$ we naturally obtain:

$$\begin{aligned} f'(r,q) &= \sum_{t \in H} p_r \cdot f(r,t,q-1) + \sum_{t \in H} (1-p_r) \cdot f(r,t,q) \\ &= \begin{cases} p_r + (1-p_r) = 1, & \text{if} \quad q = 0 \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

If $r \neq \text{fst}(\pi_{\text{w}(r)})$, since we always have $t \geq d^{\min}_{r,w}$, we have:

$$\begin{aligned} f'(r,q) &= \sum_{t \in H} p_r \cdot f(r,t,q-1) + \sum_{t \in H} (1-p_r) \cdot f(r,t,q) \\ &= p_r \cdot f'(r,q-1) \; + \; (1-p_r) \cdot f'(r,q). \end{aligned}$$

We directly see that the definition of $f'(r,q)$ is exactly the same as the corresponding function "$f(m,r)$" described in Bertsimas, 1992 for the SS-VRP-CD with unit demands, that is, the SS-VRP-C.

5.3 IMPROVED RECOURSE STRATEGY

The improved strategy $\mathcal{R}^{q+}$ provides a better upper bound on the expected cost under perfect reoptimization.

We now introduce $\mathcal{R}^{q+}$, a somewhat cleverer strategy than $\mathcal{R}^q$. It is based on the same request assignment and ordering as strategy $\mathcal{R}^q$, yet it improves the latter by saving operational time. More specifically, $\mathcal{R}^{q+}$ avoids some pointless round trips from waiting vertices, traveling directly towards a revealed request from a previously satisfied one. Furthermore, a vehicle is now allowed to travel directly from a customer vertex c to the next planned waiting vertex w_n without passing by the waiting vertex $\text{w}(c)$ associated with c. Consequently, this allows a vehicle to finish servicing c later if reaching w_n from c is faster than reaching w_n from c via $\text{w}(c)$. Figure 5.6 provides an intuitive illustration of the differences between strategies $\mathcal{R}^q$ and $\mathcal{R}^{q+}$ from an operational point of view.

Under $\mathcal{R}^{q+}$, the location from which the vehicle travels to satisfy a request r may be any customer vertex $v \in C$, in addition to $\text{w}(r)$. Given $\text{w}(r)$, then let $d^{\min+}_{r,v} = \max(\underline{on}(\text{w}(r)),\ \Gamma_r,\ e_r - t_{v,r})$ be the minimum time at which a vehicle can leave its current location $v \in C \cup \{\text{w}(r)\}$ in order to satisfy a request r. Again, recall that the request ordering and assignment is the same as under $\mathcal{R}^q$, that is, based on $d^{\min}_{r,\text{w}(r)}$ and $d^{\max}_{r,\text{w}(r)}$ as described in Section 5.2.1. Therefore, $d^{\min+}_{r,v}$ will only be useful to request notification and vehicle operation phases.

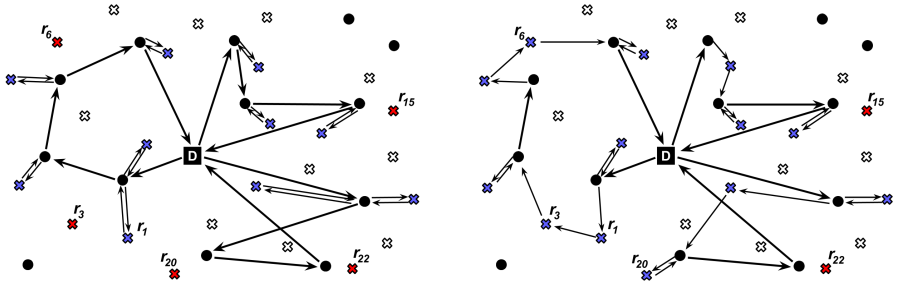


Figure 5.6: Comparative examples of strategies $\mathcal{R}^q$ (left) and $\mathcal{R}^{q+}$ (right). A filled cross represents a revealed request. Under $\mathcal{R}^q$, some requests ($r_3, r_6, r_{15}, r_{20}, r_{22}$) can be missed. By avoiding pointless journeys when possible, $\mathcal{R}^{q+}$ is likely to end up with a lower number of missed requests than strategy $\mathcal{R}^q$. For example, if request r_3 is revealed by the time request r_1 is satisfied, then traveling directly to r_3 could help satisfy it. Similarly, on a different route, by traveling directly to the waiting vertex associated with request r_{20}, the vehicle could save enough time to satisfy r_{20}.

Let $s(\mathrm{w}(r))$ be the waiting vertex that follows $\mathrm{w}(r)$ in the first-stage solution (x, τ). Since the vehicle is now allowed to travel to $s(\mathrm{w}(r))$ from the customer vertex $c_r \in C$ of a request $r \in \xi$, we also need a variant of $d^{\max}_{r,\mathrm{w}(r)}$ in order to take the resulting savings into account. We call it $d^{\max+}_{r,v} = \min\left(l_r - t_{v,r},\ \underline{on}(\mathrm{s}(\mathrm{w}(r))) - t_{v,r} - s_r - t_{r,\mathrm{s}(\mathrm{w}(r))}\right)$.

5.3.0.1 *Vehicle operations*

If vehicle k is traveling between a waiting vertex and a customer vertex, or if it is serving a request, then its operation remains unchanged. Otherwise, its operations are defined in Algorithm 3.

5.3.0.2 *Request notification*

At a current time $t \geq \Gamma_r$, the time $available^+(r)$ at which the vehicle will be able to take care of r is still deterministic and computable. However, under $\mathcal{R}^{q+}$, a request r can be considered as the vehicle is idle either at a customer vertex $c_{r'} \in C$ such that $r' \in \pi_w, r' <_R r$ or at waiting vertex $w = \mathrm{w}(r)$. Let $v^+(r)$ be the vertex from which request r will be served ($v^+(r)$ is known at time $t \geq \Gamma_r$). The request r is then accepted if and only if:

$$available^+(r) \leq d^{\max+}_{r,v^+(r)} \wedge \quad q_r + \sum_{r' \in \pi_k \cap A^t} q_{r'} \leq Q. \tag{5.15}$$

Algorithm 3: Operations (TRAVEL, SERVE or WAIT) of vehicle k, at current time t. Vertex v is the position of vehicle k, w the waiting vertex it is currently assigned to, $\mathrm{s}(w)$ the waiting vertex (or the depot) that follows w in x.

```
if t = on(s(w)) − t_{v,s(w)} then TRAVEL from v to s(w);
else
    P ← set of requests of π_w not yet satisfied, either accepted or
    not yet revealed;
    if P = ∅ then TRAVEL to s(w);
    else
        r^next ← smallest element of P according to <_R;
        if w(r^next) = w and r^next is revealed and accepted then
            WAIT until d^min_{r^next,v}, TRAVEL to r^next and SERVE the request;
        if w(r^next) = w but r^next is not known yet (t < Γ_{r^next}) then
            TRAVEL back to waiting location w
        if w(r^next) ≠ w then
            WAIT until on(s(w)) − t_{r,s(w)} and TRAVEL to s(w)
```

where $available^+(r)$ and $v^+(r)$ are defined as follows. Given current time $t \geq \Gamma_r$ and previous request $r' \in r^-$, function $available(r)$ defined in section 5.2.1 must be adapted to strategy $\mathcal{R}^{q+}$:

$$available^+(r) = \begin{cases} \max\left(available^+(r') + t_{v^+(r'),r'}, e_{r'}\right) + s_{r'} + t_{r',v^+(r)} & \\ & \text{if } r' \in A^t, \\ available^+(r') + t_{v^+(r'),v^+(r)} & \text{otherwise,} \end{cases}$$

with base case of a request $r_1 = \mathrm{fst}(\pi_w)$ being the first of its waiting vertex:

$$available^+(r_1) = \underline{on}(w), w = \mathrm{w}(r_1) = \mathrm{w}(r).$$

The location $v^+(r)$ from which the vehicle travels towards request r depends on whether r reveals by the time the vehicle finishes to satisfy the last accepted request:

$$v^+(r) = \begin{cases} c_{r'} & \text{if } \Gamma_r \leq \max\left(available^+(r') + t_{v^+(r'),r'}, e_{r'}\right) + s_{r'} \\ & \quad \text{and } r' \in A^t \\ v^+(r') & \text{if } r' \notin A^t \\ \mathrm{w}(r) & \text{otherwise} \end{cases}$$

with base case $v^+(r_1) = \mathrm{w}(r_1) = \mathrm{w}(r)$, $r_1 = \mathrm{fst}(\pi_w)$.

5.3.1 *Expected cost of second-stage solutions under $\mathcal{R}^{q+}$*

Unlike strategy $\mathcal{R}^q$, the satisfiability of a request r depends not only on the current time and vehicle load but also on the vertex from which the

vehicle would leave to serve it. The candidate vertices are necessarily either the current waiting location $w = \mathrm{w}(r)$ or any vertex hosting one of the previous requests associated with w. Consequently, under $\mathcal{R}^{q+}$, the probability $\mathrm{P}\{r \in A^h\}$ is decomposed over all the possible time, load, and vertex configurations:

$$\mathrm{P}\{r \in A^h\} = \sum_{t=d_{r,w}^{\min+}}^{d_{r,w}^{\max+}} \sum_{q=0}^{Q-q_r} g_1^{\mathrm{w}(r)}(r,t,q) + \sum_{\substack{r' \in \pi_w \\ r' <_R r}} \sum_{t=d_{r,r'}^{\min+}}^{d_{r,r'}^{\max+}} \sum_{q=0}^{Q-q_r} g_1^{r'}(r,t,q) \tag{5.16}$$

where

$$\begin{aligned} g_1^{w}(r,t,q) \equiv \mathrm{P}\{&\text{request } r \text{ appeared at a } t' \leq t, \text{ vehicle has load } q \\ &\text{and serves } r \text{ from } w \text{ at } t \text{ if } r \text{ is accepted}\} \\ g_1^{r'}(r,t,q) \equiv \mathrm{P}\{&\text{request } r \text{ appeared at a } t' \leq t, \text{ vehicle has load } q \\ &\text{and serves } r \text{ from } c_{r'} \text{ at } t \text{ if } r \text{ is accepted}\}. \end{aligned}$$

Each tuple (v,t,q) in the summation (5.16), where v is either $\mathrm{w}(r)$ or a previous customer vertex r', represents a possible configuration for satisfying r, and we are interested in the probability $g_1^v(r,t,q)$ that the vehicle is actually available for r while being in one of those states.

The calculus of g_1^v under $\mathcal{R}^{q+}$ goes as follows. Let the following additional random functions:

$$\begin{aligned} h^{w}(r,t,q) \equiv \mathrm{P}\{&\text{the vehicle gets rid of request } r \text{ at time } t \text{ with} \\ &\text{a load of } q \text{ and at waiting location } w\} \\ h^{r'}(r,t,q) \equiv \mathrm{P}\{&\text{the vehicle gets rid of request } r \text{ at time } t \text{ with} \\ &\text{a load of } q \text{ and at location } c_{r'}\} \end{aligned}$$

and

$$\begin{aligned} g_2^{w}(r,t,q) \equiv \mathrm{P}\{&\text{request } r \text{ did not appear, the vehicle discards it} \\ &\text{at time } t \text{ with a load of } q \text{ while being at waiting loc. } w\} \\ g_2^{r'}(r,t,q) \equiv \mathrm{P}\{&\text{request } r \text{ did not appear, the vehicle discards it} \\ &\text{at time } t \text{ with a load of } q \text{ while being at location } c_{r'}\}. \end{aligned}$$

For the very first request $r_1^k = \mathrm{fst}(\pi_k)$ of the route, trivially the current load q of the vehicle must be zero, and it seems normal for the waiting location $w = \mathrm{w}(r_1^k)$ to be the only possible location from which the vehicle can be available to handling r_1^k if the request appears, or to discard it if it doesn't:

$$g_1^{w}(r_1^k,t,q) = \begin{cases} p_{r_1^k}, & \text{if} \quad t = d_{r_1^k,w}^{\min+} \quad \wedge \quad q = 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$g_2^w(r_1^k,t,q) = \begin{cases} 1-p_{r_1^k}, & \text{if} \quad t = \max(\underline{on}(w), \Gamma_{r_1^k}) \quad \wedge \quad q = 0 \\ 0 & \text{otherwise.} \end{cases}$$

The vehicle thus cannot be available for r_1^k at any other location $r' <_R r$:

$$g_1^{r'}(r_1^k,t,q) = 0 \qquad\qquad g_2^{r'}(r_1^k,t,q) = 0$$

Concerning $r_1 = \text{fst}(\pi_w)$ the first request of any other waiting location $w \neq \text{w}(r_1^k)$, we use the same trick as for strategy $\mathcal{R}^q$ in order to obtain the probabilities for each possible vehicle load q:

$$g_1^w(r_1,t,q) = \begin{cases} p_{r_1} \sum\limits_{t'=\underline{on}(w')}^{\overline{on}(w')} \Big[h^{w'}(\text{prv}(r_1),t',q) + \sum\limits_{r'\in\pi_{w'}} h^{r'}(\text{prv}(r_1),t',q)\Big], \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if} \quad t = \max(\underline{on}(w), d_{r_1,w}^{\min+}) \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$$

$$g_2^w(r_1,t,q) = \begin{cases} (1-p_{r_1}) \sum\limits_{t'=\underline{on}(w')}^{\overline{on}(w')} \Big[h^{w'}(\text{prv}(r_1),t',q) + \sum\limits_{r'\in\pi_{w'}} h^{r'}(\text{prv}(r_1),t',q)\Big], \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if} \quad t = \max(\underline{on}(w), \Gamma_{r_1}) \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$$

with $w' = \text{w}(\text{prv}(r_1))$. From any other request $r' <_R r$ we still have:

$$g_1^{r'}(r_1,t,q) = 0 \qquad\qquad g_2^{r'}(r_1,t,q) = 0$$

For a request $r >_R \text{fst}(\pi_w), w \in W^x$:

$$g_1^v(r,t,q) = \begin{cases} p_r \cdot h^v(r^-,t,q) & \text{if} \quad t > d_{r_1,v}^{\min+} \\ p_r \cdot \sum_{t'=\underline{on}(w)}^{d_{r_1,v}^{\min+}} h^v(r^-,t',q) & \text{if} \quad t = d_{r_1,v}^{\min+} \\ 0 & \text{otherwise.} \end{cases}$$

$$g_2^v(r,t,q) = \begin{cases} (1-p_r) \cdot h^v(r^-,t,q) & \text{if} \quad t > \max(\underline{on}(w), \Gamma_r) \\ (1-p_r) \cdot \sum_{t'=\underline{on}(w)}^{\max(\underline{on}(w),\Gamma_r)} h^v(r^-,t',q) & \text{if} \quad t = \max(\underline{on}(w), \Gamma_r) \\ 0 & \text{otherwise.} \end{cases}$$

when replacing v by either $w = \text{w}(r)$ or $r' \in \pi_w, r' <_R r, w = \text{w}(r)$.

At a waiting location $w \in W^x$:

$$h^w(r,t,q) \;=\; h_1^w(r,t,q) \;+\; h_2^w(r,t,q) \;+\; h_3^w(r,t,q).$$

The aforementioned terms of the sum are:

$$h_1^w(r,t,q) = \begin{cases} g_1^w(r,t^w,q-q_r)\cdot\delta^w(r,t^w,q-q_r) \\ + \quad \sum\limits_{\substack{r'\in\pi_w \\ r'<_R r}} g_1^{r'}(r,t^{r'},q-q_r)\cdot\delta^{r'}(r,t^{r'},q-q_r), & \text{if} \quad t - t_{r,w} < \Gamma_r^{\text{next}} \\ 0 & \text{otherwise.} \end{cases}$$

where

$$\delta^v(r,t,q) = \begin{cases} 1, & \text{if} \quad t \le d^{\max+}_{r,v} \ \wedge \ q + q_r \le Q \\ 0, & \text{otherwise.} \end{cases}$$

and $t^w = t - t_{w,r} - s_r - t_{r,w}$, $t^{r'} = t - t_{r',r} - s_r - t_{r,w}$ and $\Gamma^{\text{next}}_r = \Gamma_r$ if $\text{nxt}(r)$ exists, zero otherwise. The second term h^w_2 is:

$$h^w_2(r,t,q) = g^w_1(r,t,q) \cdot \big(1 - \delta(r,t,w)\big) + g^w_2(r,t,q).$$

Finally:

$$h^w_3(r,t,q) = \sum_{\substack{r' \in \pi_w \\ r' <_R r}} \Big[g^{r'}_1(r, t - t_{r',w}, q)(1 - \delta^{r'}(r, t - t_{r',w}, q)) \\ + \ g^{r'}_2(r, t - t_{r',w}, q) \Big] \cdot \text{bool}(t - t_{r',w} < \Gamma^{\text{next}}_r)$$

where $\text{bool}(a)$ returns 1 if the Boolean expression a is true, 0 otherwise. The probability that the vehicle gets rid of request r at r's location is:

$$\begin{aligned} h^r(r,t,q) \ &= g^w_1(r, t - t_{w,r} - s_r, q - q_r) \cdot \delta^w(r, t - t_{w,r} - s_r, q - q_r) \\ &\quad + \sum_{\substack{r' \in \pi_w \\ r' <_R r}} g^{r'}_1(r, t - t_{r',r} - s_r, q - q_r) \cdot \delta^{r'}(r, t - t_{r',r} - s_r, q - q_r) \end{aligned}$$

if $t \ge \Gamma^{\text{next}}_r$, otherwise $h^r(r,t,q) = 0$. Finally, the probability that request gets discarded from another request r' location:

$$h^{r'}(r,t,q) = \begin{cases} g^{r'}_1(r,t,q) \cdot (1 - \delta^{r'}(r,t,q)) \ + \ g^{r'}_2(r,t,q), & \text{if} \quad t \ge \Gamma^{\text{next}}_r \\ 0, & \text{otherwise.} \end{cases}$$

COMPUTATIONAL COMPLEXITY. Given n customer vertices, w waiting locations, a horizon of length h, and vehicle capacity of size Q, the computational complexity of computing the whole expected cost $\mathscr{Q}^{\mathcal{R}^{q+}}(x,\tau)$ is in $\mathcal{O}\big(n^2h^3Q\big)$.

SPACE COMPLEXITY. A naive implementation of equation (5.16) would basically fill up an $n^2 \times h^3 \times Q$ array. We draw attention to the fact that even a small instance with $n = Q = 10$ and $h = 100$ would then lead to a memory consumption of 10^9 floating point numbers. Using a common eight-byte representation requires more than seven gigabytes. Like strategy $\mathcal{R}^q$, important savings are obtained by noticing that the computation of g_1 functions for a given request r under $\mathcal{R}^{q+}$ only relies on the previous request r^-. By computing g_1 while only keeping in memory the expectations of r^- (instead of all nh potential requests), the memory requirement is reduced by a factor nh. This however comes at the price of making any incremental computation, based on probabilities belonging to a similar first-stage solution, impossible.

5.4 CONCLUSIONS AND RESEARCH DIRECTIONS

In this chapter, we consider the SS-VRPTW-CR problem introduced in Saint-Guillain et al., 2017. In particular, in this thesis we extend the model with two additional recourse strategies: $\mathcal{R}^q$ and $\mathcal{R}^{q+}$. These take customer demands into account and allow the vehicles to save operational time, traveling directly between customer vertices when possible. We show how, under these recourse strategies, the expected cost of a second-stage solution is computable in pseudo-polynomial time.

Future work and research avenues

The research directions described here apply locally, that is, on the new SS-VRP we just introduced. More general conclusions and future work in the context of online stochastic VRPs are discussed in the *Conclusion and Perspectives* part, concluding the thesis.

TOWARDS BETTER RECOURSE STRATEGIES. The expected cost of a first-stage solution obviously depends on how the recourse strategy fits the operational problem. Improving these strategies may tremendously improve the quality of the upper bound they provide to exact reoptimization. The recourse strategies presented in this paper are of limited operational complexity, yet their computational complexity is already very expensive. One potential improvement which would limit the increase in computational requirements would be to rethink the way in which the potential requests are assigned to waiting locations, *e.g.* by taking their probabilities and demands into account. Another direction would be to think about better, more intelligent, vehicle operations. However, an important question remains: how intelligent could a recourse strategy be such that its expected cost stays efficiently computable?

5.5 REMAINING OF THE CURRENT THESIS PART

The current part of the thesis, and in particular the remaining of it, is devoted to solution methods and applications of the new SS-VRP we just introduced, the SS-VRPTW-CR. As usual in combinatorial optimization, two very different approaches can be considered in order to solve a combinatorial optimization problem: heuristic and exact methods.

We first introduce our benchmark in Chapter 6. In Chapter 7, we briefly explore an exact approach in order to solve the SS-VRPTW-CR to optimality. However, despite the fact that major improvements can obviously be added to our exact method, the inherent complexity of the problem forces us to renonce to optimality proof. Chapter 8

hence introduce a heuristic algorithm, as well as a meta-heuristic, we specifically designed for the SS-VRPTW-CR, in order to achieve good performances on realistically sized instances. Finally, Chapter 9 applies our theoretical and empirical results to a real case study.

6 AN ALMOST REALISTIC BENCHMARK: LYON

The benchmark is derived from the benchmark described in Melgarejo et al., 2015 for the Time-Dependent TSP with Time Windows (TD-TSPTW). The latter has been created using real accurate delivery and travel time data obtained from the city of Lyon, France. It is available at http://becool.info.ucl.ac.be/resources/ss-vrptw-cr-optimod-lyon, as well as the solution files and detailed result tables of the experiments conducted in the following sections.

Our Lyon benchmark is characterized by realistic travel times.

The benchmark contains two different kinds of instances: instances with separated waiting locations and instances without separated waiting locations. In an instance without separated waiting locations, every customer vertex is also a waiting vertex, $C = W$. In all instances, the duration of an operational day is eight hours and the time horizon is $h = 480$, which corresponds to one-minute time steps.

To each potential request $r = (c_r, \Gamma_r)$ is assigned a time window $[\Gamma_r, \Gamma_r + \Delta - 1]$, where Δ is taken uniformly from $\{5, 10, 15, 20\}$. Note that the time window always starts with the reveal time Γ_r. This aims at simulating operational contexts similar to the practical application example described in introduction, the on-demand health care service at home, requiring immediate responses within small time windows. Finally, the instance format follows the framework suggested in Perboli et al., 2018.

Customer online requests attributes, as well as the stochastic knowledge, are artificially generated.

GRAPH AND TRAVEL TIMES

We derive our test instances from the benchmark described in Melgarejo et al., 2015 for the Time-Dependent TSP with Time Windows (TD-TSPTW). This benchmark has been created using real accurate delivery and travel time data coming from the city of Lyon, France. Travel times have been computed from vehicle speeds that have been measured by 630 sensors over the city (each sensor measures the speed on a road segment every 6 minutes). For road segments without sensors, travel speed has been estimated with respect to speed on the closest road segments of similar type. Figure 6.1 displays the set of 255 delivery addresses extracted from real delivery data, covering two full months of time-stamped and geo-localized deliveries from three freight carriers operating in Lyon. For each couple of delivery addresses, travel duration has been computed by searching for a quickest path between the two addresses. In the original benchmark, travel durations are computed for different starting times (by steps of 6 minutes), to take into account the fact that travel durations depend on

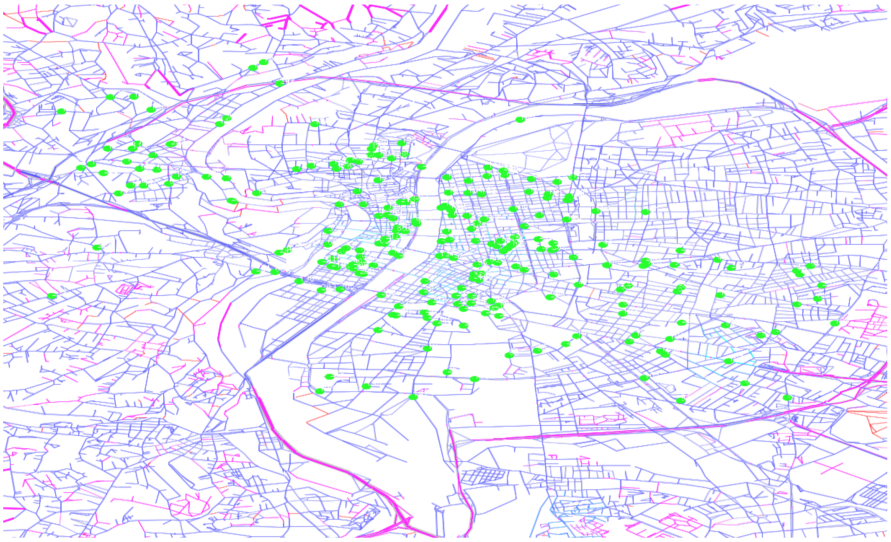


Figure 6.1: Lyon's road network. In green, the 255 customer vertices.

starting times. In our case, we remove the time-dependent dimension by simply computing average travel times (for all possible starting times). We note $\overline{V}$ the set of 255 delivery addresses, and $d_{i,j}$ the duration for traveling from i to j with $i, j \in \overline{V}$. This allows us to have realistic travel times between real delivery addresses. Note that in this real-world context, the resulting travel time matrix is not symmetric.

INSTANCE GENERATION

We have generated two different kinds of instances: instances with separated waiting locations, and instances without separated waiting locations. Each instance with separated waiting locations is denoted n**c**-m**w**-i, where $n \in \{10, 20, 50\}$ is the number of customer vertices, $m \in \{5, 10, 30, 50\}$ is the number of waiting vertices, and $x \in [1, 15]$ is the random seed. It is constructed as follows:

1. We first partition the 255 delivery addresses of $\overline{V}$ in m clusters, using the k-means algorithm with $k = m$. During this clustering phase, we have considered symmetric distances, by defining the distance between two points i and j as the minimum duration among $d_{i,j}$ and $d_{j,i}$.

2. For each cluster, we select the median delivery address, *i.e.*, the address in the cluster such that its average distance to all other addresses in the cluster is minimal. The set W of waiting vertices is defined by the set of m median addresses.

3. We randomly and uniformly select the depot and the set C of n customer vertices in the remaining set $\overline{V} \setminus W$.

Each instance without separated waiting locations is denoted n**c**+**w**-i. It is constructed by randomly and uniformly selecting the depot and the set C in the entire set $\overline{V}$ and by simply setting $W = C$. In other words, in these instances vehicles do not wait at separated waiting vertices, but at customer vertices, and every customer vertex is also a waiting location.

Furthermore, instances sharing the same number of customers n and the same random seed x (*e.g.* 50**c**-30**w**-1, 50**c**-50**w**-1 and 50**c+w**-1) always share the exact same set of customer vertices C.

OPERATIONAL DAY, HORIZON AND TIME SLOTS

We fix the duration of an operational day to 8 hours in all instances. We fix the horizon resolution to $h = 480$, which corresponds to one minute time steps. As it is not realistic to detail request probabilities for each time unit of the horizon (*i.e.*, every minute), we introduce *time slots* of 5 minutes each. We thus have $n_{\text{TS}} = 96$ time slots over the horizon. To each *time slot* corresponds a potential request at each customer vertex.

CUSTOMER POTENTIAL REQUESTS AND ATTRIBUTES.

For each customer vertex c, we generate the request probabilities associated with c as follows. First, we randomly and uniformly select two integer values μ_1 and μ_2 in $[1, n_{\text{TS}}]$. Then, we randomly generate 200 integer values: 100 with respect to a normal distribution the mean of which is μ_1 and 100 with respect to a normal distribution the mean of which is μ_2. Let us note $nb[i]$ the number of times value $i \in [1, n_{\text{TS}}]$ has been generated among the 200 trials. Finally, for each reveal time $\Gamma \in H$, if $\Gamma \mod 5 \neq 0$, then we set $p_{(c,\Gamma)} = 0$ (as we assume that requests are revealed every 5 minute time slots). Otherwise, we set $p_{(c,\Gamma)} = \min(1, \frac{nb[\Gamma/5]}{100})$. Hence, the expected number of requests at each customer vertex is smaller than or equal to 2 (in particular, it is smaller than 2 when some of the 200 randomly generated numbers do not belong to the interval $[1, n_{\text{TS}}]$, which may occur when μ_1 or μ_2 are close to the boundary values). Figure 6.2 shows a representation of the distributions in an instance involving 10 customer vertices.

For a same customer vertex, there may be several requests on the same day at different time slots, and their probabilities are assumed independent. To each potential request $r = (c_r, \Gamma_r)$ is assigned a deterministic demand q_r taken uniformly in $[0, 2]$, a deterministic service duration $s_r = 5$ and a time window $[\Gamma_r, \Gamma_r + \Delta - 1]$, where Δ is taken uniformly in $\{5, 10, 15, 20\}$ that is, either 5, 10, 15 or 20 minutes to meet the request. Note that the beginning of the time window of a request r is equal to its reveal time Γ_r. This aims at simulating operational contexts requiring immediate responses within

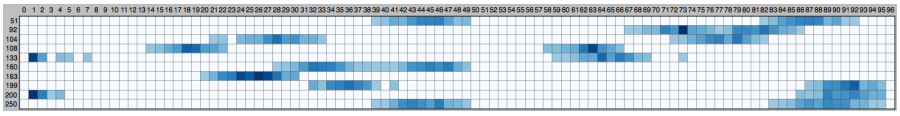


Figure 6.2: Probability distributions in instance *10-c5w-1*. Each cell represents one of the 96 time slots, for each customer vertex. The darker a cell, the more likely a request to appear at the corresponding time slot. A white cell represents a zero probability request that is, no potential request.

small time windows, such as the on-demand health care service at home discussed in Introduction of the thesis.

7

EXACT APPROACH: BRANCH-AND-CUT

In this chapter, we present a complete mathematical formulation of the SS-VRPTW-CR, along with its application to a classical branch-and-cut solution algorithm, presented in Section 7.2. The experiments, conducted in Section 7.3, show that the exact approach is clearly not efficient enough in its current, early stage, version. The last contribution of the chapter comes in Section 7.4, where we investigate the limitations of the current approach and propose some improvement ideas, inspired from the literature.

7.1 STOCHASTIC INTEGER PROGRAMMING FORMULATION

The problem stated by (5.2)-(5.3) refers to a nonlinear stochastic integer program with recourse, which can be modeled as the following simple extended three-index vehicle flow formulation:

Our 3-index formulation provides a complete mathematical representation of the SS-VRPTW-CR.

$$\underset{x,\tau}{\text{Minimize}} \quad \mathcal{Q}^{\mathcal{R}}(x,\tau) \tag{7.1}$$

subject to

$$\sum_{j\in W_0} x_{ijk} = \sum_{j\in W_0} x_{jik} = y_{ik} \qquad \forall\, i \in W_0,\ k \in [1,K] \tag{7.2}$$

$$\sum_{k\in[1,K]} y_{0k} \leq K \tag{7.3}$$

$$\sum_{k\in[1,K]} y_{ik} \leq 1 \qquad \forall\, i \in W \tag{7.4}$$

$$\sum_{\substack{i\in S \\ j\in W\setminus S}} x_{ijk} \geq y_{vk} \qquad \forall\, S \subseteq W,\ v \in S,\ k \in [1,K] \tag{7.5}$$

$$\sum_{l\in H} \tau_{ilk} = y_{ik} \qquad \forall\, i \in W,\ k \in [1,K] \tag{7.6}$$

$$\sum_{\substack{i\in W_0 \\ j\in W_0}} x_{ijk}\, d_{i,j} + \sum_{\substack{i\in W \\ l\in H}} \tau_{ilk}\, l + 1 \leq h \qquad \forall\, k \in [1,K] \tag{7.7}$$

$$y_{ik} \in \{0,1\} \qquad \forall\, i \in W_0,\ k \in [1,K] \tag{7.8}$$

$$x_{ijk} \in \{0,1\} \qquad \forall\, i,j \in W_0 :\ i \neq j,\ k \in [1,K] \tag{7.9}$$

$$\tau_{ilk} \in \{0,1\} \qquad \forall\, i \in W,\ l \in H,\ k \in [1,K] \tag{7.10}$$

Our formulation uses the following binary decision variables:

- $y_{ik} = 1$ iff vertex $i \in W_0$ is visited by vehicle $k \in [1,K]$;

- $x_{ijk} = 1$ iff the arc $(i, j) \in W_0^2$ is part of route $k \in [1, K]$;
- $\tau_{ilk} = 1$ iff vehicle k waits for $1 \leq l \leq h$ time units at vertex i.

Whereas variables y_{ik} are only of modeling purposes, yet x_{ijk} and τ_{ilk} variables solely define a SS-VRPTW-CR first stage solution. The sequence $\langle w_1, ..., w_{m'} \rangle_k$ of waiting vertices along any route $k \in [1, K]$ is obtained from x. By also considering τ, we obtain the *a priori* arrival time $\underline{on}(w)$ and departure time $\overline{on}(w)$ at any waiting vertex w in the sequence. By following the process described in the beginning of Section 5.2, each sequence π_k is computable directly from (x, τ).

Constraints (7.2) to (7.5) together with (7.9) define the feasible space of the asymmetric Team Orienteering Problem (Chao et al., 1996). In particular, constraint (7.3) limits the number of available vehicles. Constraints (7.4) ensure that each waiting vertex is visited at most once. Subtour elimination constraints (7.5) forbid routes that do not include the depot. As explained in Section 7.2, constraints (7.5) will be generated *on-the-fly* during the search. Constraint (7.6) ensures that exactly one waiting time $1 \leq l \leq h$ is selected for each visited vertex. Finally, constraint (7.7) states that the total duration of each route, starting at time unit 1, cannot exceed h.

Like classical VRP formulations, our integer program necessarily involves an exponential number of subtour elimination constraints.

SYMMETRIES The solution space as defined by constraints (7.2) to (7.11) is unfortunately highly symmetric. Not surprisingly, we see that any feasible solution actually reveals to be precisely identical under any permutation of its route indexes $p \in K$. In fact, the number of symmetric solutions even grows exponentially with the number of vehicles. Provided K vehicles, a feasible solution admits $K! - 1$ symmetries. In order to remove those symmetries from our original problem formulation, we add the following ordering constraints:

$$\sum_{i \in W} 2^i \, y_{i,p} \leq \sum_{i \in W} 2^i \, y_{i,p+1} \,, \qquad 1 \leq p \leq K - 1 \tag{7.11}$$

7.2 BRANCH-AND-CUT APPROACH

We solve program (7.1)-(7.11) by using the specialized branch-and-cut algorithm introduced by Laporte and Louveaux, 1993 for tackling stochastic integer programs with complete recourse. This algorithm is referred to as the integer L-shaped method, because of its similarity to the L-shaped method for continuous problems introduced by Slyke and Wets, 1969.

Roughly speaking, our implementation of the algorithm is quite similar to its previous applications to stochastic VRP's (see e.g. Gendreau et al., 1995, Laporte et al., 2002, Heilporn et al., 2011). As in the standard branch-and-cut scheme, an initial *current problem* (*CP*) is considered by dropping integrality constraints (7.9)-(7.10) as well as the subtour elimination constraints (7.5). In addition, the L-shaped

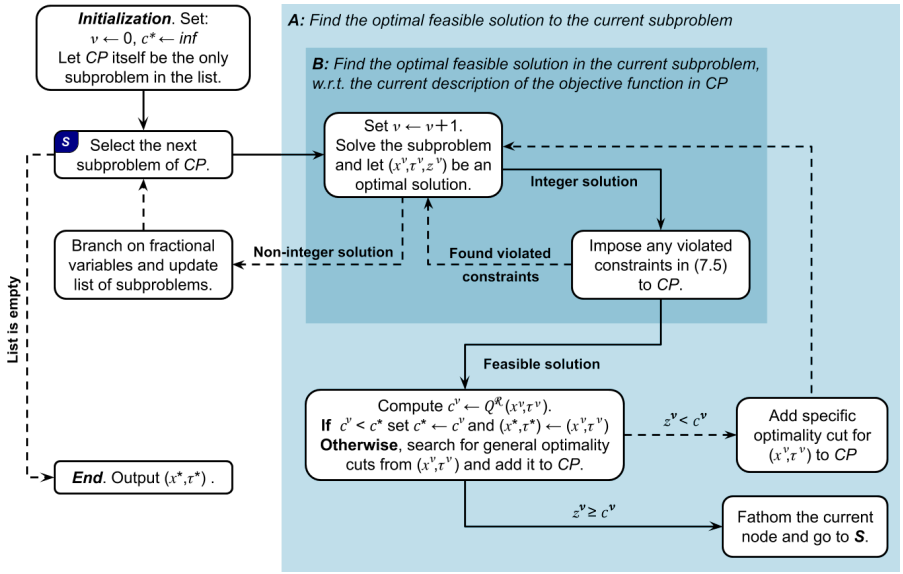


Figure 7.1: Flow-chart of the branch-and-cut algorithm.

method proposes to replace the nonlinear evaluation function $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ by a lower bounding variable θ. We hence solve a relaxed version of program (7.1)-(7.10), by defining the initial *CP*:

$$(CP) \quad \underset{x,\tau,z}{\text{Minimize}} \quad z \tag{7.12}$$

subject to constraints (7.2), (7.3), (7.4), (7.6), (7.7), (7.11) and $z \geq 0$. The *CP* is then iteratively modified by introducing integrality conditions throughout the branching process and by generating cuts from constraints (7.5) whenever a solution violates it. These are commonly called *feasibility cuts*. Let (x^v, τ^v, z^v) be an optimal solution to *CP*. In addition to feasibility cuts, a lower bounding constraint on z, so-called *optimality cut*, is generated whenever a solution comes with a wrong objective value, that is when $z^v < \mathcal{Q}^{\mathcal{R}}(x^v, \tau^v)$. This way z is gradually tightened upward in the course of the algorithm, approaching the objective value from below.

The branch-and-cut scheme is depicted in Figure 7.1. The main steps of the algorithm are as follows. Initially, the solution counter v and the cost c^* of the best solution found so far are set to zero and infinite, respectively. The list of subproblems is initialized in such a way that *CP* is the only problem in it. Thereafter, the following tasks are repeated until the list of subproblems becomes empty: 1) select a subproblem in the list and 2) find its optimal feasible solution and compare it with the best one found so far. When solving a particular subproblem, it is unlikely that *CP* contains the exact representation of the objective function through z. Instead, it is approximated by a set of lower bounding constraints. Therefore, whenever an optimal feasible solution (x^v, τ^v, z^v) is found at the end of frame *B*, it may happen that it is optimal with respect to *CP*, but not with respect to

Similarly to the subtour elimination constraints, lower bounding constraints (optimality cuts) are added dynamically to the model.

the real objective function. This can be easily checked by comparing the approximated cost z^ν with the real one $c^\nu = \mathcal{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$. If $z^\nu < c^\nu$, then (x^ν, τ^ν, z^ν) cannot be proven to be optimal since its approximated cost z^ν is wrong, and consequently better solutions may exist for the current subproblem.

Specific optimality cuts

We now present the valid optimality cuts we propose for the SS-VRPTW-CR and that we use in order to gradually strengthen the approximation of z in *CP*. Two families of optimality cuts will be considered: *specific* optimality cuts, which are only active at one feasible solution and *general* optimality cuts, which apply to a whole subtree of the branch and bound process. General optimality cuts are discussed later in Section 7.4.

Our optimality cuts are adapted from the classical ones presented in Laporte and Louveaux, 1993. Let (x^ν, τ^ν, z^ν) be an optimal solution to *CP* where x^ν, τ^ν are integer-valued. Let $A(x^\nu)$ be the set of triples (i, j, k) such that arc (i, j) is part of route k, *i.e.*,

$$A(x^\nu) = \{(i, j, k) : \ i, j \in W, k \in [1, K], x^\nu_{ijk} = 1\}$$

and $W(\tau^\nu)$ be the set of triples (i, l, k) such that vehicle k waits l units of time at vertex i, *i.e.*,

$$W(\tau^\nu) = \{(i, l, k) : \ i \in W, l \in H_0, p \in [1, K], \tau^\nu_{ilk} = 1\}.$$

Proposition 1. *The constraint*

$$z \geq \mathcal{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)\Big(\sum_{\substack{(i,j,k) \\ \in A(x^\nu)}} x_{ijk} + \sum_{\substack{(i,l,k) \\ \in W(\tau^\nu)}} \tau_{ilk} - |A(x^\nu)| - |W(\tau^\nu)| + 1\Big) \tag{7.13}$$

is a valid optimality cut for the SS-VRPTW-CR.

PROOF: Indeed, the integer solution (x^ν, τ^ν) is composed of exactly $|A(x^\nu)|$ arcs and exactly $|W(\tau^\nu)|$ variables in τ assigned to 1. For any different solution (x, τ, z), one must have either $\sum_{(i,j,p)\in A(x)} \leq |A(x^\nu)| - 1$ or $\sum_{(i,l,k)\in W(\tau)} \leq |W(\tau^\nu)| - 1$. Consequently, for any solution $(x, \tau) \neq (x^\nu, \tau^\nu)$,

$$\sum_{(i,j,p)\in A(x)} x_{ijk} + \sum_{(i,l,k)\in W(\tau)} \tau_{ilk} \ \leq \ |A(x^\nu)| + |W(\tau^\nu)| - 1. \tag{7.14}$$

By substituting into (7.13), we see that constraint (7.13) becomes $z \geq \mathcal{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$ only when $(x, \tau) = (x^\nu, \tau^\nu)$. Otherwise the constraint is dominated by $z \geq 0$. □

In practice, similarly to Laporte et al., 2002 we replace the classical cut of the form (7.13) by a numerically more stable "integer cut" of the form (7.14), cutting the current solution (x^ν, τ^ν) off the current subproblem, forcing the algorithm to move to a different one.

7.3 EXPERIMENTS

We consider recourse strategy $\mathcal{R}^\infty$ only, that is, $\mathcal{R}^q$ with uncapacitated vehicles (*i.e.*, $Q = \infty$). In order to evaluate the interest of exploiting stochastic knowledge, that is by modeling the problem as a SS-VRPTW-CR, the solutions are compared with a wait-and-serve policy which does not anticipate, *i.e.* in which vehicles are never relocated. The current experiments do not consider the generation of general optimality cuts, as discussed later in Section 7.4.

WAIT-AND-SERVE POLICY. In order to assess the contribution of our recourse strategies, we compare them with a policy ignoring anticipative actions. This *wait-and-serve* (w&s) policy takes place as follows. Vehicles begin the day at the depot. Whenever an online request r appears, it is accepted if at least one of the vehicles is able to satisfy it, otherwise it is rejected. If accepted, it is assigned to the *closest such vehicle* which then visits it as soon as it becomes idle. If there are several closest candidates, the least loaded vehicle is chosen. After servicing r (which lasts s_r time units), the vehicle simply stays idle at r's location until it is assigned another request or until it must return to the depot. Note that a request cannot be assigned to a vehicle if satisfying it prevents the vehicle from returning to the depot before the end of the horizon.

*The **wait-and-serve** policy suggests that vehicles only travel between appeared online requests, and never relocate preventively.*

Note that, whereas our recourse strategies for the SS-VRPTW-CR generalize to requests such that the time window starts later than the reveal time, in our instances we consider only requests where $e_r = \Gamma_r$. Doing it the other way would in fact require a more complex wait-and-serve policy, since the current version would be far less efficient and unrealistic in the case of requests with e_r significantly greater than Γ_r.

RELATIVE GAINS. In what follows, average results are always reported for the w&s policy. We randomly generate 10^6 scenarios according to the p_r probabilities. For each scenario, we apply the w&s approach to compute a number of rejected requests; finally, the average number of rejected requests is reported. The results are then always reported by means of average relative gains, in percentages, with respect to the w&s policy: the gain of a first-stage solution s computed is $\frac{\text{avg}-E}{\text{avg}}$, where E is the expected cost of s and avg is the average cost under the w&s policy.

		Branch&Cut (% gain after 12h)					
		scale = 1		scale = 2		scale = 5	
	w&s	$\mathcal{R}^{\infty}$	$\mathcal{R}^{q+}$	$\mathcal{R}^{\infty}$	$\mathcal{R}^{q+}$	$\mathcal{R}^{\infty}$	$\mathcal{R}^{q+}$
10c-5w-1	12.8	8.2	14.1	7.3*	12.8 *	4.4*	9.5*
10c-5w-2	10.8	-4.8	7.4	-4.8	7.4	-8.8*	0.5*
10c-5w-3	8.0	-46.9	-26.5	-46.9*	-26.5	-55.9*	-43.2
10c-5w-4	10.5	-10.9	0.9	-10.9*	0.9*	-10.9*	0.9*
10c-5w-5	8.4	-17.9	2.1	-17.9*	2.5	-20.5*	1.3*
#eval		8.10^4	5.10^4	10^5	9.10^4	10^5	9.10^4
10c+w-1	12.8	31.0	38.2	30.5	33.9	29.9	36.5
10c+w-2	10.8	17.9	18.8	13.2	16.9	15.2	23.6
10c+w-3	8.0	12.1	16.4	14.6	32.1	-4.3	4.4
10c+w-4	10.5	8.2	12.3	10.2	14.2	3.5	28.1
10c+w-5	8.4	19.7	30.4	5.3	22.6	-0.5	20.3
#eval		5.10^4	3.10^4	7.10^4	6.10^4	10^5	9.10^4

Table 7.1: Results on small instances: $n = 10$, $K = 2$, $Q = \infty$ and $\beta = 60$. Branch&Cut for *10c-5w-x* (resp. *10c+w-x*) instances, $m = 5$ and $W \cap C = \emptyset$ (resp. $m = n$ and $W = C$). For each instance, we give the average cost over 10^6 sampled scenarios using the *wait-and-serve* policy (*w&s*) and the gain of the best solution found by Branch&Cut within a time limit of 12 hours, using either $\mathcal{R}^{\infty}$ or $\mathcal{R}^{q+}$. Results marked with a star (*) have been proven optimal. #eval gives the average number of expectation computations for each run, corresponding to the added optimality cuts (7.13). Grey cells show best results.

SETUP. Experiments have been done on a cluster composed of 64-bit AMD Opteron 1.4-GHz cores. The code is developed in *C++11* with *GCC4.9*, using *-O3* optimization flag. The current source code of our library for (SS-)VRPs is available from the online repository: `bitbucket.org/mstguillain/vrplib`.

7.3.1 *Results*

The results obtained on instances *10c-5w-x* and *10c+w-x* are displayed in Table 7.1. The scale term represents the accuracy of the horizon. When scale is one, the original horizon is considered, with its 480 time units. With scale = 5, the time horizon, as well as all the time attributs of the problem (travel times, time windows, *etc.*) is squeezed to $480/5 = 96$ time units, hence simplifying (and thereby approximating!) the computation of the objective function. Such approximation techniques are further studied in the next chapter.

Consequently to the enumerative nature of the proposed branch-and-bound method, which is due to the lack of general optimality cuts,

experimentations showed a very poor efficiency of the exact algorithm. Its performances, *even given 12 hours of computation time*, do not permit to prove optimality on instances involving more than five waiting vertices, even when using a simplified horizon.

7.4 GENERAL OPTIMALITY CUTS

We now introduce optimality cuts for the SS-VRPTW-CR that provide a useful bound at both integer and non-integer solutions. Based on those bounds, a node involving a non-integer solution can potentially be fathomed during the branch and cut process, hence pruning the subset of feasible solutions down the current branch. Following Hjorring and Holt, we refer to these as *general optimality cuts*, in opposition to *specific optimality cuts* of the form (7.13).

Without general optimality cuts, our branch-and-cut algorithm is nothing but a fancy ***enumeration*** *of the feasible first-stage solutions.*

Recall that from a current integer and first stage feasible solution (x^ν, τ^ν) to *CP*, we can easily construct the set of vehicle routes, from sequence $\langle 0, w_1, ..., w_{m'}, 0\rangle_1^\nu$ to sequence $\langle 0, w_1, ..., w_{m''}, 0\rangle_K^\nu$, and that each waiting vertex in each route is associated a waiting time in τ^ν. Finally, provided (x^ν, τ^ν) the sequences $\pi_1, ..., \pi_K$ of ordered potential requests is obtained by following the request assignment rule involved in the recourse strategy. These sequences of requests then allow the computation of the solution's expected cost.

Provided the appropriate computation of $\mathrm{P}\{r \in A^h\}$, using either (5.7) or (5.16), observe that equation (5.6) sums only positive terms and that the f-probabilities are computed recursively from previous requests. Consequently, a lower bound $\widetilde{\mathscr{Q}}^{\mathcal{R}}$ on $\mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$ can be easily obtained by considering a set of request sequences $\widetilde{\pi}_1, ..., \widetilde{\pi}_K$, where $\widetilde{\pi}_k = \langle r_1, r_2, ..., r_{\rho' \leq \rho}\rangle$ is a prefix of $\pi_k = \langle r_1, r_2, ..., r_\rho\rangle$ for route $k \in [1, K]$:

$$\widetilde{\mathscr{Q}}^{\mathcal{R}}(\widetilde{\pi}_1, ..., \widetilde{\pi}_K) = \sum_{k=1}^{K} \sum_{r \in \widetilde{\pi}_k} p_r - \mathrm{P}\{r \in A^h\}. \tag{7.15}$$

Given a first stage solution (x^ν, τ^ν) involving prefixes $\widetilde{\pi} = \widetilde{\pi}_1, ..., \widetilde{\pi}_K$, then $\widetilde{\mathscr{Q}}^{\mathcal{R}}(\widetilde{\pi})$ is therefore a valid lower bound for any first stage solution (x', τ') that involves the same set of prefixes $\widetilde{\pi}$.

We designate by $A(\widetilde{\pi})$ and $W(\widetilde{\pi})$ the minimal sets of both arcs $(i, j, p) \in W_0^2 \times K$ and waiting times $(i, l, p) \in W \times H \times K$, for which the corresponding variable x_{ijp}^ν and τ_{ilp}^ν must be set to one in a first stage solution (x^ν, τ^ν) in order to obtain the prefixes $\widetilde{\pi} = \widetilde{\pi}_1, ..., \widetilde{\pi}_K$. Notice that request prefixes are closely related to the concept of partial route proposed in Hjorring and Holt, 1999, except that we consider prefixes of request sequences instead of partial routes.

Proposition 2. *Let $(x^\nu, \tau^\nu, \theta^\nu)$ be an optimal solution to* CP. *Let $\widetilde{\pi}^\nu = \widetilde{\pi}_1, ..., \widetilde{\pi}_K$ be a set of prefixes from (x^ν, τ^ν). Assuming $\widetilde{\mathscr{Q}}^{\mathcal{R}}(\widetilde{\pi}^\nu)$ to be a lower bound on the expected cost $\mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$ of any solution (x, τ), such that*

$x_{ijp} = 1$ for all $(i,j,p) \in A(\widetilde{\pi}^\nu)$, $\tau_{ilp} = 1$ for all $(i,l,p) \in W(\widetilde{\pi}^\nu)$ and all other variables are unspecified, the constraint

$$\theta \geq \widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi})\Big(\sum_{(i,j,p)\in A(\widetilde{\pi})} x_{ijp} + \sum_{(i,l,p)\in W(\widetilde{\pi})} \tau_{ilp} - |A(\widetilde{\pi})| - |W(\widetilde{\pi})| + 1\Big) \tag{7.16}$$

is a valid general optimality cut for the SS-VRPTW-CR.

PROOF: Let (x', τ', θ') be an optimal solution to *CP*. By substituting into (7.16), the constraint becomes $\theta \geq \widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi}^\nu)$ if and only if (x', τ') is such that $x'_{ijp} = 1$ for all $(i,j,p) \in A(\widetilde{\pi}^\nu)$ and $\tau'_{ilp} = 1$ for all $(i,l,p) \in W(\widetilde{\pi}^\nu)$. Otherwise, (7.16) is dominated by $\theta \geq 0$. In particular, for any solution (x', τ', θ'), the inequality reduces to $\theta \geq \widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi}^\nu)$ if and only if there exists, associated to (x', τ'), a set of prefixes $\widetilde{\pi}'$ such that $A(\widetilde{\pi}^\nu) \subseteq A(\widetilde{\pi}')$. By definition of $\widetilde{\mathcal{Q}}^{\mathcal{R}}$, it implies $\widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi}^\nu) \leq \widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi}')$ and consequently the inequality $\theta \geq \widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi}^\nu)$ is then valid for both (fractional) solutions (x^ν, τ^ν) and (x', τ'). □

In order to generate a general optimality cut (7.16) from a (fractional) solution (x^ν, τ^ν), we then need to identify the prefixes $\widetilde{\pi}^\nu$ and the corresponding sets $A(\widetilde{\pi}^\nu)$ and $W(\widetilde{\pi}^\nu)$. As described in Section 5.2 however, each request is assigned to a feasible waiting vertex depending on both its reveal time and the set of feasible waiting vertices for that request, that is, the set of waiting vertices visited at a moment allowing to serve the request. A potential request can thus be assigned to a visited vertex only if all the visited vertices are known. Given a (fractional) solution, this significantly reduces the number of candidate set of prefixes, which is bounded by $\mathcal{O}(h)$. This is illustrated in Figure 7.2. Finally, given both a (fractional) solution (x^ν, τ^ν) and a particular set of prefixes $\widetilde{\pi}^\nu$, the sets $A(\widetilde{\pi}^\nu)$ and $W(\widetilde{\pi}^\nu)$ can be trivially obtained.

We now explain how a set of prefixes is selected from a solution. Notice that at a current node of the branch and bound process, $\widetilde{\mathcal{Q}}^{\mathcal{R}}$ provides a useful lower bound only if it is $\geq \theta^*$, the cost of the best feasible solution found so far. Moreover, the smaller the number of decision variables involved in a general cut, the higher the number of fractional and integer solutions for which the cut will be active. Given a (fractional or integer) solution (x, τ), we hence generate at most one general cut, based on a set $\widetilde{\pi} = \widetilde{\pi}_1, ..., \widetilde{\pi}_K$ of request sequence prefixes chosen such that $\widetilde{\mathcal{Q}}^{\mathcal{R}}(\widetilde{\pi}) \geq \theta^*$ and that $|A(\widetilde{\pi})| + |W(\widetilde{\pi})|$ is as small as possible.

The f-probabilities involved in equation (7.15) can be computationally expensive to compute. In practice and as indicated by the flow chart of Figure 7.1, it is however more convenient to compute general optimality cuts from an integer solution. Given a feasible integer solution (x^ν, τ^ν) to *CP*, its optimality with respect to the cur-

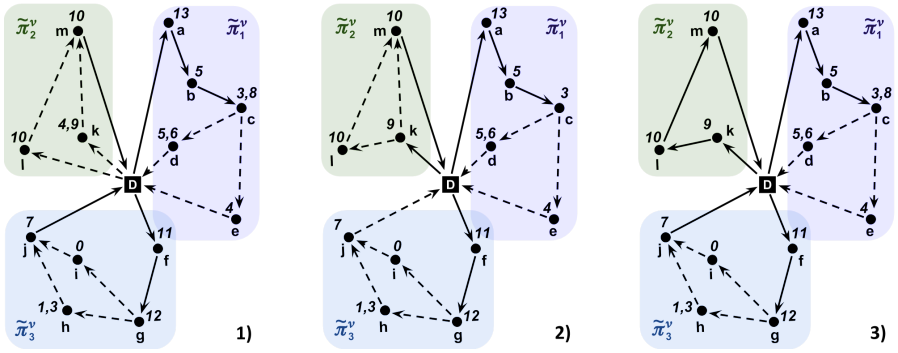


Figure 7.2: Examples of fractional solutions that could result from the LP relaxation. For the ease of the illustration, only a subset of the fractional edges are represented. Each waiting vertex is designated by a letter, the associated waiting times are given as integers instead of binary variables whereas arrows represent the routes. The illustrated solution has both integer and non-integer valued variables. In solution 1, we have $x_{0a1} = 1$ whereas x_{cd1}, x_{ce1} are both fractional and sum to 1. Similarly, $\tau_{b51} = 1$ whereas τ_{c31}, τ_{c81} are both fractional (and sum to 1). Given fractional solutions 1, 2 or 3, which are the candidate sets of prefixes (and the corresponding decision variables) that could be considered for deriving a cut of the form (7.16) ? In fractional solution 1, the set of waiting vertices about to be visited at time $t = 1$ is not known (at least concerning vehicle $k = 2$), and none of our assignment heuristics can be applied to any potential request. In fractional solution 2, the set of (about to be) visited vertices is known from time $t = 1$ to $t' = \min\{t_1, t_2, t_3\}$, with $t_1 = d_{0,a} + 13 + d_{a,b} + 5 + d_{b,c} + 3$, $t_2 = d_{0,K} + 9$ and $t_3 = d_{0,f} + 11 + d_{f,g} + 12$. If we apply assignment $A1$, a set of prefixes is obtained by considering all the requests $r = (c, \Gamma)$ such that $\Gamma \leq t'$. Prefixes are obtained similarly from solution 3, except that since the waiting time at vertex c is not known, we have $t_1 = d_{0,a} + 13 + d_{a,b} + 5 + d_{b,c}$.

rent subproblem must be checked by computing the exact expected cost $\mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$. Consequently, all the f-probabilities for the current solution are available and can be directly reused in order to identify interesting sets of partial routes, and finally come out with useful general optimality cuts.

All-in-one computation of optimality cuts

By using a set $\widetilde{\pi}^\nu$ of prefixes obtained from a solution $(x^\nu, \tau^\nu, \theta^\nu)$, from (7.15) we know that $\widetilde{\mathscr{Q}}^{\mathcal{R}}(\widetilde{\pi}) \leq \mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$. The condition $\theta^* < \widetilde{\mathscr{Q}}^{\mathcal{R}}(\widetilde{\pi})$ is therefore sufficient to discard the solution. This idea was first proposed by Angulo et al., 2016, which suggested to improve the L-shaped method so that unnecessary (and costly) computations of the real expected cost are prevented by the use of an approximation function. In practice, that means that given a solution $(x^\nu, \tau^\nu, \theta^\nu)$ where (x^ν, τ^ν) is integral, the computation of $\mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$, which iterates over the ordered sequences of potential requests within each vehicle route, can be interrupted as soon as the current value of the summation in (5.6) exceeds θ^*.

In addition, given the same solution $(x^\nu, \tau^\nu, \theta^\nu)$ suppose we intertwine the route computations such that, before considering time unit $t+1$, we consider all the potential requests of all the K vehicle routes associated to time unit t. Equation (5.6) is then equivalently rewritten as:

$$\mathscr{Q}^{\mathcal{R}}(x, \tau) = \sum_{t \in H} \sum_{k=1}^{K} \sum_{r \in \pi_k^t} p_r - \mathrm{P}\{r \in A^h\}. \tag{7.17}$$

Let $\pi_k^t \subset \pi_k$ be the subsequence of requests assigned to vehicle k such that $t = \Gamma_r$. In that case, if the computation is rather stopped whenever the summation exceeds θ^*, then the set $\widetilde{\pi}^\nu$ of prefixes that corresponds to the set of requests considered until then is both useful (that is, $\widetilde{\mathscr{Q}}^{\mathcal{R}}(\widetilde{\pi}^\nu) \geq \theta^*$) and of minimal size. A general optimality cut of the form (7.16) is then generated.

In the case the current solution reveals to be the best encountered so far with $\mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu) < \theta^*$, the computation of (7.17) completes and delivers its real expected cost (which becomes the new θ^*). It is then very likely that $\theta^\nu < \mathscr{Q}^{\mathcal{R}}(x^\nu, \tau^\nu)$, in which case a specific optimality cut of the form (7.13) must be imposed to *CP*.

This "all-in-one" computation therefore allows simultaneous computation of both general and specific optimality cuts, if any, but also provides potential savings on the computation. It however comes at the price of a higher worst case complexity: $\mathcal{O}(nh^3Q)$ (resp. $\mathcal{O}(n^2h^4Q)$) for recourse strategy $\mathcal{R}^q$ (resp. $\mathcal{R}^{q+}$).

7.5 CONCLUSION AND FURTHER WORK

The branch-and-cut approach considered in this paper may not be perfectly designed for the SS-VRPTW-CR. First notice that each optimality cut of the form (7.13) is likely to be active at only one feasible solution. As pointed out by Hjorring and Holt, 1999, if only these cuts are to be added to *CP*, then our branch-and-cut method must generate such a cut for almost each feasible first stage solution. Thus, although not trivially doable, general optimality cuts that are active at fractional solutions should be devised as well.

Besides the lack of general optimality cuts, another true limitation is the fact that, unlike the exact enumerative method, the SIP formulation (7.1)-(7.10) does not exploit the following property: there is always an optimal solution that *(a)* use all the vehicles and *(b)* use all the available waiting time. As a consequence, the solution space of the SIP formulation is significantly bigger.

In Section 7.4, we propose a set of general optimality cuts for our SIP formulation. However, although mathematically sound, they have not been empirically tested yet, hence they come with no guarantee of efficiency. Furthermore, branch-and-cut is only one method among all the possible approaches that could be tested on the SS-VRPTW-CR. In particular set partitioning methods such as column generation, which becomes commonly used for stochastic VRPs, could also provide interesting results.

8

HEURISTIC APPROACH: PROGRESSIVE FOCUS SEARCH

This chapter introduces a new meta-heuristic, called *Progressive Focus Search* (PFS), for solving computationally demanding blackbox optimization problems, such as the SS-VRPTW-CR.

We also introduce a new public benchmark for the SS-VRPTW-CR, based on real-word data coming from the city of Lyon. Experimental results on this benchmark show that PFS obtains better results than a classical search, that solves the problem without introducing approximation factors. By comparing with a basic (yet realistic) wait-and-serve policy which does not exploit stochastic knowledge, we show that our stochastic models are particularly beneficial when the number of vehicles increases and when time windows are tight. In other words, the more the problem configuration is complicated and demanding in terms of anticipation, the higher are the gains of using our SS-VRPTW-CR model. Eventually, all these experiments show that allowing vehicles to wait directly at potential customer vertices lead to better expected results than using separated relocation vertices.

Section 8.1 describes the proposed meta-heuristic. We then show in Section 8.2 how it can be instantiated in the case of the SS-VRPTW-CR. Section 8.3 experimentally validates the solution method on the Lyon benchmark presented in Chapter 6.

8.1 PROGRESSIVE FOCUS SEARCH

Solving a static stochastic optimization problem, such as the SS-VRPTW-CR, involves finding values for a set of first-stage decision variables that optimize an expected cost with respect to some recourse strategy:

$$\min_{x} \quad \mathcal{Q}^{\mathcal{R}}(x), \quad x \in X$$

Solving this kind of problem is always challenging. Besides the exponential size of the (first-stage) solution space X, the nature of the objective function $\mathcal{Q}^{\mathcal{R}}$, an expectation, is usually computationally demanding. Because enumerating all possible scenarios is usually impossible in practice, some approaches tend to circumvent this bottleneck by restricting the set of considered scenarios, using for example the sample average approximation method (Ahmed and Shapiro, 2002). In some cases, expectations may be directly computed in (pseudo) polynomial time, by reasoning on the random variables themselves rather than on the scenarios. However, the required computational

effort depends on the recourse strategy $\mathcal{R}$ and usually remains very demanding, as it is the case for the SS-VRPTW-CR.

*The **Progressive Focus Search** metaheuristic aims at laveraging the complexity of both the first-stage solution space and the evaluation of the solutions' expected cost.*

The Progressive Focus Search (PFS) metaheuristic aims at addressing these issues with two approximation factors, intended to reduce the size of the solution space and the complexity of the objective function. The initial problem P_{init} is simplified into a problem $P_{\alpha,\beta}$ having simplified objective function and solution space. Parameters α and β define the approximation factors of the objective function and of the solution space, respectively, and $P_{\alpha,\beta} = P_{\text{init}}$ when $\alpha = \beta = 1$. Whenever $\alpha > 1$ or $\beta > 1$, the optimal cost of $P_{\alpha,\beta}$ is an approximation of that of P_{init}. Starting from some initial positive values for α and β, the idea of PFS is to progressively decrease these values using an update policy. The simplified problem $P_{\alpha,\beta}$ is iteratively optimized for every valuation of (α, β), using the best solution found at the end of one iteration as starting point in the solution space for the next iteration.

The definition of the simplified problem $P_{\alpha,\beta}$ depends on the problem to be solved. In Sections 8.1.1 and 8.1.2, we give some general principles concerning α and β and describe how to apply them to the case of the SS-VRPTW-CR. In Section 8.1.3, we describe the generic PFS metaheuristic.

8.1.1 *Reducing objective function computational complexity with α*

We assume the expected cost to be computed by filling matrices in several dimensions. In order to reduce the complexity, some of these dimensions must be scaled down. This is achieved by changing the scale of the input data and the decision variable domains related to the selected dimensions, dividing the values by the scale factor α and rounding to integer if necessary.

For example, in the SS-VRPTW-CR the dimensions considered at computing the objective function are: the number of waiting vertices n, the vehicle capacity Q, and the time horizon h. Let $h = 18000$ be the time horizon in the initial problem, corresponding to five hours in units of one second. If we choose to reduce the time dimension with respect to a scale factor $\alpha = 60$, then all durations in the input data (travel times, service times, time windows, etc.) are rounded to the nearest multiple of 60. Thus, the time horizon in the simplified problem $P_{\alpha,\beta} = P_{60,\beta}$ is of $h_{60} = 300$, corresponding to a five-hour time horizon in units of one minute. The domains of waiting times decision variables are reduced accordingly, scaled from $[1, 18000]$ in $P_{1,\beta}$ to $[1, 300]$ in $P_{60,\beta}$.

Similarly, if we choose to reduce the vehicle capacity dimension with respect to a scale factor $\alpha = 1000$, and if the vehicle capacity in the initial problem is $Q = 500000$, *e.g.* 500 kg in steps of 1 g, then all demands must be rounded to multiples of 1000. The capacity in

$P_{1000,\beta}$ becomes $Q_{1000} = 500$, thus 500 kg in units of 1 kg. When scaling dimensions of different nature, such as time and capacity, different scale factors should be considered, leading to a vector α.

Experiments (in Section 8.3) have shown us that the closer α is to 1, the more accurate the approximation of the actual objective function is. Progressively reducing α during the search process allows us to quickly compute rough approximations at the beginning of the search process, when candidate solutions are usually far from being optimal, and spend more time computing more accurate approximations at the end of the search process, when candidate solutions get closer to optimality.

8.1.2 *Simplifying the solution space size with β*

When applying a scaling factor α, for consistency reasons the nature of the scaled input data may impose to the domains of some decision variables to be reduced accordingly. Yet the solution space can further simplified by reducing the domains of (part of) the remaining decision variables, or even by further reducing the same decision variables. Let $Dom(v)$ be the initial set of values that may be assigned to v, that is, the domain of a decision variable v. Domain reduction is not necessarily done for all decision variables, but only for a selected subset of them, denoted as V_β. The simplified problem is obtained by selecting $|Dom(v)|/\beta$ values and only considering these candidate values when searching for solutions, for each decision variable $v \in V_\beta$. Ideally, the selection of this subset of values should be done in such a way that the selected values are evenly distributed within the initial domain $Dom(v)$. We note $Dom_{\alpha,\beta}(v)$, the domain of a decision variable v in the simplified problem $P_{\alpha,\beta}$.

For example, in the SS-VRPTW-CR a subset of decision variables defines the waiting times on the visited waiting vertices: τ_w defines the waiting time on w, with $Dom(\tau_w) = [1, h]$. If the temporal dimension is not scaled with respect to α, or if $\alpha = 1$, then $Dom_{\alpha,\beta}(\tau_w)$ is reduced to a subset of $[1, h]$ that contains h/β values. To ensure that these values are evenly distributed in $[1, h]$, we may keep multiples of β. However, if the temporal dimension is scaled with respect to α, the selected values must thereafter be scaled.

Another subset of decision variables in the SS-VRPTW-CR defines the waiting vertices to be visited by the vehicles. The initial decision variable domains are then equivalent to W. Reducing the domains of these decision variables can be achieved by restricting to a subset of W that contains $|W|/\beta$ waiting vertices. To ensure that these values are evenly distributed in the space, we may use geographical clustering techniques. Such approach will be considered in Chapter 9.

Progressively decreasing the value of β allows us to progressively move from diversification to intensification: at the beginning of the

search process, there are fewer candidate values for the decision variables of V_β. The solution method is therefore able to move quickly towards more fruitful regions of the search space. For minimization (resp. maximization) problems, we can easily show that the optimal solution of a simplified problem $P_{\alpha,\beta}$ is an upper (resp. lower) bound of the optimal solution of the problem $P_{\alpha,1}$; this is a direct consequence of the fact that every candidate solution of $P_{\alpha,\beta}$ is also a candidate solution of $P_{\alpha,1}$.

8.1.3 *PFS algorithm*

PFS requires the following input parameters:

- An initial problem P_{init};
- Initial values (α_0, β_0) for α and β, as well as final values $(\alpha_{\min}, \beta_{\min})$;
- An update policy $\mathcal{U}$ that returns the new values α_{i+1} and β_{i+1} given α_i and β_i;
- A computation time policy $\mathcal{T}$ such that $\mathcal{T}(\alpha, \beta)$ returns the time allocated for optimizing $P_{\alpha,\beta}$;
- A solution algorithm Θ such that, given a problem P, an initial solution s, and a time limit δ, $\Theta(P, s, \delta)$ returns a possibly improved solution s' for P.

PFS is designed to be used with any local search based optimization algorithm.

PFS is described in Algorithm 4. At each iteration i, the simplified problem P_{α_i,β_i} is built (line 3), and the current solution s is updated accordingly (line 3): every value assigned to a decision variable which is concerned by the scale factor α is updated with respect to the new scale α_i, and if a value assigned to a decision variable does not belong to the current domain associated with α_i and β_i, then it is replaced with the closest available value. Note that the updated solution may not be a feasible solution of P_{α_i,β_i} (because of value replacements and rounding operations on input data). Therefore the optimizer Θ must support starting with infeasible solutions.

Algorithm 4: Progressive Focus Search (PFS)

1 Initialize i to 0 and construct an initial solution s to problem P_{init};
2 **repeat**
3 Build problem P_{α_i,β_i} and update the current solution s to P_{α_i,β_i};
4 $s \leftarrow \Theta(P_{\alpha_i,\beta_i}, s, \mathcal{T}(\alpha_i, \beta_i))$;
5 $(\alpha_{i+1}, \beta_{i+1}) \leftarrow \mathcal{U}(\alpha_i, \beta_i)$;
6 Increment i
7 **until** $\alpha_{i-1} = \alpha_{min} \ \wedge \ \beta_{i-1} = \beta_{min}$;
8 **if** $\alpha_{min} > 1$ **then** Update the current solution s to $P_{1,1}$;
9 **return** s;

Algorithm 5: Local search to compute a first stage solution of SS-VRPTW-CR

```
1 Let (x, τ) be an initial feasible first stage solution.
2 Initialize the neighborhood operator op to 1
3 while some stopping criterion is not met do
4     Select a solution (x', τ') at random in N_op(x, τ)
5     if some acceptance criterion is met on (x', τ') then
6         set (x, τ) to (x', τ') and op to 1
7     else change the neighborhood operator op to op % n_op + 1 ;
8 return the best first stage solution computed during the search
```

Algorithm Θ is then used to improve s with respect to the simplified problem P_{α_i,β_i} within a CPU time limit defined by the computation time policy $\mathcal{T}$ (line 4). Finally, new values for α and β are computed, according to the update policy $\mathcal{U}$ (line 5). This iterative optimization process stops when $\alpha_{i-1} = \alpha_{\min}$ and $\beta_{i-1} = \beta_{\min}$, *i.e.*, when the last optimization of s with Θ has been done with respect to the targeted level of accuracy defined by $(\alpha_{\min}, \beta_{\min})$. To ensure termination, we assume that the update policy $\mathcal{U}$ eventually returns $(\alpha_{\min}, \beta_{\min})$ after a finite number of calls. Finally, if the final value of α is larger than 1, so that s is a scaled solution, then s is scaled down to become a solution of the initial problem $P_{1,1}$ (line 8).

8.2 PFS FOR THE SS-VRPTW-CR

In this section, we describe how the Progressive Focus Search meta-heuristic can be instantiated in order to tackle the SS-VRPTW-CR.

8.2.1 *Local search optimizer*

We use a simple local search (LS) algorithm as optimizer Θ, described in Algorithm 5. It implements a Simulated Annealing (Kirkpatrick et al., 1983) meta-heuristic for approximating the optimal first stage solution (x, τ), minimizing $\mathcal{Q}^{\mathcal{R}}(x, \tau)$. The computation of $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ is performed according to equations of Section 5.2.2 (and further section 5.3) and is considered from now as a black box. Starting from an initial feasible first stage solution (x, τ), Algorithm 5 iteratively modifies it by using a set of $n_{op} = 9$ neighborhood operators. At each iteration, it randomly chooses a solution (x', τ') in the current neighborhood (line 4), and either accepts it and resets the neighborhood operator op to the first one (line 5), or rejects it and changes the neighborhood operator op to the next one (line 6). At the end, the algorithm simply returns the best solution (x^*, τ^*) encountered so far.

INITIAL SOLUTION AND STOPPING CRITERION. The initial first stage solution is constructed by randomly adding each waiting vertex in a route $k \in [1, K]$. All waiting vertices are thus initially part of the solution. The stopping criterion depends on the computational time dedicated to the algorithm.

NEIGHBORHOOD OPERATORS. We consider 4 wellknown operators for the VRP: relocate, swap, inverted 2-opt, and cross-exchange (see Kindervater and Savelsbergh, 1997; Taillard et al., 1997 for detailed description). In addition, we introduce 5 new operators dedicated to waiting vertices: 2 for either inserting or removing from W^x a waiting vertex w picked at random, 2 for increasing or decreasing the waiting time τ_w at random vertex w $\in W^x$, and 1 that transfers a random amount of waiting time units from one waiting vertex to another.

ACCEPTANCE CRITERION. We use a Simulated Annealing acceptance criterion (Kirkpatrick et al., 1983). Improving solutions are always accepted, while degrading solutions are accepted with a probability that depends on the degradation and on a temperature parameter, *i.e.*, the probability of accepting (x', τ') is $e^{-\frac{1-\mathcal{Q}^{\mathcal{R}}(x,\tau)/\mathcal{Q}^{\mathcal{R}}(x',\tau')}{T}}$. The temperature T is updated by a *cooling factor* $0 < f_T < 1$ at each iteration of Algorithm 5: $T \leftarrow \alpha \cdot T$. During the search process, T gradually evolves from an initial temperature T_{init} to nearly zero. A restart strategy resets the temperature to $T \leftarrow T_{\text{init}}$ each time T decreases below a fixed limit T_{min}. In all experiments, SA parameters were set to $T_{\text{init}} = 2$, $T_{\text{min}} = 10^{-6}$, and $f_T = 0.95$.

8.2.2 *Scale factor α*

In the initial problem $P_{1,1}$, temporal data is expressed with a resolution of one-minute time units. The α factor is used to scale down this temporal dimension. The time horizon is scaled down to round(h/α), so that each time step in $P_{\alpha,\beta}$ has a duration of α minutes. Every temporal input value (travel times $d_{i,j}$, reveal times Γ_r, service times s_r, and time windows $[e_r, l_r]$) is scaled from its initial value t to round(t/α). Rounding operations are chosen in such a way that the desired quality of service is never underestimated by scaled data: l_r is rounded down while all other values are rounded up. This ensures that a feasible first stage of a simplified problem $P_{\alpha,\beta}$ always remains feasible once adapted to $P_{1,1}$.

8.2.3 *Domain reduction factor β*

The decision variables concerned by domain reductions are waiting time variables: $V_\beta = \{\tau_w : w \in W\}$. In $P_{1,1}$, we have $Dom(\tau_w) = [1, h]$. Domains are reduced by selecting a subset of $|Dom(\tau_w)|/\beta$ values,

evenly distributed in $[1, h]$. As the temporal dimension is also scaled with respect to α, selected values are scaled down: $Dom_{\alpha,\beta}(\tau_w) = \{\text{round}(i/\alpha) : i \in [1, h], i \mod \beta = 0\}$.

It is both meaningless (for vehicle drivers) and too expensive (for the optimization process) to design first-stage solutions with waiting times that are precise to the minute. Hence, in our experiments the domain of every waiting time decision variable is always reduced by a factor $\beta \geq 10$. When $\beta = 10$, waiting times are multiples of 10 minutes. When $\alpha = 1$ and $\beta = 10$, we have $Dom_{1,10}(\tau_w) = \{10, 20, 30, \ldots, 480\}$, but temporal data (travel and service times, time windows, *etc.*) are precise to the minute.

8.3 EXPERIMENTS AND RESULTS

We consider the three recourse strategies $\mathcal{R}^\infty$, $\mathcal{R}^q$ and $\mathcal{R}^{q+}$. Recourse strategy $\mathcal{R}^\infty$ was introduced by Saint-Guillain et al., 2017 to deal with uncapacitated vehicle, whereas $\mathcal{R}^q$ presented in Section 5.2 generalizes it to take customer integer demands into account. Finally, recall that $\mathcal{R}^{q+}$ is an improved, cleverer version of $\mathcal{R}^q$, which is however computationally more demanding.

We compare the respective contribution and applicability of the three strategies. We also show how to combine them in order to take the best of each, using several variations of PFS. An exact method allows us to measure optimality gaps, in order to assess the quality of the solutions found by PFS. In order to evaluate the interest of exploiting stochastic knowledge, the solutions are compared with the wait-and-serve policy, previously presented in Section 7.3. As with Section 7.3, the results are reported by means of average relative gains, in percentages, with respect to the w&s policy.

SETUP. Experiments have been done on a cluster composed of 64-bit AMD Opteron 1.4-GHz cores. The code is developed in *C++11* with *GCC4.9*, using *-O3* optimization flag. The current source code of our library for (SS-)VRPs is available from the online repository: `bitbucket.org/mstguillain/vrplib`.

8.3.1 *Experiments on small instances*

We consider small test instances, having $n \in \{10, 20\}$ customer vertices. Furthermore, PFS is here instantiated such that we perform only a single optimization step (lines 2-7 of Algorithm 4): $\alpha_0 = \alpha_{\min}$ and $\beta_0 = \beta_{\min}$. The simplified problem $P_{\alpha,\beta}$ is therefore first optimized for a duration of T seconds, and the returned solution is adapted with respect to the initial problem $P_{1,1}$, ensuring that all results are expressed according to the original input data. This limited experimental setting,

while ignoring the impact of performing several optimization steps in PFS, aims at determining:

1. Whether the loss of precision, introduced by α and β, is counterbalanced by the fact that the approximation $P_{\alpha,\beta}$ is easier to solve than the initial problem.

2. The impact of avoiding pointless trips in recourse strategy $\mathcal{R}^{q+}$, compared with simpler (but computationally less demanding) strategy $\mathcal{R}^{q}$.

3. The interest of exploiting stochastic knowledge, by comparing the expected costs of SS-VRPTW-CR solutions with their average costs under the w&s policy.

4. The quality of the solutions computed by the LS algorithm under different scale factors. These are compared with optimal solutions obtained with the exact method. When $\alpha > 1$ or $\beta > 1$, the exact method solves $P_{\alpha,\beta}$, and the results are reported according to the final solution, scaled back to $P_{1,1}$.

8.3.1.1 *Enumerative exact method.*

We use an alternative exact method in order to cope with the limitations of the branch-and-cut algorithm.

The branch-and-cut algorithm described in Chapter 7 suffers from important limitations that affect its performances. In fact, it is not able to prove optimality for most of the test instances, even those of very small size (five waiting vertices). In order to assess the ability of our algorithms to find (*near*-)optimal solutions, we devise a simple enumerative optimization method which is able to compute optimal solutions on these small instances. To that end, the solution space is restricted to the solutions that *(a)* use all available vehicles and *(b)* use all the available waiting time. Indeed, if $K \leq |W|$, then on the basis of any optimal solution which uses only a subset of the available vehicles, a solution of the same cost can be obtained by assigning an idle vehicle to either a non-visited waiting vertex (if any) or the last visited vertex of any non-empty route (visiting at least two waiting locations), so that *(a)* does not remove any optimal solution. Furthermore, if an optimal first-stage solution contains a route for which the vehicle returns to the depot before the end of the horizon, adding the remaining time to the last visited waiting vertex will never increase the expected cost of the solution, so that *(b)* is also valid. The resulting solution space is then recursively enumerated in order to find the first-stage solution with the optimal expected cost.

8.3.1.2 *Impact of the scale factor α*

Table 8.1 shows the average gains, in percentages, of using an SS-VRPTW-CR solution instead of the w&s policy, for small instances composed of $n = 10$ customer vertices with $K = 2$ uncapacitated

		Exact (% gain after 30 minutes)					
		$\alpha = 1$		$\alpha = 2$		$\alpha = 5$	
	w&s	$\mathcal{R}^q$	$\mathcal{R}^{q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q+}$
10c-5w-1	12.8	8.9*	15.4	7.3*	12.8*	4.4*	9.5*
10c-5w-2	10.8	-4.8*	7.4	-4.8*	7.4*	-8.8*	0.5*
10c-5w-3	8.0	-46.9*	-26.5	-46.9*	-26.5*	-55.9*	-43.2*
10c-5w-4	10.5	-10.9*	0.9	-10.9*	0.9*	-10.9*	0.9*
10c-5w-5	8.4	-17.9*	2.5	-17.9*	2.5	-20.5*	0.5*
#eval		10^4		10^4		10^4	10^4
10c+w-1	12.8	35.3	34.4	35.3	34.4	32.7	26.5
10c+w-2	10.8	28.1	19.1	30.1	21.5	30.1	29.7
10c+w-3	8.0	14.4	17.1	18.8	17.1	13.3	13.7
10c+w-4	10.5	7.8	11.0	12.4	11.6	7.8	11.4
10c+w-5	8.4	3.5	8.9	8.4	6.6	23.7	1.7
#eval							

		PFS (% gain after 5 minutes)					
		$\alpha = 1$		$\alpha = 2$		$\alpha = 5$	
	w&s	$\mathcal{R}^q$	$\mathcal{R}^{q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q+}$
10c-5w-1	12.8	8.9	14.1	7.3	13.1	4.4	9.2
10c-5w-2	10.8	-4.8	0.2	-4.8	4.9	-8.8	0.5
10c-5w-3	8.0	-46.9	-29.9	-43.1	-30.6	-43.1	-32.9
10c-5w-4	10.5	-10.9	-8.7	-10.9	-4.9	-10.9	-2.2
10c-5w-5	8.4	-17.9	-6.1	-18.9	-2.8	-19.5	1.1
#eval		3^*10^4	3^*10^3	7^*10^4	5^*10^3	2^*10^5	2^*10^4
10c+w-1	12.8	39.1	30.3	38.3	36.7	34.9	34.1
10c+w-2	10.8	32.3	18.8	32.1	25.2	32.3	25.8
10c+w-3	8.0	26.1	18.8	27.6	20.3	23.1	23.9
10c+w-4	10.5	22.6	12.3	23.3	16.4	18.8	16.5
10c+w-5	8.4	31.6	15.8	32.7	21.1	29.3	28.5
#eval		3^*10^4	3^*10^3	7^*10^4	6^*10^3	2^*10^5	2^*10^4

Table 8.1: Results on small instances ($n = 10$, $K = 2$, $Q = \infty$) when $\alpha \in \{1, 2, 5\}$ and $\beta = 60$. For each instance, we give the average cost over 10^6 sampled scenarios using the *wait-and-serve* policy (*w&s*) and the gain of the best solution found by the exact approach within a time limit of 30 minutes and PFS within a time limit of 5 minutes (average on 10 runs). Results marked with a star (*) have been proved optimal. #eval gives the average number of expectation computations for each run: solutions enumerated (Exact) or LS iterations (PFS). Grey cells show best results.

Instance:	10c-5w-1	10c-5w-2	10c-5w-3	10c-5w-4	10c-5w-5
Travel time in C:	19.6′	16.8′	12.5′	18.0′	13.0′
Travel time C to W:	23.7′	19.9′	19.5′	20.9′	18.2′
Time window:	11.6′	12.7′	12.3′	13.2′	12.6′

Table 8.2: Statistics on instances 10c-5w-*i*: the first (resp. second) line gives the average travel time between customer vertices (resp. between a customer and waiting vertices); the last line gives the average duration of a time window.

vehicles. We consider three different values for α. When $\alpha = 1$ (resp. $\alpha = 2$, $\alpha = 5$), the time horizon is $h = 480$ (resp. $h_\alpha = 240$, $h_\alpha = 96$) and each time unit corresponds to one minute (resp. two and five minutes). In all cases, the domain reduction factor β is set to 60: waiting times are restricted to multiples of 60 minutes.

Unlike the recourse strategies, which must to deal with a limited set of predefined waiting locations, the w&s policy makes direct use of the customer vertices. Therefore, the relative gain of using an optimized SS-VRPTW-CR first-stage solution is highly dependent on the locations of the waiting vertices. Gains are always greater for *10c+w-i* instances, where any customer vertex can be used as a waiting vertex: for these instances, gains with the best-performing strategy are always greater than 23%, whereas for *10c-5w-i* instances, the largest gain is 16%, and is negative in some cases.

The relative gain of using the SS-VRPTW-CR first-stage solutions depends on whether the average travel times between customer vertices exceeds the duration of their time windows.

The results obtained on instance *10c-5w-3* are quite interesting: gains are always negative; *i.e.*, waiting strategies always lead to higher expected numbers of rejected requests than the w&s policy. By looking further into the average travel times in each instance, in Table 8.2, we find that the average travel time between customer vertices in instance *10c-5w-3* is rather small (12.5), and very close to the average duration of time windows (12.3). In this case, anticipation is of less importance and the w&s policy appears to perform better. Furthermore, average travel time between waiting and customer vertices (19.5) is much larger than the average travel time between customer vertices.

We note that the exact enumerative method runs out of time under $\mathcal{R}^{q+}$ for all instances, when $\alpha = 1$. Increasing α to 2 speeds up the solution process and makes it possible to prove optimality on all *10c-5w-i* instances except instance 5. Setting $\alpha = 5$ allows to find all optimal solutions. However, optimizing with coarser scales may degrade the solution quality. This is particularly true for *10c-5w-i* instances which are easier, in terms of solution space, than *10c+w-x* instances as they have half the number of waiting locations: for *10c-5w-i* instances, gains are often decreased when α is increased because, whatever the scale is, the search finds optimal or near-optimal solutions.

For PFS, gains with recourse strategy $\mathcal{R}^{q+}$ are always greater than gains with recourse strategy $\mathcal{R}^{q}$ on *10c-5w-i* instances. However, we

observe the opposite on *10c+w-i* instances. This comes from the fact that expected costs are much more expensive to compute under $\mathcal{R}^{q+}$ than under $\mathcal{R}^{q}$. Table 8.1 displays the average number of times the objective function $\mathcal{Q}^{\mathcal{R}}(x,\tau)$ is evaluated (#eval), that is the number of solutions considered by either the local search or the exact method, in which case it corresponds to the size of the solution space (when enumeration is complete and under assumptions *(a)* and *(b)* discussed in Section 8.3.1.1). We note that the number of LS iterations is ± 10 times smaller when using $\mathcal{R}^{q+}$ compared to $\mathcal{R}^{q}$. As *10c-5w-i* instances are easier than *10c+w-i* instances, around 10^4 iterations is enough to allow the LS optimizer of PFS to find near-optimal solutions. In this case, gains obtained with $\mathcal{R}^{q+}$ are much larger than those obtained with $\mathcal{R}^{q}$. However, on *10c+w-i* instances, 10^4 iterations are not enough to find near-optimal solutions. For these instances, better results are obtained with $\mathcal{R}^{q}$.

When optimality has been proven by Exact, we note that PFS often finds solutions with the same gain. With $\alpha \in \{2,5\}$, PFS may even find better solutions: this is due to the fact that optimality is only proven for the simplified problem $P_{\alpha,\beta}$, whereas the final gain is computed after scaling back to the original horizon at scale 1. When optimality has not been proven, PFS often finds better solutions (with larger gains).

8.3.1.3 *Combining recourse strategies: $\mathcal{R}^{q/q+}$*

Results obtained from Table 8.1 show that although it leads to larger gains, the computation of expected costs is much more expensive under recourse strategy $\mathcal{R}^{q+}$ than under $\mathcal{R}^{q}$, which eventually penalizes the optimization process as it performs fewer iterations within the same time limit (for both Exact and PFS).

We now introduce a pseudo-strategy that we call $\mathcal{R}^{q/q+}$, which combines $\mathcal{R}^{q}$ and $\mathcal{R}^{q+}$. For both Enum and PFS, strategy $\mathcal{R}^{q/q+}$ refers to the process that uses $\mathcal{R}^{q}$ as the evaluation function during all the optimization process. When stopping at a final solution, we reevaluate it using $\mathcal{R}^{q+}$. Table 8.3 reports the gains obtained by applying $\mathcal{R}^{q/q+}$ on instances *10c-5w-i* and *10c+w-i*. By using $\mathcal{R}^{q/q+}$, we actually use $\mathcal{R}^{q}$ to guide the LS optimization, which permits the algorithm to consider a significantly bigger part of the solution space. For both Enum and PFS, $\mathcal{R}^{q/q+}$ always leads to better results than $\mathcal{R}^{q}$. From now on, we will only consider strategies $\mathcal{R}^{q/q+}$ and $\mathcal{R}^{q+}$ in the next experiments.

Expected costs computed under $\mathcal{R}^{q+}$ are always significantly better than under $\mathcal{R}^{q}$, showing that the recourse strategy performs more clever operations.

8.3.1.4 *Impact of the domain reduction factor β*

Table 8.4 considers instances involving 20 customer vertices and either 10 separated waiting locations (*20c-10w-i*) or one waiting location at each customer vertex (*20c+w-i*). It compares results obtained by PFS for two different computation time limits, with $\beta \in \{10,30,60\}$. When

		Exact (% gain after 30 minutes)					
		$\alpha = 1$		$\alpha = 2$		$\alpha = 5$	
	w&s	$\mathcal{R}^q$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q/q+}$
10c-5w-1	12.8	8.9	14.0	7.3	12.8	4.4	9.5
10c-5w-2	10.8	-4.8	7.4	-4.8	7.4	-8.8	0.5
10c-5w-3	8.0	-46.9	-26.5	-46.9	-26.5	-55.9	-37.3
10c-5w-4	10.5	-10.9	0.9	-10.9	0.9	-10.9	0.9
10c-5w-5	8.4	-17.9	2.5	-17.9	2.5	-20.5	0.5
10c+w-1	12.8	35.3	37.7	35.3	37.7	32.7	34.0
10c+w-2	10.8	28.1	33.2	30.1	34.6	30.1	34.6
10c+w-3	8.0	14.4	24.4	18.8	29.9	13.3	21.9
10c+w-4	10.5	7.8	13.5	12.4	20.0	7.8	13.5
10c+w-5	8.4	3.5	10.6	8.4	13.3	23.7	31.0

		PFS (% gain after 5 minutes)					
		$\alpha = 1$		$\alpha = 2$		$\alpha = 5$	
	w&s	$\mathcal{R}^q$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^q$	$\mathcal{R}^{q/q+}$
10c-5w-1	12.8	8.9	14.0	7.3	12.8	4.4	9.5
10c-5w-2	10.8	-4.8	7.1	-4.8	7.4	-8.8	0.5
10c-5w-3	8.0	-46.9	-26.5	-43.1	-22.5	-43.1	-22.5
10c-5w-4	10.5	-10.9	0.9	-10.9	0.9	-10.9	0.9
10c-5w-5	8.4	-17.9	2.5	-18.9	1.8	-19.5	1.1
10c+w-1	12.8	39.1	40.9	38.3	39.5	34.9	36.2
10c+w-2	10.8	32.3	35.6	32.1	35.5	32.3	34.4
10c+w-3	8.0	26.1	35.0	27.6	35.6	23.1	30.3
10c+w-4	10.5	22.6	29.2	23.3	29.3	18.8	24.1
10c+w-5	8.4	31.6	39.8	32.7	40.7	29.3	35.0

Table 8.3: Comparison of $\mathcal{R}^q$ with the hybrid strategy $\mathcal{R}^{q/q+}$ (that uses strategy $\mathcal{R}^q$ as evaluation function during the optimization process, and evaluates the final solution with strategy $\mathcal{R}^{q+}$) on the small instances used in Table 8.1, with $\beta = 60$.

		Exact (% gain after 30 minutes)					
		$\beta = 60$		$\beta = 30$		$\beta = 10$	
	w&s	$\mathcal{R}^{q/q+}$	$\mathcal{R}^{q+}$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^{q+}$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^{q+}$
20c-10w-1	22.6	9.3	-12.1	9.7	-11.4	12.6	-16.0
20c-10w-2	19.8	-11.8	-27.9	-5.0	-29.1	-3.7	-31.4
20c-10w-3	21.1	-0.5	-16.7	5.6	-15.9	6.4	-21.3
20c-10w-4	25.3	4.6	-4.3	5.2	-6.9	5.4	-9.2
20c-10w-5	20.9	-10.7	-25.9	-1.0	-24.6	0.2	-23.8
20-c+w-1	22.6	15.4	2.8	17.2	2.2	17.4	0.2
20-c+w-2	19.8	7.6	-10.0	5.6	-16.8	6.9	-14.3
20-c+w-3	21.1	2.8	-11.1	3.9	-14.1	3.1	-13.7
20-c+w-4	25.3	14.3	5.2	15.3	2.6	14.0	3.0
20-c+w-5	20.9	13.6	-2.6	14.0	-11.2	16.7	-7.7

		PFS (% gain after 5 minutes)					
		$\beta = 60$		$\beta = 30$		$\beta = 10$	
	w&s	$\mathcal{R}^{q/q+}$	$\mathcal{R}^{q+}$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^{q+}$	$\mathcal{R}^{q/q+}$	$\mathcal{R}^{q+}$
20c-10w-1	22.6	10.8	5.5	12.0	5.6	15.2	6.4
20c-10w-2	19.8	-5.7	-12.2	-3.8	-10.2	-1.9	-8.7
20c-10w-3	21.1	1.1	-2.8	7.7	-0.9	8.1	0.2
20c-10w-4	25.3	5.7	4.3	5.5	3.8	5.1	4.7
20c-10w-5	20.9	-9.1	-14.0	-0.0	-7.0	0.8	-6.4
20-c+w-1	22.6	17.9	13.5	19.4	11.4	20.2	13.0
20-c+w-2	19.8	12.2	2.7	10.7	2.1	12.3	3.5
20-c+w-3	21.1	4.8	1.2	6.4	0.4	7.8	0.3
20-c+w-4	25.3	15.9	12.4	18.6	13.5	19.6	14.2
20-c+w-5	20.9	15.7	9.8	19.0	11.0	18.5	12.0

Table 8.4: Relative gains 5 and 30 minutes, using three domain reduction factors ($\beta \in \{10, 30, 60\}$), with $K = 2$ uncapacitated vehicles and a scale factor $\alpha = 2$. Instances involve $n = 20$ customer locations and either 10 or 20 available waiting locations.

$\beta = 10$ (resp. $\beta = 30$ and $\beta = 60$), domains of waiting time variables contain 48 (resp. 16 and 8) values, corresponding to multiples of 10 (resp. 30 and 60) minutes. In all cases, the scale factor α is set to 2.

When considering the recourse strategy $\mathcal{R}^{q+}$ with a five-minute computation time limit, we observe that better results are obtained with $\beta = 60$, as domains are much smaller. When the computation time is increased to 30 minutes, or when considering strategy $\mathcal{R}^{q/q+}$, which is cheaper to compute, then better results are obtained with $\beta = 10$, as domains contain finer-grained values.

We observe that $\mathcal{R}^{q/q+}$ always provides better results than pure $\mathcal{R}^{q+}$, whatever the waiting time multiple β used. Except when switching to significantly greater computational times, $\mathcal{R}^{q/q+}$ seems more adequate as it combines the limited computational cost incurred by $\mathcal{R}^{q}$ with the nicer expected performances of the cleverer strategy $\mathcal{R}^{q+}$.

8.3.2 *Experiments on large instances*

We now consider instances with $n = 50$ customer vertices. Instances *50c-30w-i* and *50c-50w-i* have $m = 30$ and $m = 50$ separated waiting locations, respectively. Instances *50c+w-i* have $m = 50$ waiting vertices which correspond to the customer vertices. Each class is composed of 15 instances such that, for each seed $i \in [1, 15]$, the three instances classes *50c-30w-i*, *50c-50w-i*, and *50c+w-i* contain the same set of 50 customer vertices and thus only differ in terms of the number and/or positions of waiting vertices. For each instance, the vehicle's capacity is set to $Q = 20$, and we consider three different numbers of vehicles $K \in \{5, 10, 20\}$. In total, we thus have $45 \times 3 = 135$ different configurations.

We first compare and discuss the behaviors of different instantiations of PFS. Then, based on the PFS variant that appears to perform best, further experiments (Section 8.3.2.3) measure the contribution of a two-stage stochastic model, through the use of a SS-VRPTW-CR formulation and our recourse strategies.

8.3.2.1 *Instantiations of PFS*

All runs of PFS are limited to $T = 10800$ seconds (three hours). We compare seven instantiations of PFS, which have different update and computation time policies $\mathcal{U}$ and $\mathcal{T}$, while all other parameters are set as described in Section 6.2.1. Strategy $\mathcal{R}^{q/q+}$ is used for all experiments. The different instantiations are:

- *PFS-α*β10*: the scale factor α is progressively decreased from 5 to 2 and 1 while the domain reduction factor β remains fixed to 10. More precisely, $\alpha_0 = 5$, $\alpha_{\min} = 1$, and $\beta_0 = \beta_{\min} = 10$. The update policy $\mathcal{U}$ successively returns $\alpha_1 = 2$ and $\alpha_2 = 1$, while $\beta_1 = \beta_2 = 10$. The computation time policy $\mathcal{T}$ always returns

3600 seconds, so that the three LS optimizations have the same CPU time limit of one hour.

- *PFS-α1β**: α remains fixed to 1 while β is progressively decreased from 60 to 30 and 10. More precisely, $\alpha_0 = \alpha_{\min} = 1$, $\beta_0 = 60$, and $\beta_{\min} = 10$. The update policy $\mathcal{U}$ successively returns $\beta_1 = 30$ and $\beta_2 = 10$, while $\alpha_1 = \alpha_2 = 1$. The computation time policy $\mathcal{T}$ always returns 3600 seconds.

- *PFS-α*β**: both α and β are progressively decreased. We set $\alpha_0 = 5$, $\alpha_{\min} = 1$, $\beta_0 = 60$, and $\beta_{\min} = 10$. The update policy $\mathcal{U}$ returns the following couples of values for (α_i, β_i): (5, 60), (2, 60), (1, 60), (5, 30), (2, 30), (1, 30), (5, 10), (2, 10), (1, 10). The computation time policy $\mathcal{T}$ always returns 1200 seconds. The PFS optimization process is hence composed of nine LS optimizations of 20 minutes each.

- *PFS-αaβb* which performs only a single LS optimization step with $T = 10800$ and $\alpha_0 = \alpha_{\min} = a$ and $\beta_0 = \beta_{\min} = b$, as experimented in Section 7. We consider two different values for α, *i.e.*, $a \in \{1, 2\}$, and two different values for β, *i.e.*, $b \in \{10, 60\}$, thus obtaining four different instantiations.

8.3.2.2 *Comparison of the different PFS instantiations*

The performances of the seven PFS instantiations and the baseline w&s approach are compared in Figure 8.1 by using *performance profiles*. Performance profiles (Dolan and Moré, 2002) provide, for each considered approach, a cumulative distribution of its performance compared to other approaches. For a given method A, a point (x, y) on A's curve means that in $(100 \cdot y)\%$ of the instances, A performed at most x times worse than the best method on each instance taken separately. A method A is strictly better than another method B if A's curve always stays above B's curve.

According to Figure 8.1 (left), algorithms *PFS-α*β10* and *PFS-α*β** show the best performances when tested on the 15 instances of class *50c+w-i* with $K = 20$ vehicles. More experiments are conducted and reported in Figure 8.1 (right) in order to distinguish between the algorithms *PFS-α*β10*, *PFS-α1β** and *PFS-α*β** on all 135 instances. In comparison to the other approaches, algorithms *PFS-α*β10* and *PFS-α*β** clearly obtain the best performances on average over the 135 configurations.

Figure 8.2 illustrates, on a single instance (*50c-50w-1* with $K = 10$ vehicles), the evolution through time of the gain of the expected cost of the current solution s, with respect to the average cost of the w&s policy, during a single run of *PFS-α1β10*, *PFS-α*β10*, *PFS-α1β**, and *PFS-α*β**. For each incumbent solution s, the left part of Figure 8.2 plots the gain of s under $\mathcal{R}^q$ at its current scale α. It corresponds to

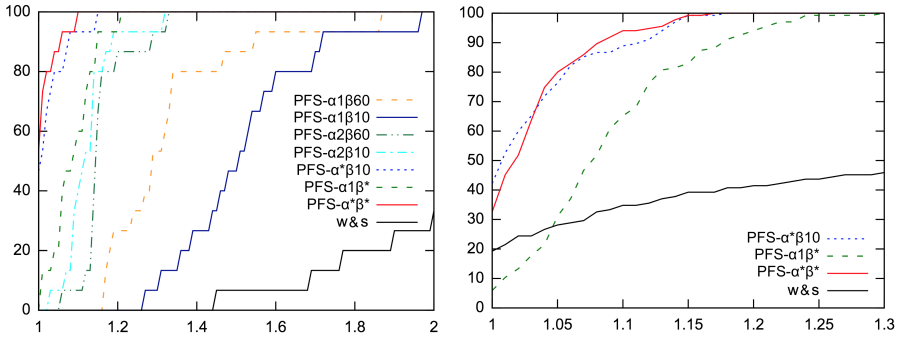


Figure 8.1: Performance profiles. Left: comparison of the seven PFS instantiations and the *w&s* policy on the 15 instances of class *50c+w-i*, using $K = 20$ vehicles. Right: comparison of PFS instantiations *PFS-α*β10*, *PFS-α1β** and *PFS-α*β** on the 3 classes (*50c-30w-i*, *50c-50w-i*, *50c+w-i*), with $K \in \{5, 10, 20\}$ vehicles (135 instances).

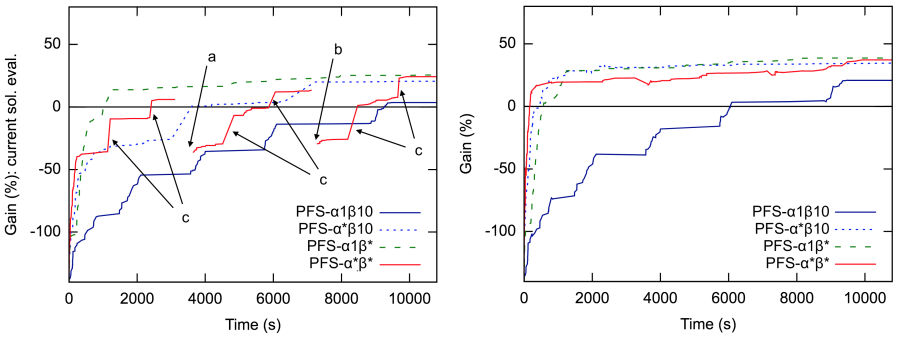


Figure 8.2: Evolution through time of the gain of the expected cost of the current solution with respect to the average cost of the w&s policy, during a single execution of four PFS instantiations for instance *50c-50w-1* (with $K = 10$ vehicles). Left: gain evaluated under $\mathcal{R}^q$ at current scale α. Right: gain evaluated under $\mathcal{R}^{q+}$ at scale $\alpha = 1$.

the quality of s as evaluated by the LS algorithm. The right part plots the corresponding gain under $\mathcal{R}^{q+}$ at scale $\alpha = 1$. In the left part, we clearly recognize the nine different optimization phases of *PFS-α*β**. A drop in the expected cost happens whenever the current solution s is converted to a higher scale factor. This happens twice during the run: from $\alpha_2 = 1$ to $\alpha_3 = 5$ (point *a*) and from $\alpha_5 = 1$ to $\alpha_6 = 5$ (point *b*). In both cases, the resulting solution becomes infeasible and the algorithm needs some time to restore feasibility. A sudden leap happens when converting to a lower scale. This happens six times (points *c*): from $\alpha_i = 5$ to $\alpha_{i+1} = 2$ and from $\alpha_{i+1} = 2$ to $\alpha_{i+2} = 1$, with $i \in \{0, 3, 6\}$. This is a direct consequence of the fact that rounding operations are always performed in a pessimistic way. Whereas the quality of s under $\mathcal{R}^q$ at scale α appears to be worse than that of *PFS-α1β10* (*e.g.*, at point *b*), the true gain of s (evaluated under $\mathcal{R}^{q+}$, $\alpha = 1$) remains always better with *PFS-α*β**.

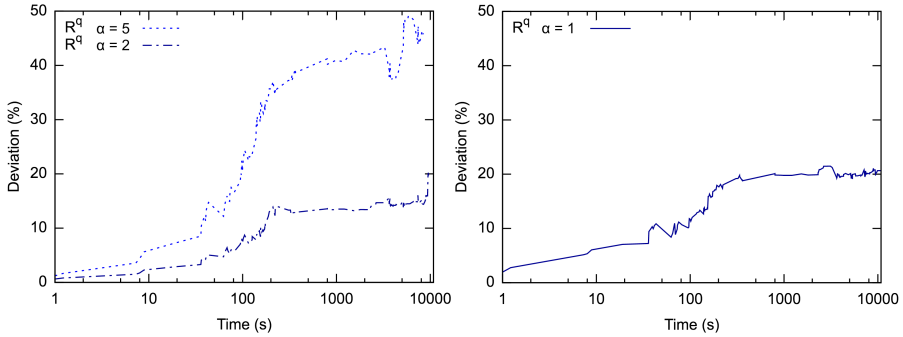


Figure 8.3: Scale approximation quality and impact of recourse strategies. For each solution s encountered while running *PFS-α*β**, on instance *50c-50w-1* as displayed in Figure 8.2, left curves show the evolution of the gap (in %) between costs computed with $\alpha \in \{2,5\}$ and those computed with $\alpha = 1$. On right, the gap between $\mathcal{R}^q$ and $\mathcal{R}^{q+}$, both with $\alpha = 1$.

Finally, Figure 8.3 compares the expected costs when varying either the scale α (left) or the recourse strategy (right), using the same sequences of solutions than those used for Figure 8.2. On left, the evolution of the deviation (%) between costs computed with $\alpha = 1$, and $\alpha \in \{2,5\}$, under strategy $\mathcal{R}^q$. On right, the deviation between costs computed with $\mathcal{R}^q$ and $\mathcal{R}^{q+}$, with $\alpha = 1$ in both cases. Scale $\alpha = 2$ (left, long dashed) always provides a better approximations, closer to the one as computed under $\alpha = 1$, than scale $\alpha = 5$ (left, dashed). We also notice a significant increase in the gaps as the algorithm finds better solutions: under 100 seconds, costs computed at scale $\alpha = 2$ (resp. $\alpha = 5$) remain at maximum 10% (resp. 20%) from what would be computed under $\alpha = 1$, and tend to stabilize at around 20% (resp. 45%) in the long term. Similar observations can be made (Figure 8.3, right) regarding the gap between costs computed with $\mathcal{R}^q$ at $\alpha = 1$ and those computed with $\mathcal{R}^{q+}$, $\alpha = 1$. Similarly, the cost difference subsequent to the recourse strategy tends to increase progressively with the quality of the solutions. This could be explained by the time discrepancies generated by rounding operations when a solution is scaled. Better solutions having complex, tighter schedules are then less robust to such time approximations, and more sensible to the discrepancy effects which propagate and impact on the customer time windows.

8.3.2.3 *Results on large instances*

We now analyze how our SS-VRPTW-CR model behaves compared to the w&s policy, when varying both vehicle fleet size and the urgency of requests. We consider algorithm *PFS-α*β** only.

INFLUENCE OF THE NUMBER OF VEHICLES. Table 8.5 shows how the performance of the SS-VRPTW-CR model relative to the w&s

	*PFS-α*β**						
	w&s	50c30w		50c50w		50c+w	
	#rejects	#rejects	%gain	#rejects	%gain	#rejects	%gain
$K = 5$	39.3	40.1	-2.5	40.7	-4.3	39.0	0.4
$K = 10$	33.9	27.1	18.4	25.5	22.9	23.6	29.2
$K = 20$	33.7	20.3	38.9	17.9	45.8	16.0	52.2

Table 8.5: Average number of rejected requests on instances *50c-30w-i*, *50c-50w-i*, *50c+w-i*, *w.r.t.* the number of vehicles.

	*PFS-α*β**						
	w&s	50c30w		50c50w		50c+w	
	#rejects	#rejects	%gain	#rejects	%gain	#rejects	%gain
$K = 5$	21.0	28.1	-35.1	27.3	-31.4	26.3	-26.3
$K = 10$	13.8	14.8	-11.7	13.8	-3.5	12.9	3.4
$K = 20$	13.6	10.4	21.2	9.0	31.7	8.8	33.7

Table 8.6: Average number of rejected requests, all time window durations being doubled.

policy varies with the waiting locations and the number of vehicles. For 5, 10, and 20 vehicles, the average over each of the instance classes (15 instances per class) is reported.

The more vehicles are involved, the more performant are the SS-VRPTW-CR first-stage solutions compared to a non-anticipative policy.

It shows us that the more vehicles are involved, the more important clever anticipative decisions are, and therefore the more beneficial a SS-VRPTW-CR solution is compared to the w&s policy. It is likely that, as conjectured in Saint-Guillain et al., 2017, a higher number of vehicles leads to a less uniform objective function, most probably with the steepest local optima. Because it requires much more anticipation than when there are only five vehicles, using the SS-VRPTW-CR model instead of the w&s policy is found to be particularly beneficial provided that there are at least 10 or 20 vehicles. With 20 vehicles, our model decreases the average number of rejected requests by 52.2% when vehicles are allowed to wait at customer vertices (*i.e.* for the class of instances *50c+w-i*).

On the other hand, we also observe that due to the lack of anticipative actions, the w&s policy globally fails at tacking the advantage of a larger number of vehicles. Indeed, allowing 20 vehicles does not significantly improve the performances of the baseline policy compared to only 10 vehicles.

INFLUENCE OF THE TIME WINDOWS. We now consider less urgent requests, by conducting the same experiments as in Section 8.3.2.3 while modifying the time windows only. Table 8.6 shows the average gain of using an SS-VRPTW-CR model when the service quality is reduced by *multiplying all the original time window durations by two*. The results show that for $K = 5$ vehicles, the w&s policy always performs better. With $K = 20$ vehicles, however, the average relative

	*PFS-α*β**						
	w&s	50c30w		50c50w		50c+w	
	#rejects	#rejects	%gain	#rejects	%gain	#rejects	%gain
$K = 5$	14.2	22.5	-61.1	22.0	-57.8	21.1	-50.7
$K = 10$	7.4	10.9	-60.0	9.7	-41.7	8.8	-26.1
$K = 20$	7.3	8.1	-19.8	7.0	-4.1	6.0	14.0

Table 8.7: Average number of rejected requests, all time window durations being tripled.

gain achieved by using the SS-VRPTW-CR model remains significant: there are 33.7% fewer rejected requests on average for the class of instances *50c+w-i*.

Table 8.7 illustrates how the average gain is impacted when *time windows are multiplied by three*. Given 20 vehicles, the SS-VRPTW-CR model still improves the w&s policy by 14% when vehicles are allowed to wait directly at customer vertices. Together with Table 8.5, Tables 8.6 and 8.7 show that the SS-VRPTW-CR model is more beneficial when the number of vehicles is high and the time windows are small, that is, in instances that are particularly hard in terms of quality of service and thus require much more anticipation.

*The more **urgent** are the requests, the more interesting is the SS-VRPTW-CR model.*

POSITIONS OF THE WAITING LOCATIONS. From all the experiments conducted on our benchmark, it immediately appears that, no matter the operational context (number of customer vertices, vehicles) or the approximations that are used (scaling factor, waiting time multiples), *allowing the vehicles to wait directly at customer vertices always leads to better results than using separated waiting vertices*. Unless the set of possible waiting locations is restricted, *e.g.*, big vehicles cannot park anywhere in the city, placing waiting vertices in such a way that they coincide with customer vertices appears to be the best choice.

8.4 CONCLUSIONS AND RESEARCH DIRECTIONS

This chapter introduced PFS, a meta-heuristic particularly suitable for our SS-VRPTW-CR problem when coupled with the LS algorithm. More generally, PFS is applicable to any problem in which: *a)* the objective function is particularly complex to compute but depends on the accuracy of the data and *b)* the size of the solution space can be controlled by varying the granularity of the operational decisions. We show that PFS allows to efficiently tackle larger problems for which an exact approach is not possible.

Experiments show that SS-VRPTW-CR recourse strategies provide significant benefits compared to a basic, non-anticipative but yet realistic policy. Results for a variety of large instances show that the benefit of using the SS-VRPTW-CR increases with the number of vehicles involved and the urgency of the requests. Finally, all our experiments

indicate that allowing the vehicles to wait directly at potential customer vertices, when applicable, leads to better expected results than using separated relocation vertices.

Future work and research avenues

The research directions described here apply locally, that is, on the new meta-heuristic we just introduced. More general conclusions and future work in the context of online stochastic VRPs are discussed in the *Conclusion and Perspectives* part, concluding the thesis.

ON SOLUTION METHODS. An adaptive version of PFS, therefore improving the algorithm by making dynamic the decision about changing the scale factor α or the domain reduction factor β, could be designed. Exact optimal methods should also be investigated. However, the black box nature of the evaluation function $\mathcal{Q}^{\mathcal{R}}$ makes classical (stochastic) integer programming approaches (*e.g.* branch-and-cut, L-shaped method, *etc.*) unsuitable for the SS-VRPTW-CR unless efficient valid inequalities that are active at fractional solutions can be devised (such as those proposed by Hjorring and Holt, 1999, for the SS-VRP-D). Amongst other possible candidates for solving this problem, we could consider set-partitioning methods such as column generation, which are becoming commonly used for stochastic VRPs. Approximate Dynamic Programming (ADP, Powell, 2009) is also widely used to solve routing problems in presence of uncertainty. Combined with scaling techniques, ADP is likely to provide interesting results.

ON SCALING TECHNIQUES. We have shown through experiments that the computational complexity of the objective function is an issue that can be successfully addressed by scaling down problem instances. However, the scale is only performed in terms of temporal data, decreasing the accuracy of the time horizon. It may also be valuable to consider a reduced, clustered set of potential requests, which would also allow us to significantly reduce computational effort when evaluating a first-stage solution.

9

APPLICATION: A PRIORI OPTIMIZATION FOR POLICE PATROL MANAGEMENT IN BRUSSELS

The Static and Stochastic Vehicle Routing Problem with random Requests (*SS-VRP-R*) describes realistic operational contexts, in which a fleet of vehicles has to deal with customer requests appearing in a dynamic fashion. Based on a probabilistic knowledge about requests appearance, the SS-VRP-R seeks for *a priori* sequences of vehicle relocations, optimising the expected responsiveness to the requests.

In this chapter, we show how the existing recourse strategies, proposed for the SS-VRP with both random Customers and Reveal Times (*SS-VRPTW-CR*), can be adapted to meet the objective function of the SS-VRP-R. The resulting model is then applied to the real case study of a police units management in Brussels, Belgium. In this context, the expected average intervention delay is minimised.

In order to cope with the reality of the urban context, travel time dependency is introduced in our computational models. Experiments show the contribution and the adaptability of the recourse strategies to a real life, complex, operational context. Provided an adequate solution method, simulation-based results show the high quality of the a priori solutions designed.

9.1 PROBLEM DEFINITION

The SS-VRP-R is very similar to the SS-VRPTW-CR, except that the online customer requests are always accepted, without any time window. Henceforth, the objective function is no longer to minimize the expected number of rejected requests. Instead, the SS-VRP-R maximizes the expected quality of service, by minimizing the expected average time needed for a vehicle to meet an appeared online request.

*The **SS-VRP-R** aims at minimizing the **expected average** service **delay** of online requests.*

9.1.1 *Objective function*

Under a given recourse strategy $\mathcal{R}$, let

$$\text{delay}^{\mathcal{R}}(r, \xi(\omega))$$

be the deterministic function that returns the delay between reveal time Γ_r (of an online request r) and the moment at which a vehicle reaches r's location, when following a specific scenario $\xi(\omega)$. We are

interested in the average time needed to meet all the requests that appeared in $\xi(\omega)$, our second-stage value function:

$$\mathcal{Q}^{\mathcal{R}}(x, \tau, \xi(\omega)) = \frac{1}{|R(\xi(\omega))|} \sum_{r \in R(\xi(\omega))} \text{delay}^{\mathcal{R}}(r, \xi(\omega)), \tag{9.1}$$

where $R(\xi(\omega)) \subseteq R$ is the set of requests that revealed to appear in scenario $\xi(\omega)$. Hence, we define the expected second-stage value function as the expectation of the average time needed to meet any request that reveals to appear:

$$\begin{aligned} \mathscr{Q}^{\mathcal{R}}(x, \tau) &= \mathrm{E}\mathcal{Q}^{\mathcal{R}}(x, \tau, \boldsymbol{\xi}) \\ &= \frac{1}{|R|} \sum_{r \in R} \mathrm{E}\big[\ \text{delay}^{\mathcal{R}}(r, \boldsymbol{\xi}) \mid r \text{ appears}\ \big], \end{aligned} \tag{9.2}$$

where $\text{delay}^{\mathcal{R}}(r, \boldsymbol{\xi})$ is a random function of the a priori solution (x, τ), the recourse strategy $\mathcal{R}$ and the random variables describing the potential requests R. Therefore, $\mathrm{P}\{\text{delay}^{\mathcal{R}}(r, \boldsymbol{\xi}) = \Delta \mid r \text{ appears}\}$ is the probability that r is met after a delay of Δ time units, conditionally that r appears.

Henceforth, the SS-VRP-R problem can be formulated as the following two-stage stochastic program:

$$\text{(SS-VRP-R)} \quad \underset{x,\tau}{\text{Minimize}} \ \ \mathscr{Q}^{\mathcal{R}}(x, \tau) \tag{9.3}$$

$$\text{s.t.} \quad (x, \tau) \text{ is a first-stage solution,} \tag{9.4}$$

where the expected second-stage value function $\mathscr{Q}^{\mathcal{R}}(x, \tau)$ is defined in (9.2).

9.1.2 *Recourse Strategy and Expected Delays: constant travel times*

In its current form, and provided an implementation of strategy $\mathcal{R}$ allowing the computation of $\text{delay}^{\mathcal{R}}(r, \xi(\omega))$, equation (9.2) can be directly approached by enumerating all the possible scenarios:

$$\mathscr{Q}^{\mathcal{R}}(x, \tau) = \sum_{\xi_i \in \mathcal{S}} \frac{p(\xi_i)}{|R(\xi_i)|} \sum_{r \in R(\xi_i)} \text{delay}^{\mathcal{R}}(r, \xi_i). \tag{9.5}$$

Naturally, due to the size of $\mathcal{S}$ such approach is definitely not feasible in practice. Sampling methods, such as Sample Average Approximation (SAA, Ahmed and Shapiro, 2002), allow to approximate $\mathscr{Q}^{\mathcal{R}}$ on a limited subset of $\mathcal{S}$, but they do not provide any guarantee.

In what follows, we show how to efficiently compute $\mathscr{Q}^{\mathcal{R}}$ by directly exploiting the closed form expressions provided in Section 5.2, for the SS-VRPTW-CR. In fact, because the SS-VRPTW-CR is closely related to the SS-VRP-R, the recourse strategy $\mathcal{R}^{\infty}$ proposed for the SS-VRPTW-CR can be naturally exploited here for the SS-VRP-R, hence leading to a new variant, referred to as $\mathcal{R}'$ in what follows.

The two problems are quite similar, however there are a couple of differences we must consider while exploiting a recourse strategy initially designed for the SS-VRPTW-CR. Customer requests do not have time window in the SS-VRP-R; whenever a request appears, it must be served as soon as possible. More important, all the requests must be serviced, meaning that unlike the SS-VRPTW-CR none can be rejected. We then make the assumption that there exists a finite time window duration for every request, so that a solution can be found in which a request is never rejected, even under the SS-VRPTW-CR recourse strategy $\mathcal{R}^{\infty}$.

Given a first stage solution (x, τ), and following the computation of $\mathcal{R}^{\infty}$, function $g_1(r, t)$ returns the probability that a vehicle leaves its current location at time t to meet an accepted request r. Function $g_1(r, t)$ is specifically defined for $\mathcal{R}^{\infty}$ in Saint-Guillain et al., 2017. It can also be naturally derived from the definition of $g_1(r, t, q)$, provided for $\mathcal{R}^{q}$ in Section 5.2 of the current thesis.

A vehicle must depart from location v at time unit $\Gamma_r + \Delta - d_{v,r}$ in order to meet a request r with a delay Δ. If we assume that a request r that appears gets immediately accepted, then:

$$\mathrm{P}\{\mathrm{delay}^{\mathcal{R}'}(r, \boldsymbol{\xi}) = \Delta \mid r \text{ appears}\} \;=\; \frac{g_1(r,\ \Gamma_r + \Delta - t_{v,r})}{p_r} \tag{9.6}$$

if and only if (x, τ) is such that r is always accepted under recourse strategy $\mathcal{R}'$, that is, if and only if:

$$p_r = \mathrm{P}\{r \in A^h\} = \sum_{t=d^{\min}_{r,\mathrm{w}(r)}}^{d^{\min}_{r,\mathrm{w}(r)}} g_1(r, t).$$

For such solutions and with strategy $\mathcal{R}'$, equation (9.2) then becomes:

$$\mathcal{Q}^{\mathcal{R}'}(x, \tau) = \frac{1}{|R|} \sum_{r \in R} \sum_{\Delta \in H} \Delta \cdot \frac{g_1(r,\ \Gamma_r + \Delta - t_{\mathrm{w}(r),r})}{p_r}. \tag{9.7}$$

*We directly exploit **SS-VRPTW-CR closed-form expressions** in order to compute the **SS-VRP-R objective function**.*

Even with sufficiently large time windows, depending on the first stage solution (x, τ) an appeared request may still be rejected. This is due to the fact that, while following the sequences of waiting vertices planned in (x, τ), strategy $\mathcal{R}'$ enforces the a priori arrival times of the vehicles at these waiting vertices. A request is then rejected if it prevents the associated vehicle from respecting its schedule, fixed a priori in (x, τ). Henceforth, while exploiting strategy $\mathcal{R}'$, we consider a first stage solution (x, τ) as being SS-VRP-R feasible if and only if $\forall r : \mathrm{P}\{r \in A^h\} = p_r$. We will see in Section 9.3.2, *Solution Method*, how we can deal with this issue.

9.1.3 *Recourse Strategy and Expected Delays: time-dependent travel times*

As further discussed in the next section, variable travel times, and more specifically time dependent travel times, could be an important

aspect to take into account while dealing with urban vehicle routing. For now, we describe how our equations can be adapted accordingly.

*Our SS-VRPTW-CR closed-form expressions can be easily adapted to **time-dependent travel times**!*

In fact, an interesting feature of our recourse strategies is that they can be easily adapted to a context in which the travel time along an arc depends on the moment the trip is performed. Let us define

$$t^{\text{dep}} = d^{\text{TD}}_{v,v'}(t^{\text{arr}})$$

as the time a vehicle must leave v in order to reach v' at time t^{arr}. In order to adapt the computation of g_1 to travel time dependency, one just needs to redefine expressions $d^{\min}_{r,w}$, $d^{\max}_{r,w}$ and $f(r,t,q)$. In fact, these are the only places where travel times are involved. We then have (this should be compared with the expressions at Sec. 5.2.1.2)

$$d^{\text{minTD}}_{r,w} = \max\{\underline{on}(w),\ \Gamma_r,\ d^{\text{TD}}_{w,r}(e_r)\}$$

and

$$d^{\text{maxTD}}_{r,w} = \min\{d^{\text{TD}}_{w,r}(l_r),\ \ S^{\text{TD}}_r\big(\overline{on}(w)\big)\}$$

where $S^{\text{TD}}_r(t^{\text{arr}}) = d^{\text{TD}}_{w,r}\big(\ d^{\text{TD}}_{r,w}(t^{\text{arr}}) - s_r\ \big)$ is the time at which a vehicle needs to depart from w in order to make a round trip to r, while servicing it, and reaching back w at time t^{arr}. The computation of $f^{\text{TD}}(r,t,q)$ is adapted as follows (see. Section 5.2.2.2):

$$\begin{aligned} f^{\text{TD}}(r,t,q) = {} & g^{\text{TD}}_1(r^-,\ S^{\text{TD}}_r(t),\ q - q_{r^-}) \cdot \delta(r^-,\ S^{\text{TD}}_r(t),\ q - q_{r^-}) \\ & +\ g^{\text{TD}}_1(r^-,\ t,\ q) \cdot \big(1 - \delta(r^-,\ t,\ q)\big)\ +\ g^{\text{TD}}_2(r^-,\ t,\ q). \end{aligned}$$

With $d^{\text{minTD}}_{r,w}$ and $d^{\text{maxTD}}_{r,w}$, the time-dependent versions of random functions g^{TD}_1 and g^{TD}_2 are simply obtained by replacing the f functions by their time-dependent version f^{TD}.

These are the only modifications required to adapt $\mathcal{R}^q$, and therefore $\mathcal{R}^\infty$, to time-dependent travel times. Quite similar modifications, although a bit more complicated, could also be applied to strategy $\mathcal{R}^{q+}$. This will not be developed here, as we do not consider the latter strategy the current study. Finally, equation (9.7) then becomes quite naturally

$$\mathscr{Q}^{\mathcal{R}'}(x,\tau) = \frac{1}{|R|} \sum_{r \in R} \sum_{\Delta \in H} \Delta \cdot \frac{g^{\text{TD}}_1\big(r,\ d^{\text{TD}}_{\text{w}(r),r}(\Gamma_r + \Delta)\big)}{p_r}. \tag{9.8}$$

In the next section, we explain how the time-dependent departure time function $d^{\text{TD}}_{v,v'}$ is computed in practice, within our specific operational context.

9.2 CASE STUDY: BRUSSELS POLICE DEPARTMENT

Our case study describes the problem faced by a particular subset of the police mobile units in Brussels, Belgium. Most of the units working every day for the police department are assigned to minor

interventions or safety control during particular events. Our case study concerns a specific team of police units, aimed at taking action on urgent interventions, such as road traffic accidents, violence or alarms. As a consequence, these units spend their time cruising the city, waiting for intervention requests. Based on the SS-VRP-R model, we investigate on the best re-location policies for each of these mobile units, thereby minimising the average expected intervention delay.

In this section, we describe how we derived the problem input data (graph nodes, travel times and request probabilities) for the historical data provided by the police department. The historical data consist in a list of recorded events, each corresponding to a call received from citizens (or alarms), or a situation spotted on-the-fly by the police unit (*e.g.* flagrante delicto). Each event is described by a timestamp and a GPS coordinates. It is also described by a type, allowing to distinguish between alarms, armed violences, traffic accidents, *etc.* The recorded period ranges from 2013 to 2017, included.

The problem input data must be realistically inferred from these available records, in a way that best describes the situation, while remaining both computationally and practically tractable. In particular, some parts of the data will be ignored while constructing our benchmark. In fact, we focus on a particular operational context, namely what happens in the city from 4am to 10pm, during regular working days. In what follows, the input data are inferred from historical records ranging from 2013 to 2016. Records of year 2017 are kept for experimental validation purposes only.

9.2.1 *Graph nodes: customer and waiting vertices*

Because our approach requires a finite set of customer and waiting vertices, the operational area, namely the city of Brussels, must be appropriately discretised. For practical reasons, such as to limit the size of the time-dependent travel time matrices, we limit the number of vertices to 150.

To this end, we look at the set of all recorded events and cluster them geographically, using a k-means algorithm, thus with $k = 150$. Figure 9.1 shows the distribution of these clusters, as well as the distribution of the events as a heat map. In what follows, the set of graph vertices V will then be constituted of the locations of these clusters and, consequently, customer and waiting vertices will then be taken amongst them: $C = V, W \subseteq V$. The configurations of waiting vertices are further described in Section 9.3. The depot is placed on the area of the Central Commissariat, hence belonging to the vertices: $0 \in V$.

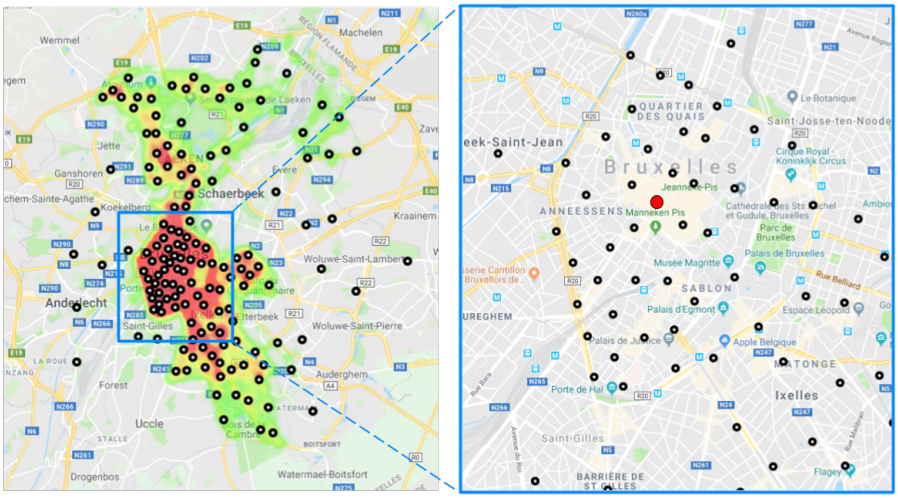


Figure 9.1: On the left, heat map of historical events according to the data, from 2013 to 2017. The concentration of the events along a vertical axis is due to the particular areas covered by the police department. The 150 clusters, which are the vertices composing V, are represented by circles. Right: focus on the city centre. The depot vertex is depicted in red, next to the main square.

9.2.2 *Time-dependent travel times*

Urban VRPs have their own constraints and specificities. Whereas travel times may not vary significantly for long, inter-city or inter-country journeys (or are they?), the time needed to connect two given locations within Brussels is in fact dependent of the time the trip is performed. Figure 9.2 shows how these travel times vary, for two different journeys illustrated in Figure 9.3, depending on both the moment and the type of the day: either working day (Tuesday) or day off (Sunday). These data have been retrieved by requesting Google Maps' API, for every five minutes of a 24-hour day.

Interestingly, the travel time variations during a working day, for the long urban trip between Porte de Hal and the Atomium (Figure 9.2, upper green curve), appends to be very similar to the variations depicted in Eglese et al., 2006. Based on this analysis, we split the 24 hours into 14 time bins. We use these time bins in order to construct our time-dependent travel-time matrices.

We hence retrieved, for each arc $(v, v') \in V^2$ of our graph and for each time bin $[t, t'[$, the predicted time needed for travelling from v to v' when leaving v at time t. This is achieved by requesting the Google Maps' API, while setting the departure time to t and departure date to either a working or an off day. The date is taken sufficiently late in the future, so that the prediction best reflects the historical average for such time of day and day of week, and do not reflect any current traffic consideration. In practice, by considering a working day, this ends up requiring $150^2 \times 14 = 315000$ travel times. Travel times during off

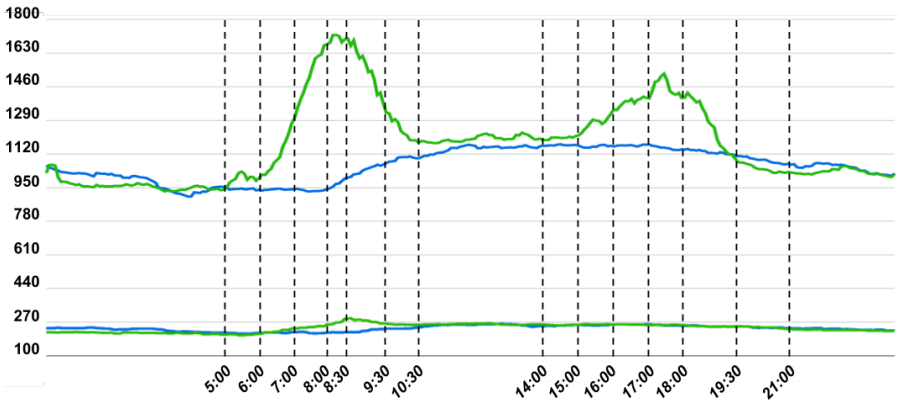


Figure 9.2: The predicted travel times of two different journeys, depending on the time of day and day of week (green: working day, blue: day off): from Manneken Pis to the Central Station (down) and from Porte de Hal to the Atomium (up). Time bins are separated by dashed lines.

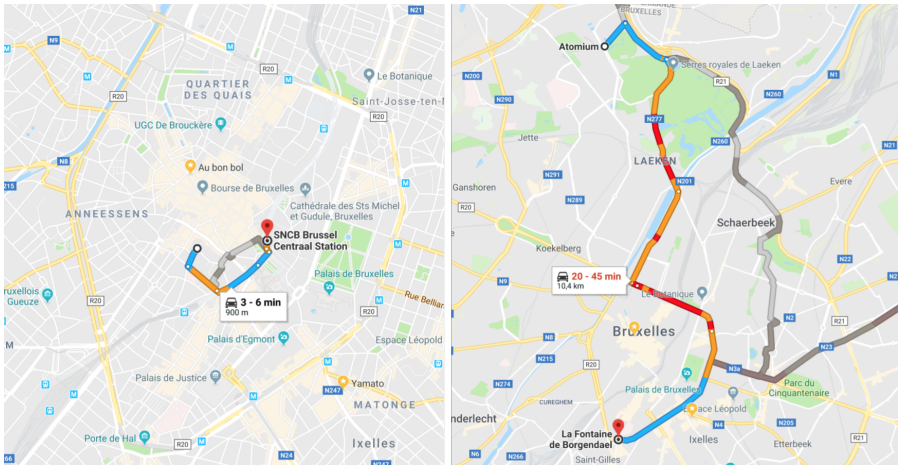


Figure 9.3: Left: a short urban path in Brussels, from Manneken Pis to the Central Station. Right: a long urban path, from Porte de Hal to the Atomium.

days are not required, as our benchmark considers regular working days only.

For each arc $(v, v') \in V^2$, these time-dependent matrices result in a piecewise constant function $f_{v,v'} : H \to H$ returning a travel time based on the time bin at which the trip is started. However, the leaps from one time bin to another in the step function $f_{v,v'}$ result in unrealistic properties. In particular, as discussed in Ichoua et al., 2003 and Fleischmann et al., 2004b, this results in a non-FIFO property.

The FIFO property states that a traveller that leaves v at time t cannot reach v' before another traveller that left v at a time $t' < t$. In other words, one cannot arrive earlier at destination by delaying its departure. A formal definition of the FIFO property is provided in Melgarejo et al., 2015. In order to restore the FIFO property, we compute our FIFO time-dependent travel time function $t^{\text{TD}}_{v,v'}$ by using the algorithm described in Eglese et al., 2006. It follows the approach proposed in Ichoua et al., 2003 and has the advantage of being computationally interesting when travel times are small, in average, compared to the time bins (which is, in fact, our case). Their procedure works as follows. Provided the travel distance along arc (v, v'), a piecewise constant *time-dependent speed function* is computed based on the original travel time function $f_{v,v'}$; whenever a trip must be spread across several time bins, the algorithm then makes use of the speed function to determine a total travel time that preserves the FIFO property. Note that instead of explicitely follow the algorithm they propose, a somehow sighlty more meaningful version can be devised by using dynamic programming. Let $v_{v,v'}(t) = f_{v,v'}(t)/\text{dist}_{v,v'}$ return the travel velocity along arc (v, v') at time t, based on its length $\text{dist}_{v,v'}$. On this basis, we obtain our FIFO time-dependent departure time function $d^{\text{TD}}_{v,v'}(t^{\text{arr}})$, discussed in Section 9.1.3, by following the recursive function:

$$\phi_{v,v'}(a,\ d) = \begin{cases} a - \frac{d}{v_{v,v'}(a)} & \text{if} \quad a - \frac{d}{v_{v,v'}(a)} > \underline{\text{bin}}(a) \\ \phi_{v,v'}\Big(\underline{\text{bin}}(a),\ d - v_{v,v'}(a) \cdot \big(a - \underline{\text{bin}}(a)\big)\Big) & \text{otherwise,} \end{cases} \tag{9.9}$$

returning the correct departure time, when arriving at time a, of someone who would travel along arc (v, v') for a distance d. Here $\underline{\text{bin}}(a)$ is simply the first time unit of the time bin that contains a. In fact, we necessarily have

$$\forall a' < a : \underline{\text{bin}}(a) < a' \quad \Rightarrow \quad v_{v,v'}(a') = v_{v,v'}(a),$$

which justifies that if $a' = a - d/v_{v,v'}(a)$ falls in the same time bin than a, then a' is the correct departure time. Otherwise, earlier than a the traveler was actually traveling at a travel speed $v_{v,v'}(a - \delta) \neq v_{v,v'}(a)$, meaning that at least two different speed rates are experienced along

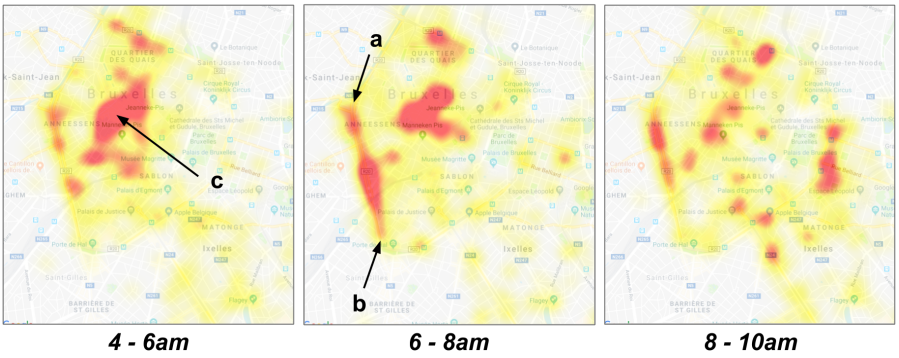


Figure 9.4: Evolution of how the events appear in average in the morning, during working days around the city centre. From 4am to 6am, events seem rather concentrated in downtown (**c**). From 6am to 8am, a significant part of the events occur along the main traffic lane, between points **a** and **b**. Events then tend to become sparser after the rush hour, from 8am to 10am.

arc (v, v'). The last piece of arc traveled, thereby at speed $v_{v,v'}(t^{\text{arr}})$ and starting at time $a = \underline{\text{bin}}(t^{\text{arr}})$, is then of length

$$l = \left(t^{\text{arr}} - v_{v,v'}(t^{\text{arr}}) \cdot \underline{\text{bin}}(t^{\text{arr}})\right).$$

In particular, up to time a a distane of $\text{dist}_{v,v'} - l$ has been traveled so far. The departure time, from v in order to reach v' at time t^{arr}, is then:

$$d^{\text{TD}}_{v,v'}(t^{\text{arr}}) = \phi_{v,v'}\left(t^{\text{arr}},\ \text{dist}_{v,v'}\right). \tag{9.10}$$

9.2.3 *Potential requests: exploiting historical data*

The history of events that occurred in the city during the observed years (2013 to 2016) tells us, for any period of interest (*e.g.* Mondays, 7am to 8am), the average event activity of a given area. This naturally provides an indication on the likelihood that an event will occur at a corresponding period, in the future. For instance, the heat maps in Figure 9.4 illustrate the observations that can be made on the data set. When looking at events that occurred during every working day morning, we notice a significant evolution from 4am to 10am, where traffic incidents of course play an important role during the rush hour.

Each historical event occurred at a specific location, at a specific date and time. We first partition the period of observation (2013 to 2016) into intervals of interest. We consider only regular working days in our benchmark: Monday to Friday, 4am to 8pm, excluding national holidays. Since they may not be representative of regular working days, the months of July, Augustus and December are also ignored, as most Belgian citizens take holidays during these periods. After filtering, 5180 events remain in the database, observed over 738 days. We thus have an average of 7 observed events per day.

Similarly, we aggregate events by area of interest. We define these areas as being the clusters described above, computed when discretising the geography of our operational context. Each event is therefore grouped to the closest vertex $v \in V$, in terms of coordinates. Doing so, the average distance between an event and its associated vertex is of 200 meters. The average number of events by cluster is of 34.

For a classical operational day, our entire time horizon thus represents 6 hours (4am to 10am). It is further divided in intervals of 30 minutes, called *time slots*. We finally associate a potential request $r = (c, \Gamma)$ to each pair of vertex $c \in V$ and time slot $\Gamma \in H$, where Γ is the first time unit of the corresponding time slot. When considering the entire horizon, this leads to $|R| = 150 \times 32 = 4800$ potential requests. We then approximate the request's probability p_r based on historical data, as the average number of time an event appeared in the area belonging to the cluster $c \in V$ and during the interval $[\Gamma, \Gamma + 30'[$ of the day.

9.3 EXPERIMENTS AND RESULTS

Our experiments here aim at answering the following questions. Under a two-stage assumption, that is, when excluding online reoptimization, is it possible to identify first-stage solutions that beat a simple, intuitive operational policy? Also, what is the impact of the time-dependent travel times on the quality of the first-stage decisions?

The first question is rather important. Based on the stochastic knowledge we were able to extract from our historical records, it will determine whether the operational model provided by our SS-VRPTW-CR recourse strategy $\mathcal{R}'$ is useful or not at taking good a priori decisions in the SS-VRP-R context. To that extend, first-stage solutions will be computed under various experimental conditions (*e.g.* with or without PFS, varying the set of waiting locations, *etc.*). The solutions obtained on the basis of the data from 2013 to 2016 will be confronted to the observations of 2017. The average behavior of the first-stage solutions will then be measured against the events recorded during year 2017, and compared to the average results obtained with a simple wait-and-serve policy. According to the expert's knowledge, within our simulations each intervention is assumed to last two hours.

***First-stage solutions** are optimized in light of the data collected from 2013 to 2016.*

9.3.1 *Simulations: wait-and-serve policy versus recourse strategy*

Similarly to experiments of Sections 7.3 and 8.3, we rely on the same simple *wait-and-serve* policy to decide vehicle actions, without taking the stochastic knowledge into account. During the simulation, the vehicles will therefore always react to online events of 2017 by assigning each new request to the closest available vehicle. Whereas such policy is in fact very simple to simulate, in real life it would already require

***Results** are obtained by simulating on the data recorded during 2017.*

a minimal communication between the vehicles, which must be able to share their positions and status.

On the contrary, when simulating based on a SS-VRPTW-CR first-stage solution, the vehicles are not even required to share their position or status. In fact, according to SS-VRPTW-CR recourse strategies, all the potential requests being pre-assigned to the waiting locations, each vehicle can operate in a totally independent way. Hence, all the intelligence here lies in the a priori sequences of waiting locations.

In practice, our SS-VRPTW-CR recourse strategy are easier to physically implement than the basic wait-and-serve policy.

Naturally, instead of relying on a priori decisions only, better performances can always be achieved by enabling a form of collaboration between the vehicles. This however implies complicated online decisions, which will be studied in Part iii of this thesis. Only the contribution of the a priori decisions is of interest for now.

9.3.2 *Solution method*

We use the same local search algorithm as for the former experiments of Section 8.3. In order to cope with the complexity of our real life problem, it may as well be combined to the Progressive Focus Search (PFS) meta-heuristic presented in Section 8.1.

As discussed in Section 9.1.2, in practice finding a solution in which a request is always satisfied under SS-VRPTW-CR recourse strategy $\mathcal{R}'$, is not trivial. In such context, it is more adequate to reason in the more general terms of *expected service delay of an accepted request*. We then implement our LS algorithm while replacing objective function (9.2) by the more general lexicographical bi-objective function $\mathcal{Q}^{\mathcal{R}'}_{\text{lex}}$, which optimises the expected number of accepted requests first, and their expected service delay second:

*The **SS-VRP-R objective function** is **approximated** by using a **bi-objective function** on the SS-VRPTW-CR: expected number of accepted requests with 20 minutes time window, and expected intervention delay on an accepted request.*

$$\mathcal{Q}^{\mathcal{R}'}_{\text{lex}}(x,\tau) = \left(\sum_{r\in R} \left(p_r - \mathrm{P}\{r \text{ is accepted}\}\right), \; \frac{1}{|R|} \sum_{r\in R} \sum_{\Delta\in H} \Delta \cdot \frac{g_1^{\text{TD}}(\Gamma_r + x - d_{w(r),r})}{\mathrm{P}\{r \text{ is accepted}\}} \right), \tag{9.11}$$

with $\mathrm{P}\{r \text{ is accepted}\} = \sum_{t=t_r^{\min}}^{t_r^{\min}} g_1^{\text{TD}}(r,t)$, and where g_1^{TD} is obtained by adapting g_1 as explained in Section 9.1.3. Solutions that are SS-VRP-R feasible thus have an objective value (z_1, z_2) with $z_1 = 0$.

We immediately see that objective function (9.11) is more convenient for a local search based method, as it facilitates transitions between SS-VRP-R feasible solutions. In fact, they appear to be very sparse when using the SS-VRPTW-CR recourse strategy $\mathcal{R}'$. The time window of an online request is always set to 20 minutes, allowing null or very low values for z_1 to be reached.

In practice however, we observe in Section 9.3.5 that, obtaining SS-VRP-R feasible solutions, with $z_1 = 0$, is computationally very difficult. Nevertheless, even when $z_1 > 0$, solutions having very low (z_1, z_2)

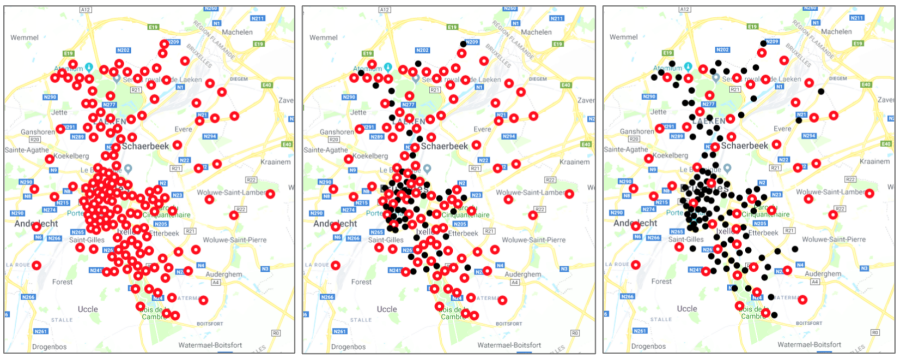


Figure 9.5: Configurations of waiting locations (empty red circles). Left: 150. Center: 100. Right: 50. Remaining vertices are displayed in black.

values constitute good operational basis, when tested under SS-VRP-R conditions.

9.3.3 *Waiting locations*

The size of the solution space greatly depends on the number of available waiting vertices.

There is no restriction on the locations where the police units can be relocated. Hence, it seems natural to define W, the set of possible waiting (re)location, as the set of all 150 vertices describing our urban area: $W = C = V$. However, the solution space, and therefore the computational performances, also depend on the number of waiting vertices. During the experiments, we will sometime use restricted subsets of V as waiting vertices. We consider three configurations, with $|W| \in \{50, 100, 150\}$. We have $W = V$ when $|W| = 150$. Let W^m denote the set W, such that $|W| = m$. The sets of waiting vertices are chosen so that $\forall m < m' : \ W^m \subset W^{m'}$. That way, a solution computed using W^m is also a valid solution when using $W^{m'}$. The sets of waiting vertices are depicted in Figure 9.5.

9.3.4 *Waiting time multiples and horizon scale*

Our time horizon represents the six-hour interval comprised between 4am and 10am. At its original scale, it has a resolution of one time unit per minute, that is, 360 time units.

Similarly to experiments of Section 8.3, both the horizon scale and the waiting time multiples can be adapted in order to simplify the computations. By reducing the resolution of the horizon, the complexity of computing the objective function $\mathscr{Q}^{\mathcal{R}'}_{\text{lex}}$ is decreased. With a scale of 2 for instance, each time unit counts for two minutes, leading to a reduced horizon of 180 time units only. Of course, such approximation comes at the price of a less accurate resulting expected value. Finally, by restricting the set of possible waiting times that can be assigned to vehicles at waiting locations, we greatly reduce the solution space.

		$\|W\| = 50$			$\|W\| = 100$			$\|W\| = 150$		
K	wm	$s = 1$	$s = 2$	$s = 5$	$s = 1$	$s = 2$	$s = 5$	$s = 1$	$s = 2$	$s = 5$
3	10	16.7	21.2	3.3	17.6	18.7	2.5	18.2	17.2	0.8
	30	13.9	20.4	2.7	19.7	19.3	0.5	20.2	18.8	0.5
	60	11.4	17.8	2.7	12.6	16.2	1.1	15.0	16.9	-0.3
4	10	26.8	24.1	2.7	28.9	24.1	2.4	28.4	23.4	2.3
	30	26.4	24.3	2.1	28.9	25.8	1.6	27.2	26.0	1.3
	60	22.1	22.5	2.1	24.9	22.1	1.2	25.2	21.8	0.2
6	10	40.0	36.0	1.8	40.4	32.5	5.0	39.6	34.1	3.3
	30	41.3	34.9	1.2	40.6	32.5	4.7	38.2	32.3	2.8
	60	36.0	33.4	1.2	36.5	31.9	3.3	37.0	31.9	1.9

Table 9.1: Average relative gains (in percentages) compared to the *wait-and-serve* policy, which passes the simulations (see Sec. 9.3.1) with an average intervention delay of 11.2 (3 vehicles), 9.9 (4 vehicles) and 9.5 minutes (6 vehicles). Each cell reports an average over 10 solutions computed with the LS algorithm, under the defined conditions: number of vehicles K, waiting time multiple wm, number of waiting locations $|W|$ and horizon scale s. Average travel times (non time-dependent) are used for both optimization and simulations.

Furthermore, assigning waiting times that are multiples of 10 minutes, for instance, is most probably accurate enough in practice.

9.3.5 *Results*

The first experiments will be carried based on constant, non time-dependent, average travels times. We will thus be able to study the impact of different experimental factors under a more classical version of our problem. Thereafter, we will confront these results on those obtained when the inherent time dependency of travel times are to be considered, in such urban context.

9.3.5.1 *Varying waiting time multiples, locations and horizon scale*

By using the local search algorithm alone, that is, without exploiting the PFS meta-heuristics, we obtain the results reported in Table 9.1. Each solution is optimized for one hour, for now by using average travel times, that is, non time-dependend ones. In other words, for now we optimize a bi-objective function that differs from (9.11), by using our classical non time-dependent g_1 function, instead of g_1^{TD}. For each solution, we report averages over 10 runs of the LS algorithm. Provided K vehicles, we observe that the gains vary greatly with the experimental conditions (wm, $|W|$, s). With three (*resp.* six) vehicles, the average gains vary from -0.3% (*resp.* 1.9%) to 21.2% (*resp.* 41.3%).

		$\|W\| = 50$			$\|W\| = 100$			$\|W\| = 150$		
K	wm	$s = 1$	$s = 2$	$s = 5$	$s = 1$	$s = 2$	$s = 5$	$s = 1$	$s = 2$	$s = 5$
3	10	15.1	21.1	3.5	16.6	18.5	3.1	17.5	17.3	1.3
	30	13.4	20.4	3.0	18.8	18.6	0.7	19.9	19.3	0.9
	60	10.1	17.8	3.0	11.6	15.9	1.6	13.9	17.6	0.1
4	10	25.1	23.4	2.3	27.9	23.1	2.3	27.3	23.5	2.0
	30	24.7	23.4	1.8	27.9	24.4	1.6	26.2	25.2	1.2
	60	21.0	21.4	1.8	23.1	20.7	1.1	24.8	21.3	0.2
6	10	38.4	33.7	1.2	38.8	30.7	3.8	38.5	32.5	2.9
	30	40.1	32.8	0.6	39.7	30.7	3.4	37.0	30.8	2.5
	60	35.4	31.1	0.7	35.0	29.4	2.2	36.1	30.0	1.5

Table 9.2: Average relative gains (in percentages) compared to the *wait-and-serve* policy (3 vehicles: 11.0 minutes; 4 vehicles: 9.8; , 6 vehicles: 9.7). Solutions (the same as for Table 9.1) are now evaluated by simulating under the time-dependent travel times.

The best results with respect to the number of vehicles are highlighted in the table. Obviously, using scale 5 does not provide any significant gain, which sounds natural as under scale 5 the horizon is discretized in five-minute time units, which seems clearly not accurate enough when the average intervention delay is less than ten minutes.

9.3.5.2 *Impact of time-dependent travel times*

We now replay all the simulations while taking time-dependency into account. All the first-stage solutions we computed so far were obtained while assuming constant, average, travel times. We hence measure how they actually behave under realistic time-dependent travel times. From now the displayed results are simply more accurate, since the operations of year 2017 are simulated under time-dependency assumptions.

When dealing with a fleet of six vehicles, intervention delays can be reduced of $\sim$ 40% compared to a basic non-anticipative policy!

Those new, more accurate, results are provided in Table 9.2. We directly notice that the average gains are globally worse, but not much different from those computed in Table 9.1 (*i.e.*, by using constant travel times). The best gains moves from 21.2% to 21.1% in the 3-vehicle case, from 28.9% to 27.9% with 4 vehicles, and from 40.4% to 40.1% with 6 vehicles. The reason is twofold. First, the gains are expressed in terms of relative differences, and the average delays of both wait-and-serve and the SS-VRP-R solutions are similarly impacted (either positively or negatively) by the time-dependent travel times. Second, we already observed, in Figure 9.2 of Section 9.2.2, that travel time variations along a path tend to be more important as the length of the path increases. However, the average travel durations only range from about six minutes (for SS-VRP-R solutions) to ten minutes (for wait-and-serve). In fact, all the point of solving the SS-VRP-R is to minimize

		$\|W\| = 50$			$\|W\| = 100$			$\|W\| = 150$		
K	wm	$s = 1$	$s = 2$	$s = 5$	$s = 1$	$s = 2$	$s = 5$	$s = 1$	$s = 2$	$s = 5$
3	10	12.8	20.7	2.8	12.8	19.0	1.4	16.6	19.5	0.5
	30	13.3	19.5	2.3	12.4	20.3	1.2	16.1	19.1	0.4
	60	8.4	17.8	1.8	10.4	18.1	1.5	12.3	17.4	0.6
4	10	20.3	25.1	1.5	24.7	27.0	1.5	23.2	25.0	-0.2
	30	19.5	24.1	1.1	20.7	24.6	1.3	22.2	23.8	-0.4
	60	14.2	21.7	0.5	19.3	22.0	1.3	23.2	22.1	-0.4
6	10	32.1	33.5	0.4	33.8	33.0	2.8	36.4	31.3	1.5
	30	32.9	31.9	0.0	36.3	31.3	2.5	36.1	31.7	1.6
	60	29.7	30.0	-0.7	33.1	29.1	2.3	32.0	30.8	1.0

Table 9.3: Average relative gains (in percentages) compared to the *wait-and-serve* policy (3 vehicles: 11.0 minutes; 4 vehicles: 9.8; , 6 vehicles: 9.7). Solutions are now optimized (and evaluated) while considering time-dependent travel time matrices. We highlight the cells that are improved, compared with Table 9.2.

these durations. As a consequence, the time-dependent travel times along these paths reveal, in general, quite close to their averages.

We also compute new solutions, which are now obtained by taking time-dependency into account during the optimization process. In other words, we now optimize the bi-objective function (9.11). Table 9.3 shows the relative gains obtained, under the same experimental contexts. In particular, the computation time remains set to one hour. We highlight the gains that reveal better than those obtained while optimizing under constant travel times, that is, improving the gains of Table 9.2.

We directly notice two important empirical results. First, exploiting more accurate, time-dependent, travel times does not permit to improve our results in general. This can be easily explained by the fact that the improvement in the travel time accuracy, which is necessarily not that much significant in general (as discussed in the previous paragraph), does not compensate the increased computational effort due to the time-dependent functions $f^{\text{TD}}_{v,v'}$ and $d^{\text{TD}}_{v,v'}$. In fact, the number of iterations performed by our local search optimizer is from 2 to 3 times lower, when dealing with time-dependent travel times. That also explains the second important result of Table 9.3: improvements are only observed when optimizing under scales 2 and 5.

As a consequence, we are forced to conclude that considering time-dependency during the optimization process is not suited for this case study. The major reasons are therefore that *a)* time-dependency increases the computational effort, thus decreasing the diversification within our local search approach, whereas *b)* the intensification is only slightly increased as the travel times do not actually vary that much in general.

*Time-dependency is apparently **not** helpful in the context of the current case-study.*

In what remains of the current Chapter, time-dependency is only exploited during the simulations (see Section 9.3.1).

9.3.5.3 *Progressive Focus Search*

We now try to improve the best results obtained so far by exploiting the PFS meta-heuristic, previously introduced in Chapter 8. The objective function is still approximated by using a scaling factor on the horizon resolution, as described in Section 8.2.2. Yet, here we use both the waiting time multiples and the number of waiting locations in order to reduce the solution space.

Let us designate a particular PFS instantiation by the tuple (s, wm, wl), where s stands for the horizon scale, wm the waiting time multiple and wl the size of the waiting vertex set. We hence consider all the following 7 update policies as candidates for PFS instantiation:

1. **s**, where only the scale varies:
   $(\mathbf{2}, 10, 150) \rightarrow (\mathbf{1}, 10, 150)$.
2. **wm**, where only the waiting multiple varies:
   $(1, \mathbf{60}, 150) \rightarrow (1, \mathbf{30}, 150) \rightarrow (1, \mathbf{10}, 150)$.
3. **wl**, where only the waiting vertex set varies:
   $(1, 10, \mathbf{50}) \rightarrow (1, 10, \mathbf{100}) \rightarrow (1, 10, \mathbf{150})$.
4. **s_wm**, where both scale and waiting multiple vary:
   $(\mathbf{2}, \mathbf{60}, 150) \rightarrow (\mathbf{2}, \mathbf{30}, 150) \rightarrow (\mathbf{1}, \mathbf{10}, 150)$.
5. **s_wl**, where both scale and waiting vertices vary:
   $(\mathbf{2}, 10, \mathbf{50}) \rightarrow (\mathbf{2}, 10, \mathbf{150}) \rightarrow (\mathbf{1}, 10, \mathbf{150})$.
6. **wm_wl**, where both waiting multiple and vertices vary:
   $(1, \mathbf{60}, \mathbf{50}) \rightarrow (1, \mathbf{30}, \mathbf{100}) \rightarrow (1, \mathbf{10}, \mathbf{150})$.
7. **all**, where all three components vary:
   $(\mathbf{2}, \mathbf{60}, \mathbf{50}) \rightarrow (\mathbf{2}, \mathbf{30}, \mathbf{150}) \rightarrow (\mathbf{1}, \mathbf{10}, \mathbf{150})$.

Table 9.4 shows the average results obtained, in relative gains over the wait-and-serve policy, of our PFS instantiations compared to the best LS configurations, as observed in Table 9.2. We directly note that our PFS instantiations generally fail at improving the gains. Only a minor improvement of 0.2% is observed under $K = 6$ vehicles. Furthermore, it appears that no PFS instantiation significantly stands out.

9.3.5.4 *Short computation times*

We now put the emphasis of our experimentations on average results under very short computation times. This is mainly motivated by the next (and last) part of this thesis: *optimizing online decisions*. In fact, dynamic operational decisions require fast, reactive answers, hence severely limiting the allowed online computation times. Furthermore,

			LS		PFS						
			wl					**s**	**s**	**wm**	
K	s	wm	50	100	**s**	**wm**	**wl**	**wm**	**wl**	**wl**	**all**
3	1	10	15.1	16.6	16.9	16.8	16.8	16.1	15.4	19.1	14.4
		30	13.4	18.8							
	2	10	**21.1**	18.5							
		30	20.4	18.6							
4	1	10	25.1	**27.9**	26.1	25.3	25.0	23.7	23.1	24.0	23.0
		30	24.7	**27.9**							
	2	10	23.4	23.1							
		30	23.4	24.4							
6	1	10	38.4	38.8	37.9	38.8	**40.3**	36.9	37.6	38.5	38.3
		30	40.1	39.7							
	2	10	33.7	30.7							
		30	32.8	30.7							

Table 9.4: Average relative gains (in percentages) of LS and PFS, provided one hour computation, compared to the *wait-and-serve* policy.

			LS		PFS						
			wl					**s**	**s**	**wm**	
K	s	wm	50	100	**s**	**wm**	**wl**	**wm**	**wl**	**wl**	**all**
3	1	10	-6.2	3.3	11.4	8.7	5.4	11.3	11.2	6.1	10.9
		30	-3.5	1.3							
	2	10	12.3	**12.4**							
		30	11.8	**12.4**							
4	1	10	5.3	10.0	16.2	11.3	9.8	16.0	15.3	10.0	15.5
		30	7.5	11.6							
	2	10	16.3	**17.2**							
		30	16.7	16.8							
6	1	10	14.4	16.9	22.5	11.7	11.1	23.4	22.9	11.4	23.8
		30	15.1	23.1							
	2	10	22.2	**24.2**							
		30	23.1	24.0							

Table 9.5: Average relative gains (in percentages) of LS and PFS, provided 30 seconds computation, compared to the *wait-and-serve* policy.

the interest of PFS could potentially increase as computation times decrease.

Table 9.5 shows the average results that are now obtained while severely limiting the computation time, allowing only to 30 seconds. For each configuration of the LS algorithm or instantiation of PFS, the average gains over wait-and-serve are computed over 50 optimized first-stage solutions (instead of the previously 10 runs). We notice that, although PFS still does not provide an improvement, the gaps with LS tend to decrease. The scale factor reveals to be of critical importance. In LS, due to the limited allowed computation time, scale 2 provides significantly better gains than scale 1. Furthermore in PFS, instantiations that do not imply a variation of the scale (namely **wm**, **wl** and **wm_wl**), thus operating under scale 1 only, perform significantly worse.

We hence design the following new PFS instantiations, potentially more suited under short computation times:

1. **2_wm**: $(2, 60, 100) \rightarrow (2, 10, 100)$.
2. **2_wl**: $(2, 10, 50) \rightarrow (2, 10, 100)$.
3. **2_wml**: $(2, 60, 50) \rightarrow (2, 10, 100)$.
4. **2_wml3**: $(2, 60, 50) \rightarrow (2, 10, 50) \rightarrow (2, 10, 100)$.

Table 9.6 reports their average results as well as those of LS under scale 2, for computation times varying from 60 down to 10 seconds. Performances of both LS and PFS are now similar, when provided 60 or 30 seconds. Under 20 and 10 seconds, the performances of PFS decrease quickly, in particular, when the number of vehicles increases. Again, it seems that PFS fails at improving the average performances, as LS with scale 2, and in particular with 100 waiting locations, significantly outperforms it on these short computation times.

A question that appears is whether the Simulated Annealing metaheuristic, providing the diversification in both LS and PFS, may be deterioring the average performances under such short computation times. In fact, when provided only a few seconds, such diversification mechanism is maybe not a luxury that can be afforded. We hence test, and report in Table 9.7 the results obtained when the Simulated Annealing initial temperature of Algorithm 5 is set to $T_{\text{init}} = 1$ (instead of 2), with a cooling factor $f_T = 0.9$ (instead of 0.95). We now observe a significant improvement of both LS and PFS, compared to results in Table 9.6. In particular, PFS performs really better when its internal diversification mechanism is lowered. Although it still fails at outperforming LS, the performances of PFS are now globally similar.

Table 9.8, the average results that are obtained when diversification is completely removed. The simulated annealing mechanism initially embedded in both LS and PFS is simply removed, and the local search

				LS		PFS			
				wl					
60s	*K*	*s*	*wm*	50	100	**2_wm**	**2_wl**	**2_wml**	**2_wml3**
	3	2	10	**15.1**	11.5	12.8	13.1	12.6	12.3
			30	11.4	11.9				
	4	2	10	16.2	16.7	15.5	**17.5**	16.6	16.5
			30	15.7	16.8				
	6	2	10	22.1	**24.5**	23.7	22.8	23.3	22.7
30s			30	22.4	23.9				
	3	2	10	10.5	11.2	**11.9**	10.6	11.2	10.3
			30	10.3	11.3				
	4	2	10	14.8	**16.4**	15.0	14.8	15.3	15.8
			30	14.0	16.1				
	6	2	10	21.1	22.3	22.0	22.3	**22.4**	22.0
20s			30	21.8	22.4				
	3	2	10	8.4	**10.5**	10.4	**10.5**	8.4	5.2
			30	9.1	9.5				
	4	2	10	12.4	14.2	11.6	7.3	10.5	4.3
			30	**15.0**	14.9				
	6	2	10	18.5	21.0	15.6	10.9	17.0	11.7
10s			30	19.2	**21.9**				
	3	2	10	6.2	9.4	3.8	2.2	4.6	5.5
			30	7.4	**9.5**				
	4	2	10	9.2	12.3	2.8	2.8	5.5	4.8
			30	11.2	**13.6**				
	6	2	10	15.7	**21.6**	6.8	5.0	7.2	3.5
			30	18.2	20.1				

Table 9.6: Average relative gains (in percentages) of LS and PFS, under short computation times, compared to the *wait-and-serve* policy.

				LS		PFS			
				wl					
60s	*K*	*s*	*wm*	50	100	**2_wm**	**2_wl**	**2_wml**	**2_wml3**
	3	2	10	15.6	15.1	14.3	14.6	**16.2**	15.4
			30	15.0	14.5				
	4	2	10	**21.4**	19.6	19.9	20.3	20.4	20.7
			30	21.3	19.6				
	6	2	10	**29.0**	27.9	26.7	28.2	28.5	28.5
30s			30	28.5	27.2				
	3	2	10	**15.5**	13.8	13.5	13.5	13.2	14.1
			30	14.5	13.4				
	4	2	10	19.8	18.7	18.4	**19.9**	17.2	17.2
			30	18.8	18.1				
	6	2	10	25.9	**26.3**	25.2	24.7	24.2	24.3
20s			30	24.7	24.9				
	3	2	10	13.0	12.9	11.4	12.0	13.2	12.1
			30	**13.6**	11.8				
	4	2	10	16.5	16.5	16.6	**17.3**	17.1	16.6
			30	16.2	16.2				
	6	2	10	23.4	**25.2**	24.4	23.9	24.7	25.0
10s			30	22.7	24.3				
	3	2	10	9.9	10.4	11.0	9.2	10.2	8.0
			30	**11.3**	10.6				
	4	2	10	15.0	**15.4**	14.6	12.4	13.2	12.9
			30	15.1	15.3				
	6	2	10	21.1	**22.2**	21.3	20.2	21.3	19.7
			30	20.1	21.8				

Table 9.7: Average relative gains (in percentages) of LS and PFS, under short computation times. Lower internal diversification: Simulated Annealing with $T_{\text{init}} = 1, f_T = 0.9$.

				LS		PFS			
				wl					
60s	*K*	*s*	*wm*	50	100	**2_wm**	**2_wl**	**2_wml**	**2_wml3**
	3	2	10	-2.2	-0.4	**2.0**	-4.7	-1.3	-1.6
			30	-6.0	-1.9				
	4	2	10	2.4	8.0	**8.1**	3.3	2.6	2.5
			30	0.0	4.6				
	6	2	10	12.0	15.5	**17.3**	11.6	13.1	12.5
30s			30	11.4	15.6				
	3	2	10	-3.6	-1.0	**2.1**	-2.5	-1.7	-2.0
			30	-6.5	0.4				
	4	2	10	0.9	**8.6**	5.6	2.1	1.7	2.1
			30	0.5	5.3				
	6	2	10	13.1	16.1	**17.7**	12.0	12.5	12.8
20s			30	9.7	15.3				
	3	2	10	-3.7	1.4	**3.7**	-2.1	-3.9	-2.1
			30	-5.0	-2.0				
	4	2	10	3.7	7.3	**8.9**	0.8	2.7	1.0
			30	-0.9	7.5				
	6	2	10	12.4	**17.8**	16.2	12.5	12.7	12.8
10s			30	10.5	14.7				
	3	2	10	-1.8	**1.7**	0.9	-4.3	-3.4	-2.1
			30	-3.2	-0.6				
	4	2	10	1.1	6.4	**8.1**	4.2	2.4	4.0
			30	-1.8	3.6				
	6	2	10	8.9	**16.8**	16.7	10.6	10.4	9.8
			30	9.5	14.5				

Table 9.8: Average relative gains (in percentages) of LS and PFS, under short computation times. No internal diversification mechanism: *Hill Climbing*.

process never moves to a degrading solution. Namely, the LS algorithm now reduces to a simple *Hill Climbing* heuristic, thus being also the embedded optimizer Θ of PFS in Algorithm 4. We observe that it results in a dramatic deterioration of the performances, both in LS and PFS. However, quite interesting is the fact that PFS now widely outperforms LS, in its **2_wm** instantiation. This is mainly due to the fact that the PFS meta-heuristic, by modifying its solution space during the optimization process, somehow tends to compensate the lack of diversification.

9.4 CONCLUSIONS

In this chapter, we described the SS-VRP-R as a practical modeling framework of the real world problem of police units management in Brussels. While relying on the intervention requests observed from 2013 to 2016, we showed how the equations previously introduced for the SS-VRPTW-CR can be adapted for the SS-VRP-R, minimizing the expected average intervention delay. Experiments are conducted while considering 2017's observations as a validation benchmark.

Coupled with a simple recourse strategy, first-stage solutions obtained under the SS-VRP-R/SS-VRPTW-CR framework reduce the average mean intervention delays by more than 30%, compared to a basic wait-and-serve policy. That improvement is due to the preventive nature of the first-stage solutions, computed in light of stochastic knowledge, and which are composed of sequences of anticipative relocation instructions for the police patrols.

Finally, in order to stick with the reality of the urban context, travel time dependency is introduced in the computational models. The results we obtain are however quite surprising. In fact, experimentations under both constant and time-dependent travel times show that exploiting time-dependency is not interesting in the context of the current case-study. Namely, it appears that the contribution of time-dependency does not compensate the increased computational cost, leading to first-stage solutions of noticeably less expected quality when computed in light of time-dependent travel time matrices. Furthermore, while time-dependency is known to have a significant impact whenever time windows are involved, our SS-VRP-R case-study does not involve any deadlines. Yet, time-dependency remains useful in order to perform more accurate simulations, even though the accuracy difference also reveals here not to be so important.

We also tested the PFS meta-heuristic, previsouly introduced in Chapter 8, while varying the available computational time on our real world instances. The results show that PFS becomes competitive when used under very short computation times. Allowing one hour computation time, simple LS performs significantly better in average. When computation times are limited to 60 seconds or less, we discovered

that both LS and PFS may suffer from a too high diversification rate. Moderating the diversification by tuning the Simulated Annealing parameters hence allowed us to improve performances. Unlike the experiments conducted on the Lyon benchmark, in Section 8.3, the use of the PFS meta-heuristic did not outperform the basic LS algorithm on our real-world case study.

END OF PART II. This ends our discussion on two-stage optimization, and thus the current thesis part. During the next chapters, the focus will be moved to online decisions, by considering the underlying multistage reoptimization problem.

Part III

MULTISTAGE VRPTWS: OPTIMIZING ONLINE DECISIONS

10

SOLVING ONLINE VRP'S BY SAMPLING

This chapter is based on Saint-Guillain et al., 2015.

We consider a dynamic vehicle routing problem with time windows and stochastic customers (DS-VRPTW), such that customers may request for services as vehicles have already started their tours. Since the requests come with time windows, none all of them can be satisfied in general, and the goal is to manage the operations such as the number of unsatisfied ones is minimized at the end of the operational day.

To solve this problem, the goal is to provide a decision rule for choosing, at each time step, the next action to perform in light of known requests and probabilistic knowledge on requests likelihood. We present a new heuristic method for solving the DS-VRPTW, based on a Stochastic Programming modeling. By definition, our approach enables a *higher level of anticipation* than heuristic state-of-the-art methods. In particular, we show that our decision rule fully integrates *nonanticipativity constraints*, previously described in Section 2.5, so that it leads to better decisions in our stochastic context.

Unlike (most of) sampling-based approaches, our decision rule integrates the ***nonanticipativity property****.*

The resulting new online decision rule, called Global Stochastic Assessment (GSA), comes with a theoretical analysis that clearly defines the nature of the method. We propose a *new waiting strategy* together with a heuristic algorithm that embeds GSA. We compare GSA with the state-of-the-art method MSA from Bent and Van Hentenryck, 2004b (see Section 4.2), and provide a comprehensive experimental study that highlights the contributions of existing and new waiting and relocation strategies. Experiments on dynamic and stochastic benchmarks, which include instances of different degrees of dynamism, show that not only our approach is competitive with state-of-the-art methods, but also enables to compute meaningful offline solutions to fully dynamic problems where absolutely no a priori customer request is provided.

The chapter is organized as follows. Section 10.1 describes the problem. GSA is then presented in Section 10.2. Section 10.3 describes an implementation that embeds GSA, based on heuristic local search. Finally, section 10.4 summarizes the experimental results. A conclusion follows in section 10.5.

10.1 PROBLEM DESCRIPTION: THE DS-VRPTW

In this section we formally define the DS-VRPTW, described informally in Section 1.2.1. Furthermore, the problem input data have already been introduced in Chapter 3. In what follows however, we consider

$V = \{0\} \cup W = \{0\} \cup C$, that is, the set W of waiting vertices coincides with the customer regions C.

10.1.1 *DS-VRPTW solution.*

*Some requests are **accepted**, while the others are **rejected**. Accepted requests must be satisfied by the end of the day.*

At the end of the time horizon, a solution is a subset of requests $A^h \subseteq \xi^{1..h}$ together with k routes. Requests in A^h are said to be *accepted*, whereas requests in $\xi^{1..h} \setminus A^h$ are said to be *rejected*. The routes must satisfy the constraints of the classical VRPTW, restricted to the subset A^h of accepted requests. Each route starts at the depot at a time $t \geq 1$ and end at the depot at a time $t' \leq H$. For each accepted request $r \in A^h$, there must be exactly one vehicle present at customer region c_r, at a time t' comprised in $e_r \leq t' \leq l_r$, in order to start the service of the request. The vehicle's current load should be of at most $Q - q_r$. Finally, the vehicle leaves c_r at a time $t'' \geq t' + s_r$. The goal is to minimize the number of rejected requests.

*The horizon is discretized in **time units**. Operational decisions, vehicle and request acceptance, take place at the very beginning of each time unit.*

As not all requests are known at time 0, the solution cannot be computed offline, and it is computed during the online execution. More precisely, at each time $t \in H$, a decision x^t, consisting of a set of operational actions, is computed. Each decision x^t is composed of two parts. First, for each request $r \in \xi$ revealed at time Γ_r, the decision x^t must specify whether to accept or reject the request. Second, x^t must give operational decisions for each vehicle at time t: service a request, travel towards a vertex, or wait at its current position. At time 0, before the online execution, a priori decisions x^0 are computed offline.

A solution is a sequence of decisions $x^{0..h}$ which covers the whole time horizon. This sequence must satisfy VRPTW constraints, *i.e.*, the decision of $x^{0..h}$ must define k routes such that each request accepted in $x^{0..h}$ is served once by one of these routes within its time window, while respecting capacity constraints. We define the objective function r such that $r(x^{0..t})$ is $+\infty$ if $x^{0..t}$ does not satisfy VRPTW constraints, and $r(x^{0..t})$ is the number of requests rejected in $x^{0..t}$ otherwise. A solution is a decision sequence $x^{0..h}$ such that $r(x^{0..h})$ is minimal at the end of the horizon.

10.1.2 *Multistage stochastic formulation*

We note $X^t = X\big(x^{0..t-1}, \xi^{1..t}\big)$ the set of decisions that are valid at a time unit $t \in H_0$. Clearly, X^t depends on the past decisions $x^{1..t-1}$ and the online requests ξ^t that just appeared. We also note $X^{t..t'}$ the sequence of sets $\langle X^t, \ldots, X^{t'} \rangle$ where $0 \leq t \leq t' \leq h$. A each time t, given the sequence $x^{0..t-1}$ of past decisions, the best action x^t is obtained by solving the following multistage stochastic program:

$$\min_{x^t \in X^t} \mathrm{E}_{\xi^{t+1}} \left[\min_{x^{t+1} \in X^{t+1}} \mathrm{E}_{\xi^{t+2}} \left[\cdots \min_{x^{h-1} \in X^{h-1}} \mathrm{E}_{\xi^h} \left[\min_{x^h \in X^h} r(x^{0..h}) \right] \cdots \right] \right] \quad (10.1)$$

where $\mathrm{E}_{\tilde{\xi}}[\cdot]$ is the expectation operator over the specific random variable $\tilde{\xi}_i$, defined by

$$\mathrm{E}_{\tilde{\xi}_i}[f(\tilde{\xi}_1, \ldots, \tilde{\xi}_i, \ldots, \tilde{\xi}_j)] = \sum_{\xi_k \in \mathcal{S}_{\tilde{\xi}_i}} p(\xi_k)\, \mathrm{E} f(\tilde{\xi}_1, \ldots, \xi_k, \ldots, \tilde{\xi}_j).$$

Note that the use of this specific expectation operator is only of readability purposes. It aims at highlighting the strict order in which the information is incrementally revealed, during the online operations. In other words, it further emphasizes that equation (10.1) enforces *nonanticipativity constraints* so that, at each time $t' > t$, we consider the decision $x^{t'}$ that minimizes the expectation with respect to $\tilde{\xi}^{t'}$ only, without considering the possible realizations of $\tilde{\xi}^{t'+1..h}$. In fact, equation (10.1) is equivalent to the general pointwise multistage formulation (2.11) introduced in Chapter 2, by defining the c^t costs so that $c^0 x^0 + \ldots + c^h x^h = r(x^{1..h})$.

Similarly, we note that formulation (10.1) clearly differs from the two-stage stochastic problem defined by equation (10.2):

$$\min_{x^t \in X^t} \mathrm{E}_{\tilde{\xi}^{t+1..h}}[\min_{x^{t+1..h} \in X^{t+1..h}} r(x^{0..h})] \tag{10.2}$$

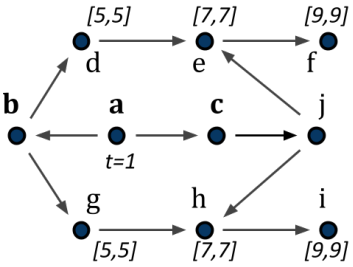
Eq. (10.2) relaxes the nonanticipativity constraints and considers the best sequence $x^{t+1..h}$ for each realization $\xi^{t+1..h} \in \tilde{\xi}^{t+1..h}$. Therefore, equation (10.1) may lead to a larger expectation of r than equation (10.2), as it is more constrained. However, the expectation computed in equation (10.1) leads to better decisions in our context where some requests are not revealed at time t. This is illustrated in Figure 10.1.

Relaxing the nonanticipativity constraints may lead to subobtimal decisions.

10.2 THE GLOBAL STOCHASTIC ASSESSMENT DECISION RULE

The two-stage stochastic problem defined by equation (10.2) may be solved by a sampling solving method such as MSA, which solves a deterministic VRPTW for each possible scenario (i.e., realization of the random variables) and selects the action x^t which minimizes the sum of all minimum objective function values weighted by scenario probabilities. However, we saw that equation (10.2) does not enforce nonanticipativity constraints because the different deterministic VRPTWs are solved independently.

In order to enforce nonanticipativity constraints while enabling sampling methods, we push these constraints in the computation of the optimal solutions for all different scenarios. Instead of computing these different optimal solutions independently, we propose to compute them all together so that we can ensure that whenever two scenarios share a same prefix of realizations, the corresponding actions are enforced to be equal.



t :	2	3	4	5	6	...	9
ξ_1^t:	∅	∅	d,e,f	∅	∅	...	∅
ξ_2^t:	∅	∅	g,h,i	∅	∅	...	∅

At time *t=1*, there is only one vehicle which is on vertex ***a***. We have to choose between two possible actions: travel to ***b*** or travel to ***c***.

Figure 10.1: A simple example of nonanticipation. The graph is displayed on the left. Time windows are displayed in brackets. For every couple of vertices (i, j), if an arrow $i \to j$ is displayed then $t_{i,j} = 2$; otherwise $t_{i,j} = 20$. To simplify, we consider only 2 equiprobable scenarios (displayed on the right). These scenarios have the same prefix (at times 2 and 3 no request is revealed) but reveal different requests at time 4. When using equation (10.1) at time $t = 1$, we choose to travel to c as the expected cost with nonanticipativity constraints is 1: At time 4, only one scenario will remain and if this scenario is ξ_1 (resp. ξ_2), request $(d, 4)$ (resp. $(g, 4)$) will be rejected. When using equation (10.2), we choose to travel to b as the expected cost without nonanticipativity constraints is 0 (for each possible scenario, there exists a sequence of actions which serves all requests: travel to d, e, and f for ξ_1 and travel to g, h, and i for ξ_2). However, if we travel to b, at time 3 we will have to choose between traveling to d or g and at this time the expected cost of both actions will be 1.5: If we travel to d (resp. g), the cost with scenario ξ_1 is 0 (resp. 3) and the cost with scenario ξ_2 is 3 (resp. 0). In this example, the nonanticipativity contraints of multistage problem (10.1) thus leads to a better action than the two-stage relaxation (10.2).

At each time $t \in H$, let $\mathcal{S}_{\boldsymbol{\xi}}^{t+1..t'}$ be the support of $\boldsymbol{\xi}^{t+1..t'}$, $t < t'$. Given the sequence $x^{0..t-1}$ of past decisions, we choose action x^t by using equation (10.3)

$$x^t = \arg\min_{x^t \in X^t} \mathcal{Q}(x^{0..t}, \boldsymbol{\xi}^{t+1..h}) \tag{10.3}$$

which is called the deterministic equivalent problem of equation (10.1), with the expected multistage value function:

$$\mathcal{Q}(x^{0..t}, \boldsymbol{\xi}^{t+1..h}) = \sum_{\xi_k \in \mathcal{S}_{\boldsymbol{\xi}}^{t+1..h}} p(\xi_i) \min_{x_k^{t+1}, \ldots, x_k^h} r(x^{0..t}.x_k^{t+1..h}) \tag{10.4}$$

$$\text{s.t.} \quad \forall \xi_i \in \mathcal{S}_{\boldsymbol{\xi}},\ t < t' \leq h :$$

$$x_i^{t'} \in X\left(x^{0..t}.x_i^{t+1..t'-1},\ \xi_i^{1..t'}\right) \tag{10.5}$$

$$\forall \xi_j \in \mathcal{S}_{\boldsymbol{\xi}} :\ \xi_i^{1..t'} = \xi_j^{1..t'} \ \Rightarrow\ t < t'' \leq t' :\ x_i^{t''} = x_j^{t''} \tag{10.6}$$

The nonanticipativity constraints (10.6) state that, whenever two scenarios ξ_i and ξ_j are identical from time $t+1$ to time t', their respective decisions for times $t+1, \ldots, t'$ must be identical as well (Shapiro et al., 2009).

Solving equation (10.3) is computationally intractable for two reasons. First, since the number $|\mathcal{S}_{\xi}^{t+1..h}|$ of possible realizations of $\xi^{t+1..h}$ is exponential in the number of vertices and in the remaining horizon size $h - t$, considering every possible scenario is intractable in practice. We therefore consider a limited multiset $\widetilde{\mathcal{S}} \subset \mathcal{S}$ of α scenarios $\widetilde{\mathcal{S}} = \{s_1, ..., s_\alpha\}$ generated from $\xi^{t+1..h}$, by using Monte Carlo sampling (Asmussen and Glynn, 2007).

Second, solving equation (10.3) basically involves solving to optimality problem $\mathcal{Q}$ for each possible action $x^t \in X^t$. Each problem $\mathcal{Q}$ involves solving a VRPTW for each possible scenario of $\widetilde{\mathcal{S}}$, while ensuring nonanticipativity constraints between the different solutions. As the VRPTW problem is an NP-hard problem, we propose to compute an upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$ based on a given sequence $x_R^{t+1..h}$ of future route actions. Because we impose the sequence $x_R^{t+1..h}$, the set of possible decisions at time t is limited to those directly compatible with it, denoted $\tilde{X}^t(x_R^{t+1..h}) \subseteq X^t$. That limitation enforces $r(x^{0..h}) < +\infty$. This finally leads to the GSA decision rule:

$$\textit{(GSA)} \qquad x^t = \underset{x^t \in \tilde{X}^t(x_R^{t+1..h})}{\arg\min} \ \overline{\mathcal{Q}}(x^{0..t}, x_R^{t+1..h}, \widetilde{\mathcal{S}}) \tag{10.7}$$

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

10.2.1 *Sequence $x_R^{t+1..h}$ of future route actions.*

This sequence is used to compute an upper bound of $\mathcal{Q}$. For each time $t < t' \leq h$, the route action $x_R^{t'}$ only contains operational decisions related to vehicle routing (*i.e.*, for each vehicle, travel towards a vertex, or wait at its current position). It does not contain decisions related to requests acceptation. The more flexible $x_R^{t'}$ with respect to $\widetilde{\mathcal{S}}$, the better the bound $\overline{\mathcal{Q}}$. We describe in Section 10.3 how a flexible sequence is computed through local search.

10.2.2 *Computation of an upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$.*

Algorithm 6 depicts the computation of an upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$ given a sequence $x_R^{t+1..h}$ of route actions consistent with past actions $x^{0..t}$. For each scenario s_i of $\widetilde{\mathcal{S}}$, Algorithm 6 builds a sequence $y^{0..h}$ for s_i, which starts with $x^{0..t}$, and whose tail $y^{t+1..h}$ is computed from $x_R^{t+1..h}$ in a greedy way. At each time $t < t' \leq h$, each request revealed at time t' in scenario s_i is accepted if it is possible to modify $y^{t'..h}$ so that one vehicle can service it; it is rejected otherwise. One can consider $y^{t'..h}$ as being a set of vehicle routes, each defined by a sequence of

Algorithm 6: The $\overline{\mathscr{Q}}(x^{0..t}, x_R^{t+1..h}, \widetilde{\mathcal{S}})$ approximation function

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

planned vertices. Each planned vertex comes with specific decisions: a waiting time and whether a service is performed. In this context, *trytoServe* performs a deterministic linear time modification of $y^{t'..h}$ such that (j, t') corresponds to the insertion of the vertex j in one of the routes defined by $y^{t'..h}$, at the best position with respect to VRPTW constraints and travel times, without modifying the order of the remaining vertices. At the end, Algorithm 6 returns the average number of rejected requests for all scenarios. Note that, when modifying a sequence of actions so that a request can be accepted (line 6), actions $y^{t'..h}$ can be modified, but $y^{0..t'-1}$ are not modified. This ensures that $\overline{\mathscr{Q}}$ preserves the nonanticipativity constraints. Indeed, the fact that two identical scenarios prefixes could be assigned two different subsequences of actions implies that either $trytoServe((j,t'), y^{t'..h})$ is able to modify an action $y^{t<t'}$ or is a nondeterministic function. In both cases, there is a contradiction. Finally, notice that contrary to other local search methods based on Monte Carlo simulation as in Ghiani et al., 2009; Schilde et al., 2011, GSA considers the whole timing horizon when evaluating a first-stage solution against a scenario.

10.2.3 *Comparison to MSA*

GSA has two major differences with MSA. Given a set of scenarios, GSA maintains only one solution, namely the sequence $x_R^{t+1..h}$, that best suits to a pool of scenarios whilst MSA computes a set of solutions, each specialized to one scenario from the pool. Furthermore, by preserving nonanticipativity GSA approximates the multistage problem of equations (10.1,10.3). In contrary, MSA relaxes these constraints and therefore approximates the two-stage problem (see Section 10.2 and Van Hentenryck et al., 2009).

In particular, given a pool of scenarios obtained by Monte Carlo sampling, MSA Expectation Algorithm 1 of Section 4.2 reformulates equation (10.2) as a *sample average approximation* (SAA, Ahmed and Shapiro, 2002; Verweij et al., 2003) problem. The SAA tackles each scenario as a separate deterministic problem. For a specific scenario $\xi^{t+1..h}$, it considers the recourse cost of a solution starting with actions $x^{0..t}$. Because the scenarios are not linked by nonanticipativity constraints, two scenarios ξ_i and ξ_j that share the same prefix $\xi^{t+1..t'}$ can actually be assigned two solutions performing completely different actions $x_i^{0..t'}$ and $x_j^{0..t'}$, for some $t' > t$. The evaluation of action x^t over the set of scenarios is therefore too optimistic, leading to a suboptimal choice. By definition, the *Regret* algorithm approximates the Expectation algorithm. The *Regret* algorithm then also approximates a two-stage problem. The *Consensus* algorithm selects the most suggested action among plans of the pool. By selecting the most frequent action in the pool, *Consensus* somehow encourages nonanticipation. However, the nonanticipativity constraints are not enforced as each scenario is solved separately. *Consensus* also approximates a two-stage problem.

10.3 SOLVING THE DYNAMIC AND STOCHASTIC VRPTW

GSA alone does not permit to solve a DS-VRPTW instance. In this section, we now show how the decision rule, as defined in equation 10.7, can be embedded in an online algorithm that solves the DS-VRPTW. Finally, we present the different waiting and relocation strategies we exploit, including a new waiting strategy.

10.3.1 *Embedding GSA*

In order to solve the DS-VRPTW, we design Algorithm 7, which embeds the GSA decision rule.

10.3.1.1 *Main Algorithm*

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

It runs in *real time*. It is started before the beginning of the time horizon, in order to compute an initial pool $\widetilde{\mathcal{S}}$ of α scenarios and an initial solution $x_R^{1..h}$ with respect to offline requests (revealed at time 0). It runs during the whole time horizon, and loops on lines 3 to 7. It is stopped when reaching the end of the time horizon. The *real time* is discretized in h time units, and the variable t represents the current time unit: It is incremented when real time exceeds the end

Algorithm 7: LS-based GSA

Initialize $\widetilde{\mathcal{S}}$ with α scenarios and compute initial solution $x_R^{1..h}$
$t \leftarrow 1$;
while *real time has not reached the end of the time horizon* **do**
    `/* Beginning of the time unit */`
    $(x^t,\ x_R^{t+1..h}) \leftarrow$ `handleRequests`$(x^{0..t-1},\ x_R^{t..h},\ \xi^t)$
    execute action x^t and update the pool $\widetilde{\mathcal{S}}$ of scenarios w.r.t. to ξ^t
    `/* Remaining of the time unit */`
    $(x_R^{t+1..h}) \leftarrow$ `optimize`$(x_R^{t+1..h})$
    $t \leftarrow t+1$ `/* Skip to next time unit */`

Function : `optimize`$(x_R^{t+1..h})$
while *real time has not reached the end of current time unit* t **do**
    $y_R^{t+1..h} \leftarrow$ `shakeSolution`$(x_R^{t+1..h})$
    if $\overline{\mathcal{Q}}(x^{0..t}, y_R^{t+1..h}, \widetilde{\mathcal{S}}) < \overline{\mathcal{Q}}(x^{0..t}, x_R^{t+1..h}, \widetilde{\mathcal{S}})$ **then**
        $x_R^{t+1..h} \leftarrow y_R^{t+1..h}$
    if $\beta =$ *number iterations since last re-initialization of* $\widetilde{\mathcal{S}}$ **then**
        Re-initialize the pool $\widetilde{\mathcal{S}}$ of scenarios w.r.t. $\xi^{t+1..h}$
return $x_R^{t+1..h}$

Function : `handleRequests`$(x^{0..t-1},\ x_R^{t..h},\ \xi^t)$
$y^{0..t-1} \leftarrow x^{0..t-1};\ y^{t..h} \leftarrow x_R^{t..h}$
for *each request revealed for a vertex* c *in realization* ξ^t **do**
    if *we find, in* $\leq \delta_{ins}$, *how to modify* $y^{t..h}$ *to serve request* (c,t) **then**
        modify $y^{t..h}$ to accept request (c,t)
    else
        modify $y^{t..h}$ to reject request (c,t)
return $(y^t,\ y^{t+1..h})$

of the t^{th} time unit. In order to be correct, Algorithm 7 requires the real computation time of lines 4-7 to be smaller than the real time spent in one time unit. This is achieved by choosing suitable values for parameters α and δ_{ins}.

Lines 4 and 5 describe what happens whenever the algorithm enters a new time unit: Function `handleRequests` (described below) chooses the next action x^t and updates $x_R^{t+1..h}$. Finally, $\widetilde{\mathcal{S}}$ is updated such that it stays coherent with respect to realization ξ^t. Each scenario $\xi^{t..h} \in S$ is composed of a sequence of sampled requests. To each customer region i is associated an upper bound $\overline{u}_i = \min(l_0 - t_{i,0} - s_i, l_i - t_{0,i})$ on the time unit at which a request can be revealed in that region, like in Bent and Van Hentenryck, 2004b. That constraint prevents tricky or inserviceable requests to be sampled. At time t, a sampled request (i, t) which doesn't appear in ξ^t is either removed if $t \geq \overline{u}_i$ or randomly delayed in $\xi^{t+1..h} \in S$ otherwise.

The algorithm spends the rest of the time unit into function `optimize`, iterating over lines 9-14, in order to improve the sequence of future route actions $x_R^{t+1..h}$. We consider a hill climbing strategy: the current solution $x_R^{t+1..h}$ is shaked to obtain a new candidate solution $y_R^{t+1..h}$, and if this solution leads to a better upper bound $\overline{\mathcal{Q}}$ of $\mathcal{Q}$, then it becomes the new current solution. Shaking is performed by the `shakeSolution` function. This function considers different neighborhoods, corresponding to the following move operators: relocate, swap, inverted 2-opt, and cross-exchange (see Kindervater and Savelsbergh, 1997; Taillard et al., 1997 for complete descriptions). As explained in Section 10.3.2, depending on the chosen waiting and relocation strategy, additional move operators are exploited. At each call to the `shakeSolution` function, the considered move operator is changed, such that the operators are equally selected one after another in the list. Every β iterations, the pool $\widetilde{\mathcal{S}}$ of scenarios is re-sampled (lines 13-14). This re-sampling introduces diversification as the upper bound computed by $\overline{\mathcal{Q}}$ changes. We therefore do not need any other meta-heuristic such as Simulated Annealing.

Note that a possible implementation of the `shakeSolution` function has been previously provided as part of Algorithm 5, which describe a variable neighborhood search in the context of the SS-VRPTW-CR.

10.3.1.2 *Function* `handleRequest`

Function `handleRequest` is called at the beginning of a new time unit t, to compute action x^t in light of online requests (if any). It implements the GSA decision rule defined in equation (10.7). The function considers each request revealed at time t for a vertex j, in a sequential way. For each request, it tries to insert it into the sequence $x_R^{t..h}$ (*i.e.*, modify the routes so that a vehicle visits j during its time window). As in `shakeSolution`, local search operations are performed during that computation. The time spent to find a feasible solution

including the new request is limited to δ_{ins}. If such a feasible solution is found, then the request is accepted, otherwise it is rejected. If there are several online requests for the same discretized time t, we process these requests in their real-time order of arrival, and we assume that all requests are revealed at different real times.

10.3.2 *Waiting and Relocation strategies*

As defined in Section 10.1, a vehicle that just visited a vertex usually has the choice between traveling right away to the next planned vertex or first waiting for some time at its current position. Unlike in the static (and deterministic) case, in the dynamic (and stochastic) VRPTW these choices may have a significant impact on the quality of the solution.

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

10.3.2.1 *Relocation strategies*

Studies Bertsimas and Ryzin, 1991 and Bertsimas et al., 1993 already showed that, for a dynamic VRP with no stochastic information, it is optimal to relocate the vehicle(s) either to the center (in case of single-vehicle) or to strategical points (multiple-vehicle case) of the service region. The idea evolved and has been successfully adapted to routing problems with customer stochastic information, in reoptimization approaches as well as sampling approaches.

*In a dynamic VRP, vehicles can be relocated to **strategic waiting locations**, even when they still have customers to visit (as long as they can wait!).*

Relocation strategies explore solutions obtained when allowing a vehicle to move towards a customer vertex even if there is no request received for that vertex at the current time slice. Doing so, one recognizes the fact that, in the context of dynamic and stochastic vehicle routing, a higher level of anticipation can be obtained by considering to reposition the vehicle after having serviced a request to a more stochastically fruitful location. Such a relocation strategy has already been applied to the DS-VRPTW in Bent and Van Hentenryck, 2007.

10.3.2.2 *Waiting strategies*

In a dynamic context, the planning of a vehicle usually contains more time than needed for traveling and servicing requests. When it finishes to service a request, a vehicle has the choice between waiting for some time at its location or leaving for the next planned vertex. A good strategy for deciding where and how long to wait can potentially help at anticipating future requests and hence increase the dynamic performances. We consider three existing waiting strategies and introduce a new one:

*A **waiting strategy** is a heuristic that automatically assigns waiting times, based on the sequences of vertices.*

- *Drive-First (DF)*: The basic strategy aims at leaving each serviced request as soon as possible, and possibly wait at the next vertex before servicing it if the vehicle arrives before its time window.
- *Wait-First (WF)*: Another classical waiting strategy consists in delaying as much as possible the service time of every planned requests, without violating their time windows. After having serviced a request, the vehicle hence waits as long as possible before moving to the next planned request.
- *Custom-Wait (CW)*: A more tailored waiting strategy aims at controlling the waiting time at each vertex, which becomes part of the online decisions.
- *Relocation-Only waiting (RO)*: In order to take maximum benefit of relocation strategy while avoiding the computational overhead due to additional decision variables involved in custom waiting, we introduce a new waiting strategy. It basically applies *drive-first* scheduling to every request and then applies *wait-first* waiting only to those requests that follow a relocation one. By doing so, a vehicle will try to arrive as soon as possible at a planned *relocation request*, and wait there as long as possible. In contrary, it will spend as less time as possible at non-relocation request vertices. Note that if it is not coupled to a relocation strategy, *RO* reduces to *DF*. Furthermore, *RO* also reduces to the dynamic waiting strategy described in Mitrović-Minić and Laporte, 2004 if we define the service zones as being delimited by relocation requests. However, our strategy differs by the fact that service zones in our approach are computed in light of stochastic information instead of geometrical considerations.

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

10.4 EXPERIMENTATIONS

We now describe our experimentations and compare our results with those of the state of the art MSA algorithm of Bent and Van Hentenryck, 2004b.

10.4.1 *Algorithms*

Different versions of Algorithm 7 have been experimentally assessed, depending on which waiting strategy is implemented and whether in addition we use the relocation strategy or not.

The *drive-first* waiting strategy, as well as its version including *relocation*, produced very bad results in comparison to other strategies, rejecting more than twice more online requests in average. Because of its computational overhead, the *custom-wait* strategy also produced bad results, even with relocation. For conciseness we therefore do not report these strategies in the result plots.

The 3 different versions of Algorithm 7 we thus consider are the following: GSA*wf*, which stands for GSA with *wait-first* waiting strategy, GSA*wfr* which stands for GSA with *wait-first* and *relocation* strategies, and finally GSA*ro* with means GSA using *relocation-only* strategy. Recall that, by definition, the *relocation-only* strategy involves relocation. In addition to those 3 algorithms, as a baseline we consider the GLS*wf* algorithm, which stands for *greedy local search* with *wait-first* waiting. This algorithm is similar to the dynamic LS described in Schilde et al., 2011, to which we coupled a Simulated Annealing metaheuristic. In this algorithm, stochastic information about future request is not taken into account and a neighboring solution is solely evaluated by its total travel cost.

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

SETUP. Computations are performed on a cluster composed of 32 processors 64-bits AMD Opteron(tm) 6284 SE cores, with CPU frequencies ranging from 1400 to 2600 MHz. Executables were developed with C++ and compiled on a Linux Red Hat environment with GCC 4.4.7.

10.4.2 *Benchmarks*

The selected benchmarks are borrowed from Bent and Van Hentenryck, 2004b which considers a set of benchmarks initially designed for the static and deterministic VRPTW in Solomon, 1987, each of these containing 100 customers. In our stochastic and dynamic context, each customer becomes a request region, where dynamic requests can occur during the online execution.

The original problems from Bent and Van Hentenryck, 2004b are divided into 4 classes of 15 instances. Each class is characterized by its degree of dynamism (DOD, the ratio of the number of dynamic requests revealed at time $t > 0$ over the number of a priori request known at time $t = 0$) and whether the dynamic requests are known

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

Figure 10.2: Summary of the test instances, grouped per degree of dynamism. Cells represent the probability that a request gets revealed during the time slice defined by each interval $[t, t']$.

early or lately along the online execution. The time horizon $H = 480$ is divided into 3 time slices. A request is said to be early if it is revealed during the first time slice $t \in [1, 160]$. A late request is revealed during the second time slice $t \in [161, 320]$. There is no request revealed during the third time slice $t \in [321, 480]$, but the vehicles can use it to perform customer operations.

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

In Bent and Van Hentenryck, 2007, a fifth class is proposed with a higher DOD of 81% in average. Unfortunately, we were not able to get those Class 5 instances. We complete these classes by providing a sixth class of instances, with DOD of 100%. Each instance hence contains no initial request, an early request with probability 0.3 and a late request with probability 0.7. Figure 10.2 summarizes the different instance classes.

We add a new instance class to the benchmark, with 100% dynamism.

10.4.3 *Results*

Average results over 10 runs are reported. In Bent and Van Hentenryck, 2004b, 25 minutes of offline computation are allocated to MSA, in order to decide the first online action at time $t = 1$. During online execution, each time unit within the time horizon was executed during 7.5 seconds by the simulation framework. In order to compensate the technology difference, we decided in this study to allow only 10 minutes of offline computation and 4 seconds of online computation per time unit. Thereafter, in order to highlight the contribution of the offline computation in our approach, the amount of time allowed at pre-computation is increased to 60 minutes, while each time unit still lasts 4 seconds. According to preliminary experiments, both the size

of the scenario pool and the resampling rate are set to $\alpha = \beta = 150$ for all our algorithms except GLS*wf*.

Figure 10.3 gives a graphical representation of our algorithms results, through performance profiles. Performance profiles provide, for each algorithm, a cumulative distribution of its performance compared to other algorithms. For a given algorithm, a point (x, y) on its curve means that, in $(100 \cdot y)\%$ of the instances, this algorithm performed at most x times worse than the best algorithm on each instance taken separately. Instances are grouped by DOD and by offline computation time. Classes 1, 2 and 3 have a DOD of 44%, hence they are grouped together. An algorithm is strictly better than another one if its curve stays above the other algorithm's curve. For example on the 60min plot of Class 6, GLS*wf* is the worst algorithm in 95% of Class 6 instances, outperforming GSA*wf* in the remaining 5% (but not the other algorithms). On the other hand, provided these 60 minutes of offline computation, GSA*ro* obtains the best results in 55% of the instances, whereas only 30% for GSA*wf* and GSA*wfr*. See Dolan and Moré, 2002 for a complete description of performance profiles.

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

We notice that the relocation strategy gets stronger as the offline computation time increases. This is due to the computational overhead induced by relocation vertices. GSA*wf* is then the good choice under limited offline computation time. However, both GSA*ro* and GSA*wfr* tend to outperform the other strategies when provided enough offline computation and high DOD.

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

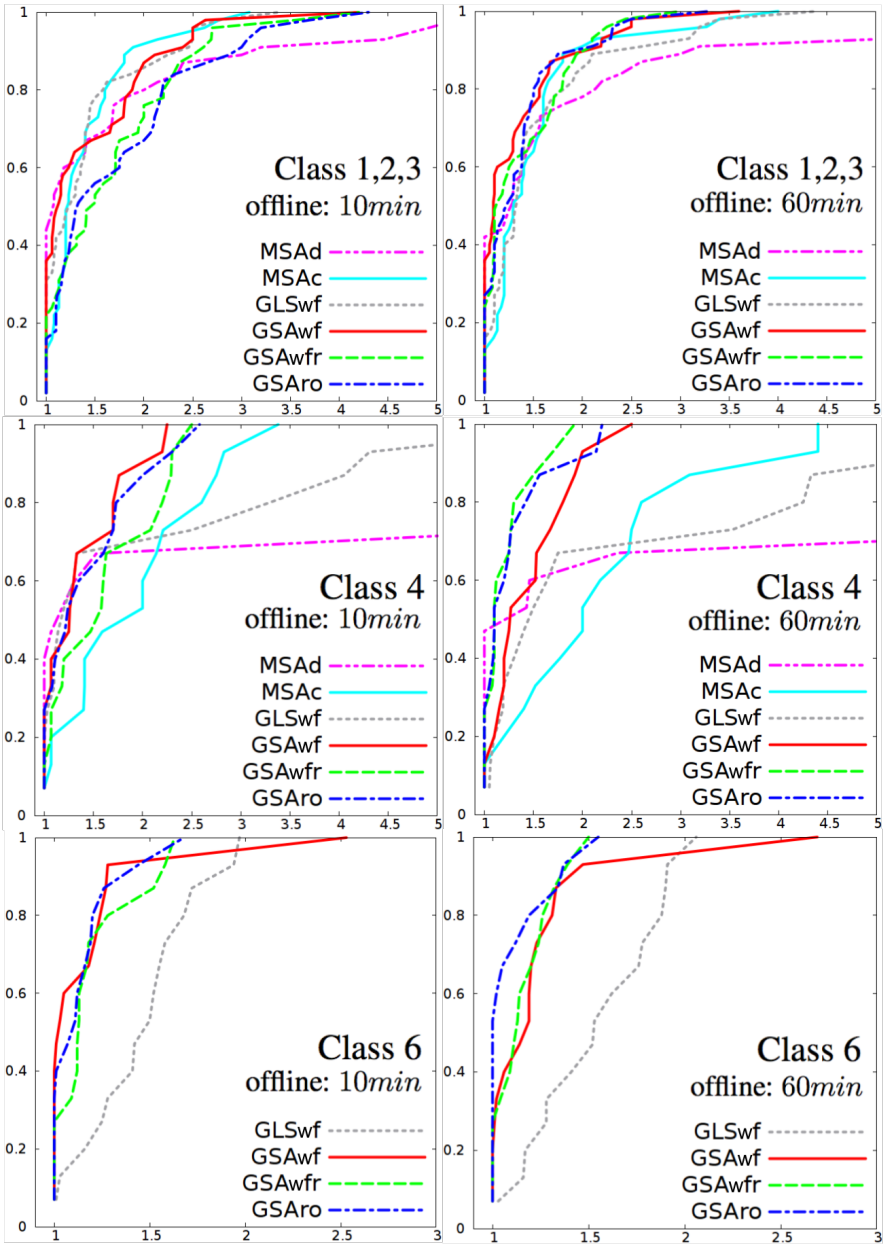


Figure 10.3: Performance profiles on classes 1 to 6. y-axis: proportion of instances; x-axis: average performance relatively to best result.

10.5 CONCLUSIONS

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

In a future study we plan to address a limitation of our solving algorithm which embeds GSA, namely the computational cost of its evaluation function. One possible direction would be to take more benefit of each evaluation, by spending much more computational effort in constructing neighboring solutions, e.g. by using Large Neighborhood Search (Shaw, 1998). Minimizing the operational cost, such as the total travel distance, is usually also important in stochastic VRPs. Studying the aftereffect when incorporating it as a second objective should be of worth. It is also necessary to consider other types of DS-VRPTW instances, such as problem sets closer to public or good transportation. Finally, the conclusions we made in Section 10.1 about the shortcoming of a two-stage formulation (illustrated in Figure 10.1) are theoretical only, and should be experimentally assessed.

11 HYBRID TWO-STAGE/MULTISTAGE APPROACH

As already pointed out by Saint-Guillain et al., 2017, a potential application of the SS-VRPTW-CR comes to online optimization problems, such as the Dynamic and Stochastic VRPTW (DS-VRPTW). Most of the approaches that have been proposed in order to solve the DS-VRPTW, including the approach we propose in Chapter 10, rely on Monte Carlo sampling. In fact, because perfect online reoptimization is intractable, heuristic methods are often preferred. Approaches based on sampling, such as Sample Average Approximation (Ahmed and Shapiro, 2002), are very common and consist in restricting the set of scenario to a randomly generated subset. Because the computed costs depend on the quality and size of the subset of sampled scenarios, they do not provide any guarantee. Thanks to recourse strategies, the expected cost of a first-stage SS-VRPTW-CR solution provides an upper bound on the expected cost under perfect reoptimization, as it also enforces the nonanticipativity constraints (see Section 2.5 and Saint-Guillain et al., 2015, for a description of these constraints).

In this chapter, we show how the SS-VRPTW-CR can be exploited at solving the DS-VRTPW, as an alternative to scenario sampling. In particular, by considering uncertainty in the online request reveal times, the SS-VRPTW-CR actually enforces the nonanticipativity property. This leads a hybrid two-stage variant of GSA (called 2*s-GSA*), where the SS-VRPTW-CR, namely the two-stage formulation of the DS-VRPTW, is exploited as an upper bound on the expected cost in the global multistage problem.

In order to solve the DS-VRPTW, **2*s-GSA*** *combines the two-stage SS-VRPTW-CR formulation with the GSA dynamic decision system.*

In Section 11.1, we summarize both the theoretical and empirical results obtained on the SS-VRPTW-CR, throughout Part ii of the thesis, in the context of online decision making. The hybrid algorithm is described in Section 11.2. Thereby, experiments are conducted on Section 11.3, comparing both standard and hybrid two-stage versions of GSA on a DS-VRPTW version of our Lyon benchmark. Finally, in Section 11.4 both GSA and 2s-GSA algorithms are applied to a dynamic, online version of our real-world police patrol management problem, previously introduced in Chapter 9.

11.1 LESSONS LEARNED FROM THE SS-VRPTW-CR

Let us consider again the DS-VRPTW, as formulated in the previous chapter, namely the multistage stochastic program (10.1):

$$\min_{x^t \in X^t} \mathrm{E}_{\xi^{t+1}} \left[\min_{x^{t+1} \in X^{t+1}} \mathrm{E}_{\xi^{t+2}} \left[\cdots \min_{x^{h-1} \in X^{h-1}} \mathrm{E}_{\xi^h} \left[\min_{x^h \in X^h} r(x^{0..h}) \right] \cdots \right] \right].$$

It can be reformulated as the deterministic equivalent program (10.3)

$$x^t = \arg\min_{x^t \in X^t} \mathcal{Q}(x^{0..t}, \boldsymbol{\xi}^{t+1..h})$$

where $\mathcal{Q}(x^{0..t}, \boldsymbol{\xi}^{t+1..h})$, the expected multistage value function, is further defined in (10.6).

The SS-VRPTW-CR recourse strategies approximate the DS-VRPTW objective function, while considerig the entire set of scenarios.

We saw in Chapter 5 that the recourse strategies we introduced for the SS-VRPTW-CR, such as $\mathcal{R}^q$, actually approximate the online problem, by providing an upper bound on the expected cost under perfect reoptimization. We also found in Chapters 8 and 9 that the latter strategy is easily adaptable, hence exploitable in various different contexts.

Whereas our standard GSA decision rule relies on an uppder bounding heuristic on a limited set of sampled scenarios, it now becomes clear that, in addition to provide a useful upper bound, our $\mathcal{R}^q$ strategy also benefits from the entire set of scenarios.

11.1.1 *Augmented first-stage SS-VRPTW-CR solutions*

Let $x_R^{t..h}$ be a sequence of future route actions, as defined in 10.2, including the current action candidate x^t. Recall that $x_R^{t..h}$ contains operational decisions for each vehicles (service a request, travel towards a vertex, wait at some position) from current time t to the end of the horizon. In particular, in addition to the visits at accepted requests, future route actions may contain relocation (and waiting) actions, thus involving waiting vertices.

Therefore, $x_R^{t..h}$ can be seen as an augmented SS-VRPTW-CR first stage solution. It is composed of waiting vertices, coupled with their respective waiting times τ. It is augmented with vertices that belong to the set of accepted requests that already appeared at time t. The expected number of rejected requests, from time $t+1$ to h, is then given by the random function $\mathcal{R}_A^q(x_R^{t..h}, \tau)$, with:

$$\forall x_R^{t..h} \ : \qquad \mathcal{Q}(x^{0..t}, \boldsymbol{\xi}^{t+1..h}) \leq \mathcal{Q}^{\mathcal{R}_A^q}(x_R^{t..h}, \tau).$$

The vertices corresponding to an accepted request, as well as past $x^{0..t-1}$ actions, are converted into fake waiting vertices which are not assigned any potential request, but whose the associated vehicle must visit at predefined moments. The computation of $\mathcal{Q}^{\mathcal{R}_A^q}(x_R^{t..h}, \tau)$ henceforth becomes identical to that of classical $\mathcal{Q}^{\mathcal{R}^q}$, for strategy $\mathcal{R}^q$, described in Section 5.2.2.

11.1.2 *Waiting times τ*

The methods used in order to determine the waiting times τ associated to each waiting location in $x_R^{t..h}$ naturally correspond to the four waiting

strategies already described in Section 10.3.2. In particular, the *Custom-Wait (CW)* waiting strategy, in which the waiting times are considered as decision variables, simply reduces to what have been done during all the previous thesis part (Chapters 5 to 9) in the context of the SS-VRPTW-CR.

Instead, another possibly interesting approach would be to determine the waiting times directly from the sequences of planned vertices. This corresponds to waiting strategies DF, WF and RO. Using *Drive-First (DF)* is however not suited here, since the request assignment heuristic of $\mathcal{R}^q$ would never assign potential requests to any waiting location. On the contrary, waiting strategies *Wait-First (WF)* and *Relocation-Only (RO)* will automatically assign non-zero waiting times to the planned waiting locations.

Relocation-Only *avoids assigning waiting times τ in a SS-VRPTW-CR solution.*

Relocation-Only particularly makes sense here, since we do not want fake vertices to be assigned waiting times. Instead, RO puts the highest possible waiting time to each planned waiting location. By avoiding the need for a specific assignment of the waiting times τ in a solution, *Relocation-Only* hence significantly simplifies the SS-VRPTW-CR solution space.

11.2 EMBEDDED TWO-STAGE ONLINE ALGORITHM: 2S-GSA

Algorithm 8 depicts the 2s-GSA hybrid method. It differs from Algorithm 7, GSA, in two points. First, the SS-VRPTW-CR random function $\mathcal{Q}^{\mathcal{R}^q_A}(x_R^{t..h}, \tau)$ replaces the approximation $\overline{\mathcal{Q}}(x^{0..t}, x_R^{t+1..h}, \widetilde{\mathcal{S}})$ in the evaluation of a solution. As a consequence, there is no scenario pool to manage anymore. Second, whereas GSA obtains its diversification with the periodic resampling of its scenario pool, 2s-GSA requires an explicit diversification mechanism. The acceptation rule of 2s-GSA (line 11) is based on a Simulated Annealing criterion, exactly as for Algorithm 5, which is actually simply embedded as a subroutine of Algorithm 8, function `optimize` (lines 8-13). To summarize, the 2s-GSA online algorithm only differs from GSA's algorithm by the implementation of the `optimize` function.

Note that the `shakeSolution` function is similar to that of Algorithm 7. It actually implements a basic variable neighborhood search, further described in Section 10.3.1.1. Furthermore, we assume `shakeSolution` to return a new augmented SS-VRPTW-CR solution $(y_R^{t+1..h}, \tau)$, thus representing a possible alternative sequence of future route actions. The waiting times τ, required by the $\mathcal{Q}^{\mathcal{R}^q_A}$ function, are hence deduced from $y_R^{t+1..h}$ when using *Relocation-Only* waiting strategy.

The `handleRequests` function is identical to that of standard GSA. It depends on a global parameter δ_{ins}, which limits the time limit to find a feasible current solution including the new online request, at line 17. Naturally, this local search phase is only performed whenever a feasible insertion position cannot be found for the request. However,

Algorithm 8: LS-based hybrid 2s-GSA

Compute initial solution $x_R^{1..h}$
$t \leftarrow 1$;
while *real time has not reached the end of the time horizon* **do**
    /* Beginning of the time unit */
    $(x^t,\ x_R^{t+1..h}) \leftarrow$ `handleRequests`$(x^{0..t-1},\ x_R^{t..h},\ \xi^t)$
    execute action x^t
    /* Remaining of the time unit */
    $(x_R^{t+1..h}) \leftarrow$ `optimize`$(x_R^{t+1..h})$
    $t \leftarrow t+1$ /* Skip to next time unit */

Function : `optimize`$(x_R^{t+1..h})$
while *real time has not reached the end of current time unit* t **do**
    $(y_R^{t+1..h},\ \tau) \leftarrow$ `shakeSolution`$(x_R^{t+1..h})$
    if *some acceptance criterion is met on* $\mathcal{Q}^{\mathcal{R}_A^q}(y_R^{t+1..h},\ \tau)$ **then**
        set $x_R^{t+1..h} \leftarrow y_R^{t+1..h}$
return *the best* $x_R^{t+1..h}$ *encountered during the search*

Function : `handleRequests`$(x^{0..t-1},\ x_R^{t..h},\ \xi^t)$
$y^{0..t-1} \leftarrow x^{0..t-1};\ y^{t..h} \leftarrow x_R^{t..h}$
for *each request revealed for a vertex* c *in realization* ξ^t **do**
    if *we find, in* $\leq \delta_{ins}$*, how to modify* $y^{t..h}$ *to serve request* (c,t) **then**
        modify $y^{t..h}$ to accept request (c,t)
    else
        modify $y^{t..h}$ to reject request (c,t)
return $(y^t,\ y^{t+1..h})$

because the only objective of the local search performed at line 17 is to restore feasibility after the new online request as been inserted, the sequence of future actions $x_R^{t+1..h}$ is likely to be severely modified. In particular, planned future waiting actions may be removed in order to make room for the new request, hence significantly impacting the expected quality of $x_R^{t+1..h}$ on the remaining horizon. Consequently, it is not clear in general whether setting $\delta_{ins} > 0$ will actually improve the average performances of the algorithm. The answer is most probably problem dependent.

11.3 EXPERIMENTS ON THE DS-VRPTW

Throughout this section, we compare the following three algorithms: GLS*wf*, GSA*ro* and 2s-GSA*ro*, that is, 2s-GSA where the waiting time are deduced by using *Relocation-Only* waiting strategy. Recall that in GLS*wf*, only the total travel cost (distance) is minimized, and stochastic information about future requests is not taken into account.

In Section 11.3.1 we first compare different parameters of the three algorithms on our less realistic benchmark, in order to determine the more promising configurations. Having selected our parameters, experiments are conducted by using the Lyon benchmark presented in Chapter 6, a significantly more realistic one, interpreted now as a DS-VRPTW benchmark instead of a SS-VRPTW-CR benchmark. Mainly for environmental reasons, we decided to limit the amount of experiments conducted throughout the section. In fact, testing in a dynamic context naturally requires a lot of computation time, as the realtime must be simulated through each experiment.

SETUP. Experiments have been done on a cluster composed of 64-bit AMD Opteron 1.4-GHz cores. The code is developed in *C++11* with *GCC4.9*, using *-O3* optimization flag. The current source code of our library for SS-VRPs and DS-VRPTs is available from the online repository: `bitbucket.org/mstguillain/vrplib`.

11.3.1 *Parameter estimation on artificial benchmark*

We first compare our three algorithms on the Bent and Van Hentenryck, 2007 benchmark, augmented with our sixth class. Recall that class 6 only contains online requests. Being highly artificial and, in our sense, highly unrealistic, we only exploit this benchmark in order to take a first glance at the best configurations for our various algorithms.

In order to determine whether using a positive δ_{ins} insertion delay globally improves the performances, we compare the three algorithms with $\delta_{ins} = 0$ and $\delta_{ins} = 5$ seconds. Furthermore, GSA*ro* is tested with two different configurations for its scenario pool: $\alpha = \beta = 150$ ($\text{GSA}_{\text{ro}}^{150}$), exactly as during the experiments of Section 10.4, and

	GLS*wf*		GSA_{ro}^{10}		GSA_{ro}^{500}		2s-GSA_{ro}^{1}	
δ_{ins}	0	5	0	5	0	5	0	5
Class 1	9.4	4.8	5.9	**1.2**	10.5	1.5	20.4	4.9
Class 2	11.5	7.6	5.3	**2.3**	8.5	3.3	19.0	6.8
Class 3	10.5	6.6	4.8	**1.9**	8.4	2.8	18.3	5.8
Class 4	15.5	12.0	4.9	**2.5**	7.7	4.6	21.6	9.7
Class 5	15.2	12.6	2.8	**1.6**	3.9	2.1	19.8	9.5
Class 6	24.1	19.7	14.3	**11.5**	16.2	12.8	30.8	17.9

Table 11.1: Average number of rejected requests on each instance class of the benchmark from Bent and Van Hentenryck, 2007, with algorithms GLS*wf*, GSA*ro* and 2s-GSA*ro*, depending on the δ_{ins} parameter. Offline computation time is set to 10 minutes. Online time units are each simulated during 30 seconds.

$\alpha = 500, \beta = 150$ (GSA_{ro}^{500}), in case the fact that the increased online computational time, now of 30 seconds per time units (instead of 4), would impact the performances of the α parameter. Finally, 2s-GSA*ro* is tested under scale 1 (2s-GSA_{ro}^{1}) for now. According to the results obtained in Section 9.3.5, the Simulated Annealing parameters of 2s-GSA are set to $T_{\text{init}} = 1$ and $f_T = 0.9$.

The offline computation time is set to 10 minutes. Each online time unit is simulated during 30 seconds. In all our benchmarks, time units are intended to last one minute each. Thus, using thirty seconds per time unit produces the results one can expect to obtain when using CPUs that are twice as slow than those used here. In fact, these thirty seconds are therefore still realistic. Recall that in Section 10.4, each time unit was simulated during only four seconds, in order to fairly compare with results obtained by another study. Every simulation is run 10 times on the corresponding benchmark instance. Only the average result is considered. For example, in Section 11.3.1 the same benchmark as in previous chapter is used. There are 15 instances in each class. Each cell of Table 11.1 is then an average computed over 10 $\times$ 15 instances, 150 runs.

The average results are shown in Table 11.1. We directly remark that setting $\delta_{ins} = 5$ increases the performances of all the algorithms. A bigger scenario pool that contains 500 samples does not increase the performances of GSA. Whereas the table shows average results that are aggregated for each class, it may be interesting to consider the results of each instance separately, as these may vary a lot within a particular instance class. Figure 11.1 (left) thus shows the performance profiles of these five algorithms, on the 15 $\times$ 6 = 90 instances. In fact, the curves confirm what we already observed in Table 11.1. The performances of 2s-GSA_{ro}^{1} are for now quite disappointing. We however observe that

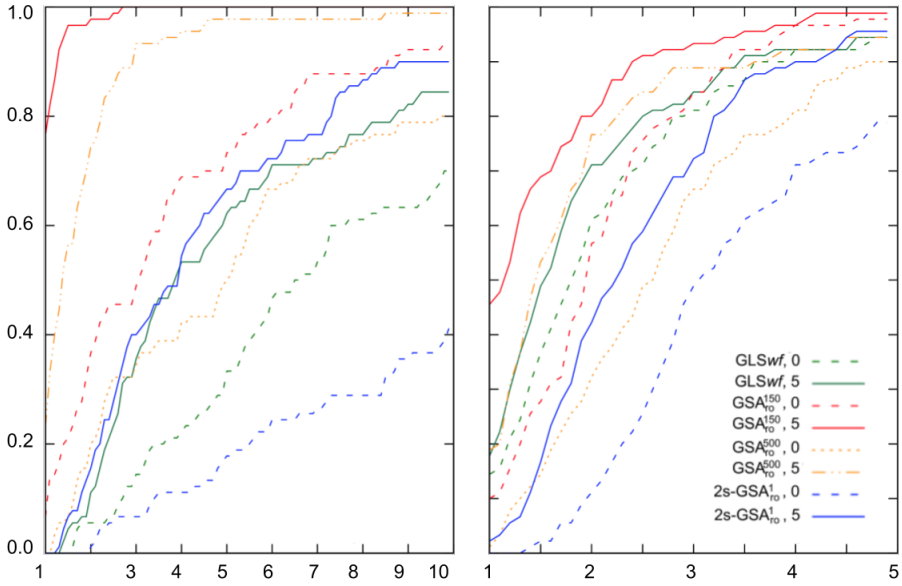


Figure 11.1: Performance profiles on all six classes of instances. Left: based on average number of rejected requests. Right: based on standard deviations.

the performance gap with the other algorithms tends to decrease has the degree of dynamism increases, and is minimal on Class 6.

Another potentially interesting performance indicator, when choosing an algorithm, may be its stability. One does probably not want an algorithm that performs very well one day and very poorly the day after, even though it performs quite well in average. Recall that for each algorithm, each of the 90 instances is experimented 10 times, in order to cope with the randomness within the optimization methods. We can therefore consider the associated standard deviations. On the right of Figure 11.1, we computed the performance profiles when we consider the standard deviations as performance indicator. It shows which are the algorithms that are the more likely to perform in a stable way. Again, 2s-GSA compares quite bad with GSA, and reveals to be unstable on these instances.

We observed bad performances, at least not really encouraging, of our hybrid algorithm 2s-GSA$^{1}_{ro}$ compared to our classical GSA. However, this benchmark is really particular, and we can hope for a better contribution when applied to more realistic contexts. In particular, this benchmark counts only three online time slots for the 240 time units, each time slot representing 80 time units. In comparison, time slots in the Lyon benchmark represent 5 time units each: one potential request per five minutes at each customer region.

11.3.2 *Results on Lyon benchmark*

We now conduct our experiments on a specific class of instances from the Lyon benchmark, presented in Chapter 6. We consider the set of 50cw-*i* instances, that is, involving 50 customer regions which can also act as waiting locations. Recall that these instances involves a horizon of 480 time units, divided in 96 time slots, each being associated to a potential request.

In order to be able to exploit the SS-VRPTW-CR benchmark in a DS-VRPTW context, we have to generate, based on the provided request probabilities, a limited number of reproducible scenarios to be simulated online. The existing data (graph information, travel times, request probabilities and attributes), along with these sampled scenarios, define a complete DS-VRTPW benchmark. For each 50cw-*i* instance, we generate five scenarios. We thus have now $15 \times 5 = 75$ DS-VRPTW instances. It is available at becool.info.ucl.ac.be/resources.

Because all the instances of the Lyon benchmark are 100% dynamic (*i.e.* no offline request), 2s-GSA may be handicaped by the *Relocation-Only* waiting strategy. In fact, whenever along a vehicle route two waiting vertices follow in a row, then RO will assign zero waiting time to the first waiting location in order to maximize the waiting time at the next one. Having many more online request vertices than waiting vertices, this is not a problem as is it unlikely to have two waiting vertices that follow. However, under 100% dynamism all the offline computation is done with waiting vertices only, whereas the requests gradually appear during the online phase. In order to take all the benefits from the SS-VRPTW-CR formulation, which aims at designing preventive vehicles routes composed of several waiting locations, it is interesting to test 2s-GSA with *Custom-Wait* (CW) waiting strategy. Indeed, CW allows 2s-GSA to choose itself appropriate waiting times at each planned waiting location. We hence add the 2s-GSA$^1_{\text{cw}}$ algorithm, standing for 2s-GSA with CW (waiting time multiples of 10 minutes) and under scale 1, as well as 2s-GSA$^2_{\text{cw}}$, its variant under scale 2.

The simulation parameters are now set to 10 minutes offline computation, with online time units of 20 seconds. Each instance is now simulated 5 times (instead of previously ten). The insertion delay δ_{ins} is still set to 5 seconds. The number of vehicles, which is not tied to the instances, is set to either 10 or 20.

Top of Figure 11.2 provides the performance profiles of each algorithm, for the 75 instances, computed on the average number of rejected requests (left of Figure 11.2) and the standard deviations (right), provided 10 vehicles. Clearly, GSA$^{150}_{\text{ro}}$ outperforms the four 2s-GSA variants. Amongst the latters, the *Relocation-Only* versions of 2s-GSA, namely 2s-GSA$^1_{\text{ro}}$ and 2s-GSA$^2_{\text{ro}}$, reveal to perform better than their *Custom-Wait* counterparts. We also remark that whereas GSA$^{150}_{\text{ro}}$

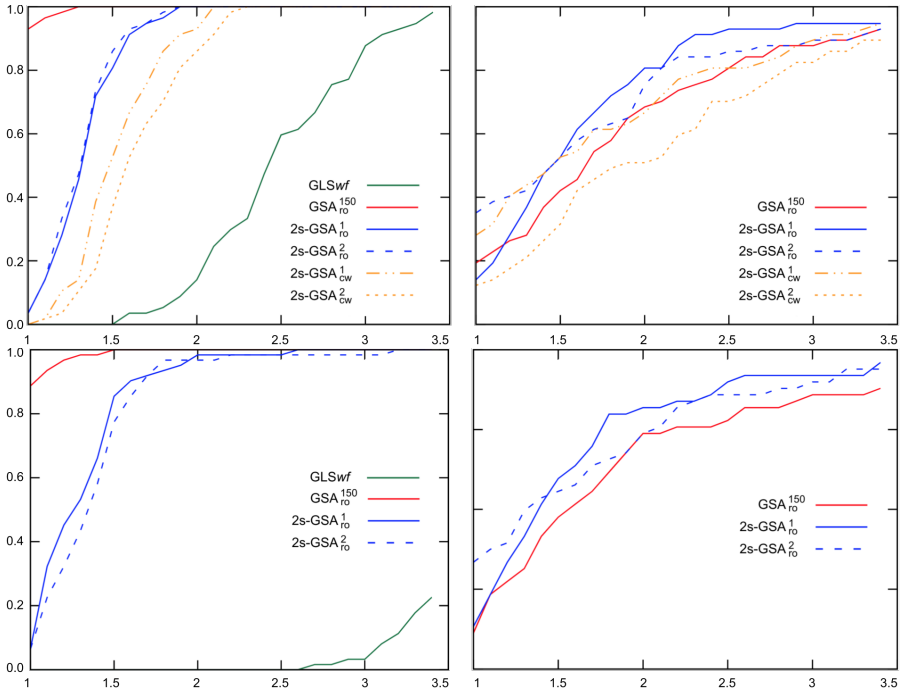


Figure 11.2: Performance profiles on the Lyon benchmark (75 instances), provided either 10 vehicles (up) or 20 vehicles (down). Left: based on average number of rejected requests. Right: based on standard deviations.

is more performant in average, 2s-GSA$^{1}_{ro}$ seems to be more stable according to the standard deviations of the average results. Note that we do not consider the standard deviations of GLS. In fact, the greedy local search method achieves a standard deviation of (or very close to) zero, for all instances. This is due to the absence of offline request and the limited amount of online requests (< 100). Since new online requests are gradually revealed as already accepted ones are being serviced, hence fixing part of the solution (nonanticipativity principle), GLS always finds the same global optimum (in terms of total travel distance), on every simulation of a given scenario.

On the bottom of Figure 11.2 are displayed the performance profiles when the number of vehicles is set to 20, for algorithms GLS*wf*, GSA$^{150}_{ro}$, 2s-GSA$^{1}_{ro}$ and 2s-GSA$^{2}_{ro}$ only. We observe that the performance gap between GSA and 2s-GSA does not significantly decrease as the number of vehicles increased. GSA$^{150}_{ro}$ remains more efficient, whereas 2s-GSA$^{1}_{ro}$ is still more stable. Naturally, the gap between GLS and the approaches based on stochastic modeling increases with the number of vehicles involved, as already observed all the other experimental contexts all along this thesis. In average, using 10 vehicles, GSA reduces the amount of rejected requests by 58%, when the average gains of 2s-GSA$^{1}_{ro}$ and 2s-GSA$^{2}_{ro}$ are of respectively 46% and 47%. When provided 20 vehicles, the average gains of the three algorithms increase to 75%, 69% and 68%.

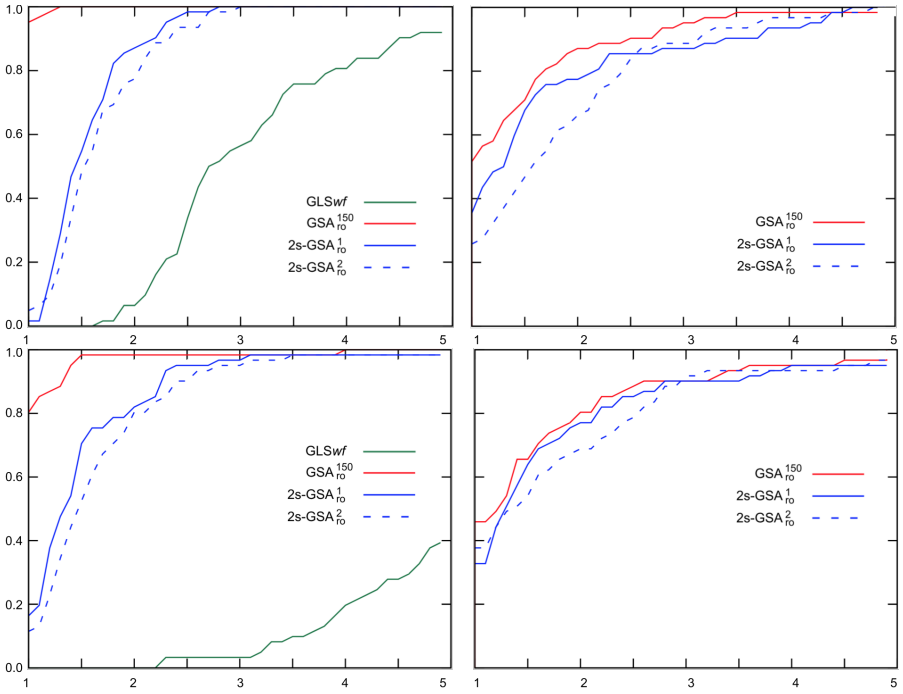


Figure 11.3: Performance profiles on the Lyon benchmark (75 instances), provided either 10 vehicles (up) or 20 vehicles (down), every request time window being doubled. Left: based on average number of rejected requests. Right: based on standard deviations.

INFLUENCE OF THE TIME WINDOWS. Figure 11.3 shows how the performance profiles evolve when the requests becomes less urgent, by multiplying the length of each time window by two, and given either 10 and 20 vehicles. Compared to GSA, the contribution of 2s-GSA does not improve in this context. Indeed, we saw in Section 8.3 that a SS-VRPTW-CR modeling becomes more suited as the requests are urgent, the problem being constrained to a high quality of service, represented by very short time windows. GSA now reduces the amount of rejected requests with 10 vehicles (*resp.* 20) by 65% (*resp.* 81%) in average. The average gains of 2s-GSA$^{1}_{ro}$ are of 49% (*resp.* 75%), and finally 47% (*resp.* 73%) for 2s-GSA$^{2}_{ro}$.

11.4 APPLICATION: DYNAMIC POLICE PATROL MANAGEMENT IN BRUSSELS

We now apply both GSA and 2s-GSA algorithms to a dynamic, online version of our real-world police patrol management problem, previously introduced in Chapter 9.

Each day of observation during 2017 provides us with a specific instance scenario.

Recall that the input data of the problem are based on observations made during real operations from 2013 to 2016. The data collected during 2017 are kept for validation, that is, the experimentations. It means that each operational day during the observed period, in 2017, actually stands for a scenario that can be experimentally simulated. After filter-

ing the observed days as explained in Section 9.2.3, it remains a total of 188 days, or scenarios, collected during 2017. These 188 scenarios will of course constitute our real-world DS-VRP-R benchmark.

11.4.1 *Problem definition: DS-VRP-R*

We note $X^t = X\left(x^{0..t-1}, \xi^{1..t}\right)$ the set of decisions that are valid at some time unit $t \in H_0$. The set X^t still depends on past decisions $x^{1..t-1}$ and revealed requests ξ^t. Unlike the DS-VRPTW, in the DS-VRP-R online requests are always accepted, and the objective here is to serve them as soon as possible. We still note $X^{t..t'}$ the sequence of sets $\langle X^t, \dots, X^{t'} \rangle$ where $0 \leq t \leq t' \leq h$. A each time t, given the sequence $x^{0..t-1}$ of past decisions, the best action x^t is then obtained by solving the following multistage stochastic program:

$$\min_{x^t \in X^t} \mathrm{E}_{\xi^{t+1}} \left[\min_{x^{t+1} \in X^{t+1}} \mathrm{E}_{\xi^{t+2}} \left[\cdots \min_{x^{h-1} \in X^{h-1}} \mathrm{E}_{\xi^h} \left[\min_{x^h \in X^h} \mathcal{Q}(x^{0..h}) \right] \cdots \right] \right] \tag{11.1}$$

where we define $\mathcal{Q}(x^{0..h})$ as the average delay on the interventions achieved by the final solution $x^{0..h}$.

In practice, GSA performs a Sample Average Approximation based on equation (9.5). 2s-GSA uses the bi-objective formulation (9.11), similarly to experiments conducted in Chapter 9.

11.4.2 *Experimental results*

We report the performance profiles of GLS*wf*, GSA*ro* and 2s-GSA*ro*, computed based on the average intervention delays achieved on each of the 188 instances. Each instance is simulated five times, with still 10 minutes of offline computation and 20 seconds per online time unit. Finally, according to the results from Section 9.3, both GSA and 2s-GSA are given 100 waiting locations amongst the 150 (customer) vertices of the discretized operational map.

This benchmark involves 360 time units, one per minute between 4am and 10am. Contrary to the Lyon benchmark (480 time units), time slots are here of 30 minutes, which may handicap 2s-GSA.

PRELIMINARY EXPERIMENTS. We first compare the following five algorithm on instances 1 to 50, provided 3 vehicles: GLS*wf*, $\text{GSA}^{150}_{\text{ro}}$, $\text{2s-GSA}^{1}_{\text{ro}}$, $\text{2s-GSA}^{2}_{\text{ro}}$ and $\text{2s-GSA}^{1}_{\text{cw}}$. The profiles are shown in Figure 11.4. We remark that both $\text{2s-GSA}^{2}_{\text{ro}}$ and $\text{2s-GSA}^{1}_{\text{cw}}$ perform worse than the other methods. Henceforth, we exclude these two algorithms from the remaining of the experiments.

OVERALL RESULTS: 3 VEHICLES. Contrary to the first 50 scenarios, GLS*wf* is now outperformed by the other approaches. Over the 188

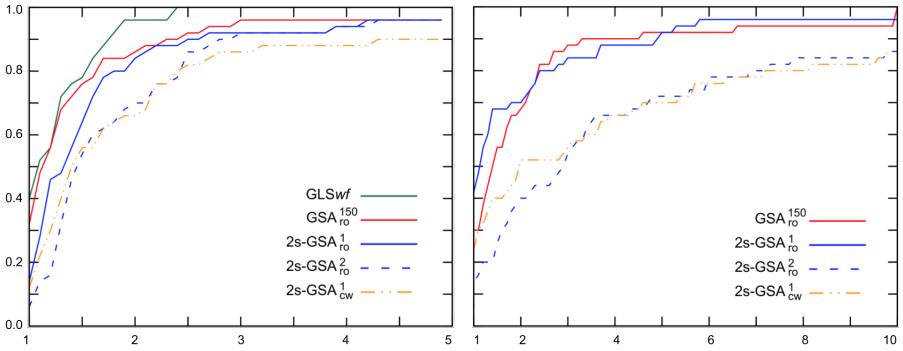


Figure 11.4: Performance profiles on the Police benchmark (first 50 scenarios), provided either 3 vehicles. Left: based on average intervention delays. Right: based on standard deviations.

scenarios, GLS*wf* achieves an average intervention delay of 12.4 minutes, GSA*ro* has an average of 10.5 minutes (avg. standard deviation: 2.6), and 2s-GSA*ro* obtains 12.5 minutes (avg. standard deviation: 2.6). GSA*ro* therefore achieves a relative gain of 15%, compared to GLS. Figure 11.5 (top) shows the associated performance profiles.

OVERALL RESULTS: 6 VEHICLES. Figure 11.5 (bottom) shows the associated performance profiles. Provided 6 vehicles, GLS*wf* has an average intervention delay of 12.3 minutes, thus not taking any advantage a larger fleet of vehicles. GSA*ro* reaches now an average of 7.7 minutes (average standard deviation: 1.9), thus improving GLS by 37%. Finally 2s-GSA*ro* obtains 12.4 minutes (average standard deviation: 2.8).

11.5 CONCLUSIONS

The hybrid algorithm 2s-GSA does not allow to beat the sampling based algorithm proposed in the previous chapter. Experiments conducted on three very different benchmarks show that the length of the time slots significantly impacts the performances of 2s-GSA. In fact, the best performances, compared to GSA, are obtained on the Lyon instance involving time slots of 5 time units only. Furthermore, on that benchmark 2s-GSA reveals to perform in a more stable way, in terms of the standard deviations over the multiple runs of each instance.

It should be noted that whereas the first two benchmarks are used to solve a DS-VRPTW, the third benchmark consider a DS-VRP-R case study. On this different online problem, 2s-GSA performs significantly worse than GSA, meaning that the underlying SS-VRPTW-CR embedded model is probably not that suited to the DS-VRP-R online problem.

Anyway, we saw that the sampling-based approach proposed by GSA is successfull on the Lyon benchmark, allowing to satisfy signifi-

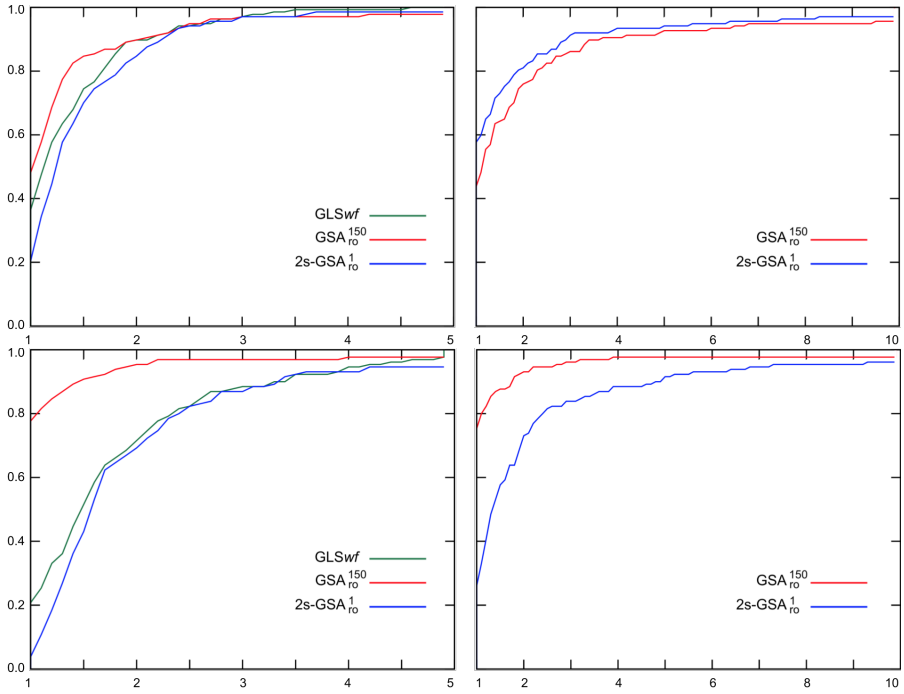


Figure 11.5: Performance profiles on the Police benchmark, provided either 3 vehicles (up) or 6 vehicles (down). Left: based on average intervention delays. Right: based on standard deviations.

cantly more online requests in average than the greedy local search baseline method considered. In fact, the contribution of GSA increased with the number of vehicles. Surprinsingly, it also increased as all the time windows were multiplied by two: from 58% to 65% with 10 vehicles. GSA also achieved to reduce by 37% the average intervention delays, in the context of our real-world online police patrol maganement problem.

CONCLUSION AND PERSPECTIVES

CONCLUSION

In this thesis, we explored two different, often complementary, approaches to time constrained online vehicle routing problems.

Optimizing a priori decisions: the SS-VRPTW-CR

This thesis introduces a new theoretical two-stage stochastic vehicle routing problem, the SS-VRPTW-CR, intended to represent a set of operational contexts that were not covered yet. It has the particularity to take the uncertainty on the request reveal times into account, hence providing the first two-stage stochastic VRP formulation that actually respects the nonanticipativity property inherent to all multistage stochastic VRPs. Two different recourse strategies for the SS-VRPTW-CR are proposed, for which the exact expected costs of the final solutions are computable in pseudo-polynomial time.

Not surprisingly, solving the SS-VRPTW-CR revealed computationally very hard, and the exact approaches we explored generally failed at solving even small instances. We hence proposed a local search method as well as a meta-heuristic, Progressive Focus Search (PFS), to efficiently solve the SS-VRPTW-CR problem. PFS, as well as our basic local search algorithm alone, showed to efficiently tackle larger problems, for which an exact approach is not possible, by performing remarkably well.

An interesting observation on the experimental results is that the benefit of using a SS-VRPTW-CR formulation actually increases with the number of vehicles, and the urgency of the requests. Also, our experiments indicated that allowing the vehicles to wait directly at potential customer vertices, when applicable, leads to better expected results than using separate relocation vertices. However, depending on the operational restrictions, this is not always possible.

We also proposed another two-stage stochastic vehicle routing problem, the SS-VRP-R, as a practical formulation of a real world problem, the management of police patrol units in Brussels. We showed the recourse strategies and closed-form expressions developed for the SS-VRPTW-CR can actually be adapted for the SS-VRP-R, in order to compute the expected average intervention delay. Experiments showed that a SS-VRP-R/SS-VRPTW-CR framework may reduce the average mean intervention delays by more than 30%, compared to a basic policy. This improvement is due to the preventive nature of the first-stage solutions, computed in light of stochastic knowledge, and which are composed of sequences of anticipative relocation instructions for the police patrols. Finally, we also described how the SS-VRPTW-CR equations can be easily adapted in order to take time dependency of the travel times into account, whenever it makes sense. However, the results obtained were quite surprising. By experimenting under either

constant or time-dependent travel times, it appeared that exploiting time-dependency is not interesting in the context of the aforementioned case-study. This contradicts the current trend of considering time-dependency, especially in urban contexts, as critical. Besides the increased computational costs, the main reason is probably that most of the trips in the city are optimized to be very short length, whereas our measurements showed that the shorter a vehicle trip, the less the travel time will be impacted by time-dependency.

Optimizing online decisions: the DS-VRPTW

In many operational contexts, being able to react fast and efficiently to online events is of critical importance. Unlike two-stage formulations, the multistage online VRP formulation involves non-trivial online decisions, which must be computed in light of the available stochastic knowledge. However, such computation is actually inherently (strongly) hard, motivating the need for heuristic methods.

We hence proposed GSA, a decision rule for dynamic and stochastic vehicle routing with time windows (DS-VRPTW), based on a stochastic programming heuristic approach. Unlike existing methods, our method does not require to relax the nonanticipativity constraints. Instead, our method is proven to approximate a multistage stochastic problem, through Monte Carlo sampling, but using a heuristic evaluation function that preserves the nonanticipativity constraints. We also introduced a new waiting strategy, *relocation-only*, aiming at taking better benefit of the preventive vehicle relocation actions.

A possible limitation of GSA may be its scenario pool, necessarily limited in the number of samples it contains, due to computational constraints. We proposed to create a two-stage/multistage hybrid version of GSA, called 2s-GSA, where the scenario pool, used to compute an average cost, is totally replaced by a expectation calculus over the SS-VRPTW-CR formulation we propose. We hence tested 2s-GSA on various DS-VRTPW benchmarks, as well as our DS-VRP-R real-world application, the online police patrol management in Brussels. It appeared that the performances of 2s-GSA vary greatly depending on the benchmark, whereas GSA always severely outperforms our best greedy local search (GLS) baseline method. The more realistic is the benchmark, the more efficient becomes 2s-GSA, compared with GSA (and GLS). Yet, GSA remains the best alternative for the considered benchmarks.

Our literature review, in Chapter 4, highlights that the City VRP contributions contain very few real-world based applications and the results are normally based on academic (and unrealistic) datasets. Moreover, these applications lack a global vision and they are scarcely repeatable in a different context. A lack emerging from the literature is the identification of the main sources types, how to mix and how to

interface them with one or more simulation and optimization modules in order to give flexible solutions to the stakeholders and the users. In this thesis, we provide three additional realworld benchmarks for the DS-VRPTW. The proposed benchmarks are expressed in a convenient format, also introduced during the thesis, making the various parts of the instance data more flexible and easily reusable in different operational contexts.

PERSPECTIVES

There are still a number of research avenues, and potentially room for significant improvement, in the application of probabilistic models to the DS-VRP framework.

First, we pointed out in previous observations how the estimation of the quality of a solution depends on the how well the recourse strategy used fits the operational context. These should be further studied. Some SS/DS-VRP's could reveal particularly suited to such approach. Applications to other operational problems, similar to VRPs, such as the robust operations management problem studied in Appendix A, may result be successful results.

Generally speaking, whereas most of the online operational problems are strongly NP-hard, the complexity of the proposed recourse strategies can be an additional indicator of the computational complexity of the problem in practice; when comparing two online problems, the complexity of the closest (still pseudo-polynomial[1]) recourse strategies developed could allow to further classify the problems. When talking about the DS-VRPTW, and more specifically, its two-stage counterpart the SS-VRPTW-CR, we saw that the expected cost of best suited recourse strategy is computable in $\mathcal{O}(n^2h^3Q)$, whereas it is of $\mathcal{O}(nh^2)$ for a stochastic single-machine scheduling problem with no minimum transition times (see Appendix A). Interestingly, it is exactly the same complexity than that of the stochastic VRP with random demands, introduced in Bertsimas, 1992. These two problems are in fact very similar, and consequently their recourse strategies, which we further showed in Section 5.2.3 to be subproblems of the SS-VRPTW-CR.

Second, this thesis focused on local search methods. Although simple to implement and easily adaptable, such algorithms could be outperformed by other approches, already known to perform particularly well on vehicle routing problems. In particular set partitioning methods such as column generation, which becomes commonly used for stochastic VRPs, could provide interesting results. Further research should also be considered for alternative computational models such

1 An important theoretical question remains: how intelligent, that is, how close to the associated online problem could a recourse strategy be, such that its expected cost stays efficiently computable?

as Bayesian networks (Darwiche, 2009), which naturally apply to the random objective functions represented by the recourse strategies. Approximate Dynamic Programming (ADP, Powell, 2009) is also widely used to solve routing problems in presence of uncertainty. Combined with the scaling techniques, ADP is likely to provide interesting results. Alternative representations of the uncertainty, e.g. by using fuzzy numbers instead of random variables (Huang and Teghem, 2012), should also be considered.

FINAL WORDS

There are still a lot of potential improvements in order to obtain an efficient dynamic decision system for online routing problems such as the DS-VRPTW or the DS-VRP-R. In this thesis, we focused on the application of local search algorithms to anticipative methods. The inherent complexity of DS-VRPs makes difficult the use of stochastic formulations. With a lot a work, we showed that it is in fact possible to develop and compute efficiently closed-form expressions to evaluate the expected quality of a solution. However, they do not suffice at outperforming classical approaches, based on Monte Carlo sampling. The simplicity of designing sampling-based evaluation algorithms specific to particular operational contexts tends to privilegiate the latters over complicated probabilistic models. Furthermore, whereas independence between the potential requests must have been assumed in order to simplify our closed-form formulae, this issue can however be easily solved with sampling. Albeit the SS-VRPTW-CR model embeded in 2s-GSA is by definition perfectly suited within a purely static and stochastic VRP framework, we must afterall admit that the famous *Occam's razor* principle stands still in the context of the online VRPs studied in this thesis.

Yet, studying DS-VRP's while looking at either their two-stage or multistage stochastic formulations is definitly of worth, and permits to emphasys important properties of the online problems. When possible, especially under simple operational contexts and when the system must provide strong guarantees, probabilistic models may be preferred over sampling approaches. From a theoretical point of view, in often leads in interesting, sometimes useful, results.

Acknowledgments

Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region. Christine Solnon is supported by the LABEX IMU (ANR-10-LABX-0088) of Université de Lyon, within the program "Investissements d' Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Part IV

APPENDIX

A

ROBUST OPERATIONS MANAGEMENT ON MARS

Unlike most classical scheduling problems, operations in a space mission must be planned days ahead. Complex decision chains and communication delays prevent schedules from being arbitrarily modified, hence online reoptimization approaches are usually not appropriate. The problem of scheduling a set of operations in a constrained context such as the *Mars Desert Research Station* (MDRS, Figure A.1) is not trivial, even in its classical deterministic version. It should be seen as a generalization of the well-known NP-complete *job-shop scheduling problem* (Lenstra and Kan, 1979), which has the reputation of being one of the most computationally demanding (Applegate and Cook, 1991). Hall and Magazine, 1994 insist on the importance of mission planning, as 25% of the budget of a space mission may be spent in making these decisions beforehand. They further cite the Voyager 2 space probe for which the development of the a priori schedule involving around 175 experiments requiring 30 people during six months. Nowadays, hardware and techniques have evolved and it is likely that a couple of super-equipped (*i.e.* with a brand new laptop) human brains may suffice in that specific case. Yet, the problems and requirements have evolved too. Instead of the single machine Voyager 2, space missions have to deal with teams of astronauts.

At the MDRS, computing an optimal schedule becomes significantly less attractive as problem data, such as the manipulation time of experiments, are different from their predicted values. In a constrained environment with shared resources and devices, such deviations can propagate to the remaining operations, eventually leading to global infeasibility. Even provided only one non-human operator, uncertainty may be of significant impact. For instance, the future M2020 planetary rover will be equipped with an onboard scheduler, designed to operate under processing time uncertainty Chi et al., 2019; Rabideau and Benowitz, 2017. The purpose of this paper is to investigate, based on the real case study of a Mars analog mission, the impact of stochastic robust modeling against a classical deterministic approach on the reliability of a priori mission planning.

APPLICATION AND CONTRIBUTIONS. How does the ABCD problem, introduced in Section 2.4.3 of the thesis, applies to realistic case studies? How to deal with processing time uncertainty when facing a larger, complex, scheduling problem which possibly involves multiple operators, unknown probability distributions and exotic constraints? In this paper, we show how even such simple insights can actually

Figure A.1: The Mars Desert Research Station (MDRS) in the Utah desert, U.S, is a Mars analog planetary habitat.

be applied to real project development and planning. As a proof of concept, we demonstrate the contribution of robust modeling on the preparation and implementation of the *UCL to Mars 2018* project that took place at the Mars Research Desert Station (Utah), March'18.

Project development realizes about 30% of the world gross product (Turner et al., 2010). Besides the space analog mission, our approach naturally applies to a larger set of project scheduling contexts whereas most of existing studies have solely been done in machine scheduling environments (Herroelen and Leus, 2005).

We formulate the core problem as a *robust single-machine scheduling problem with random processing times*. Tasks are subject to precedence constraints, strict time windows and minimum transition times. We show how the robustness of a solution, in terms of its probability to remain feasible to operational time constraints, can be exactly computed in pseudo-polynomial time, when relying on realistic recourse assumptions. By removing the recourse assumption and thus allowing online reoptimization, it still provides a useful valid lower bound on the solution's robustness. We also identify theoretical limitations to computational tractability and propose alternatives. Finally, we adapt the core problem to the goals pursued and constraints faced during the *UCL to Mars 2018* mission, and empirically measure the average gain of using our generalized stochastic formulation instead of a deterministic one. In addition to the proposed closed-form formula, computational results are compared with those obtained when using Sample Average Approximation method to estimate the solutions' robustness.

ORGANISATION. Section A.1 provides a general formal description of the problem. In section A.2, we show how the robustness of a solution, in terms of probability to remain feasible under uncertain processing times, can be efficiently computed. In fact, we provide closed-form expressions to compute it in pseudo-polynomial time. Finally, section A.3 applies theory to the practical case study of the

UCLtoMars2018 analog mission. Conclusion and further research avenues are discussed in section A.4.

A.1 CORE PROBLEM

For now we consider a *robust single-machine scheduling problem with random processing times* (R-SMS-T), involving job precedence, time windows and minimum transition times constraints. Approximation functions will thereafter be used to generalize the core theoretical results to the specific problem we face in our MDRS case-study, a *robust job-shop scheduling problem* involving additional specific constraints. A recent review on stochastic scheduling is provided by Chaari et al., 2014.

INPUT. A discrete time horizon $H = \{0, \dots, h\}$ on which a set $J = \{j_1, \dots, j_n\}$ of jobs must be scheduled on an unique machine (or operator). The machine processes one job at a time and each job must be processed exactly once. Each job j comes with a probability p_j^d that j requires a processing time $d \in H$, with $\sum_{d \in H} p_j^d = 1$. Each job j is associated a set $TW_j \subseteq H$ of valid time intervals to process j, called time windows. Precedence constraints state that a (possibly empty) set $J_j^< \subset J$ of jobs must be completed before starting job j. A *minimum transition time* states a minimum delay $w_{j',j}^{\min}$ to be observed between completion of a job $j' \in J_j^<$ and the beginning of j.

SOLUTION AND FORMULATION. A solution to the R-SMS-T is an ordered sequence of jobs: $s = \langle j_i, \dots, j_{i'} \rangle$. Every job $j \in J$ appears exactly once in the sequence. We use a two-index flow formulation in order to describe s: binary decision variables x_{ij} denote whether or not the job $j \in J$ is scheduled immediately after $i \in J$. Additional variables x_{0j} (*resp.* x_{j0}), for $j \in J$ refer to the first (*resp.* last) job j of the sequence.We formulate our R-SMS-T as:

$$\max_s \quad r(s) \tag{A.1}$$

$$\text{s.t.} \sum_{i \in J_0 \setminus \{j\}} x_{ij} = \sum_{i \in J_0 \setminus \{j\}} x_{ji} = 1 \qquad j \in J \tag{A.2}$$

$$u_j - u_i + n x_{ji} \leq n - 1 \qquad j, i \in J \tag{A.3}$$

$$u_i \leq u_j - 1 \qquad j \in J, i \in J_j^< \tag{A.4}$$

$$x_{ji} \in \{0, 1\} \qquad j, i \in J_0 \tag{A.5}$$

where we note $J_0 = J \cup 0$ for conciseness. The robustness measure $r(s)$, detailed in the next section, gives the probability of s to remain feasible. Flow conservation constraints (A.2) state that a job is preceded (and followed) by exactly one job i. In fact, inequalities (A.2)-(A.3) define the solution space of a *directed traveling salesman problem* (TSP), when using a so-called *MTZ-formulation* (Miller et al., 1960) in order to explicitly

formulate as u_j (A.3) the position of job j in the sequence. We then express precedence constraints quite naturally in (A.4). Since the time dimension is stochastic, time windows and minimum transition time constraints cannot be part of the description of a solution to the R-SMS-T. Instead, they contribute to objective function $r(s)$ as described in the next section. Constraints (A.2)-(A.5) are then sufficient to define the solution space of our robust single-machine scheduling problem with random processing times.

RECOURSE ASSUMPTIONS. In order to be efficiently computed, $r(s)$ requires assumptions (so-called *recourse strategy*) on how s is adapted to the realizations of the random processing times:

1. The machine (operator) executes its jobs according to the ordered sequence defined by s ;
2. When starting a job j, one does not know its processing time until it is actually completed.
3. A job that is not completed by the end of its current time window must be re-processed from scratch at the beginning of its next time window, if any (otherwise 4).
4. In case the machine fails at processing a job due to unfortunate processing times, the sequence is interrupted.

These assumptions directly come from the definition of our problem. In particular, by fixing the sequences of s assumption 1 explicitly forbids reoptimization. A relaxation would lead to a dynamic and stochastic problem, and would no longer permit the exact evaluation of $r(s)$ in pseudo-polynomial time (unless P=NP). Furthermore, operational contexts such as space missions do not allow the modification of the schedule in the middle of a work day. Our recourse strategy still provides a good indicator for situations that suggest reoptimization at the end of each work day. In fact, under assumptions 1-4 the probability to remain feasible as computed by $r(s)$ is a lower bound to the probability of perfect reoptimization (optimally reoptimizating each time a random event realizes) to remain feasible.

A.2 ROBUSTNESS OF A SOLUTION

We define the robustness $r(s)$ of a given solution s as its probability to remain feasible. This can be trivially expressed in terms of the set of possible scenario realizations:

$$r(s) = \sum_{\xi \in E} \Pr\{\xi\} f(s, \xi) \tag{A.6}$$

where E is the set of all probable scenarios and $f(s, \xi)$ is the indicator function returning 1 if and only if solution s remains feasible under

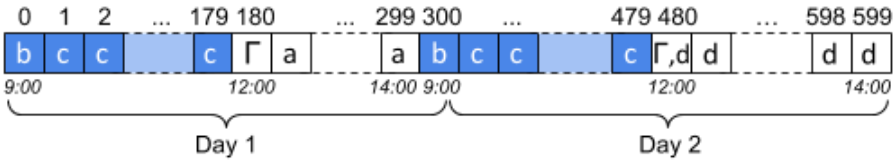


Figure A.2: Time horizon of job B. Blue cells represent $TW_B = \{0,\ldots,179,300,\ldots,479\}$. When $t <= 179$ then $TW_B^-(t) = \emptyset$. For $180 \leq t \leq 479$, we have $TW_B^-(t) = \{0,\ldots,179\}$ and, for $t \geq 480$, $TW_B^- = \{300,\ldots,479\}$.

scenario ξ, that is, the case described by assumption 4. is not encountered. Since the size of E grows exponentially with respect to the number of jobs, the computation of (A.6) rapidly becomes intractable. The most natural approximation for such an enumerative function is called *Sample Average Approximation* (SAA, Ahmed and Shapiro, 2002), which relies on Monte Carlo sampling to evaluate only a subset of E. Its accuracy however depends on the number of samples taken into consideration.

Instead, one can reason on the jobs themselves (and their associated random variables) in order to derive a tractable formula to compute $r(s)$ exactly. Let us consider that a job j is *correctly processed* in a scenario ξ if it is completed within one of its valid time windows and fulfills all minimum transition time constraints with other jobs, if any. It follows that the probability that a solution s remains feasible is the probability that every job succeeds in that sense,

$$\begin{aligned} r(s) &= \Pr\{\bigwedge_{j\in J} \text{job } j \text{ correctly processed in } s\} \\ &= \Pr\{j_{\text{last}} \text{ correctly processed in } s\} \end{aligned} \tag{A.7}$$

which, by following assumption 4., is also the probability (A.7) that we succeed at proceeding the last job $j_{\text{last}} \in J$ of the sequence $s = \langle j,\ldots,j_{\text{last}}\rangle$.

In our ABCD example, $r(s)$ is the probability that job D eventually gets completed. Our time horizon is composed by two consecutive five-hour work days, modeled as a discrete set $H = \{0,\ldots,599\}$ which contains 600 time units of one minute each. The first 300 time units belong to the first work day, and so on. Since job B can only be processed during the first three hours of each day, we have $TW_B = \{0,\ldots,179,300,\ldots,479\}$. Similarly, $TW_C = \{0,\ldots,299\}$ as job C must be completed the first day. We also denote by $TW_j^-(t) \subseteq TW_j$ the set of time units which only belong to the time window that directly precedes time t and does not contain it. Figure A.2 illustrates H according to job B. We note $j^- \in J$ the job that directly precedes j in solution s.

Let $P_j^{\text{end}}(t)$ the probability that job $j \in J$ is completed at time t exactly, and $P_j^{\text{start}}(t)$ the probability that j is started at time t precisely.

Note that, consequently to assumptions 2. and 3., starting a job does not systematically involve its completion. Job j_{last} is therefore correctly processed if and only if it is completed during one of its time windows:

$$r(s) = \sum_{t \in TW_j} P_{j_{\text{last}}}^{\text{end}}(t) \tag{A.8}$$

COMPLETION TIMES. For every job $j \in J$, the probability $P_j^{\text{end}}(t)$ that j finishes exactly at discrete time unit $t \in H$ is constituted of all the possible processing times that could have led to complete the job at that specific time t:

$$P_j^{\text{end}}(t) = \begin{cases} \sum_{d=0}^{t} p_j^d \cdot P_j^{\text{start}}(t-d) & \text{if } t \in TW_j \\ 0 & \text{otherwise.} \end{cases} \tag{A.9}$$

STARTING TIMES. We use starting times to handle time windows. Suppose we know the probability $P_j^{\text{ready}}(t')$ that the machine becomes ready for job j at time $t' \leq t$, in the sense that it is ready as soon as it completes the previous job j^-, and that any minimum transition time constraint between a job j' and j is fulfilled. Then, $P_j^{\text{start}}(t)$ is the probability that, according to P_j^{ready} and j's time windows, one actually starts to process job j at a current time t. We decline the computation of $P_j^{\text{start}}(t)$ in three different cases, depending on t:

- $t \in TW_j \wedge t-1 \notin TW_j$: t is the first time unit of the current time window. For job B, this corresponds to b cells in Figure A.2. There are two possible reasons for starting a job at such particular moments:
  - Previous job, if any, just completed or did earlier. j must wait for current time window to begin (first summation term below). In the case of job B, at $t = 300$ the first summation ranges from $\Gamma_B(300) = 180$ to 300.
  - j had to be reprocessed (second summation term below), as last attempt (during previous time window TW_j^-) revealed to require too much time to complete and had to be interrupted due to assumption 4. In the case of job B, at $t = 300$ the second summation ranges in $\{(t', d) : 0 \leq t' \leq 179, t' + d > 180\}$.

  Putting the two cases together then leads to:

$$P_j^{\text{start}}(t) = \sum_{t'=\Gamma_j(t)}^{t} P_j^{\text{ready}}(t') + \sum_{\substack{t' \in TW_j^-(t) \\ d: \\ t'+d-1 \notin TW_j}} P_j^{\text{ready}}(t')\ p_j^d \tag{A.10}$$

  where $\Gamma_j(t)$ is the first time unit that directly follows the previous time window (from time t) that is, the first moment from which

we could wait for the opening of the current time window. For job B, this corresponds to the Γ cells in Figure A.2: $\Gamma_B(0) = 0$ and $\Gamma_B(300) = 180$.

- $t \in TW_j \wedge t-1 \in TW_j$: t lies on a legal time unit, but not the first of the current time window. The only reason for starting the job at that moment is that the machine just becomes available (ready) at current time t:

$$P_j^{\text{start}}(t) = P_j^{\text{ready}}(t) \tag{A.11}$$

  For job B, this corresponds to c cells in Figure A.2.

- $t \notin TW_j$: t is not a legal time unit for processing j,

$$P_j^{\text{start}}(t) = 0. \tag{A.12}$$

  For job B, this corresponds to cells a and d in Figure A.2. Note that the schedule definitely fails at processing j if there is no future legal time unit. This happens in Figure A.2 if the machine becomes available at d cells.

AVAILABILITY TIMES. We define $P_j^{\text{ready}}(t)$ as the probability that the machine becomes available for job j at a time t. It is the probability that job j could be started at time t, regarding *both* the completion time of j^- *and* minimum transition time constraints only (*i.e.* regardless time windows of j).
First job of a sequence. If job j is the first of the sequence s then the moment at which the machine becomes available is obviously time unit zero:

$$P_j^{\text{ready}}(t) = \begin{cases} 1 & \text{if} \quad t = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{A.13}$$

Subsequent jobs. We now consider a job j which is not the first of its sequence. If there is no minimum transition time constraint associated from any job j' to j, then the machine is ready for job j as soon as previous job j^- is completed:

$$P_j^{\text{ready}}(t) = P_{j^-}^{\text{end}}(t) \tag{A.14}$$

If j has exactly one minimum transfer time constraint with an unique job $j' \in J_j^<$, probability $P_j^{\text{ready}}(t)$ becomes

$$P_j^{\text{ready}}(t) \equiv \Pr\{t = \max(\text{end}(j^-), \text{end}(j') + w_{j',j}^{\min})\} \tag{A.15}$$

where $\text{end}(j)$ is the time at which job j reveals to be completed. Namely, either job j' has completed for long enough (at a time

$t' \leq t - w^{\min}_{j',j}$) to not worry about minimum transition time $w^{\min}_{j',j}$ and the operator waits for the completion of j^- in order to start j, or the previous job j^- is completed yet but j must be delayed until current time t coincides with the completion time of j' plus minimum transition time $w^{\min}_{j',j}$. If the sequence is $s = \langle A, B, C, D \rangle$ in our example, $P^{\text{ready}}_C(t)$ is the probability for t to be exactly equal to the maximum of 1) the completion time of previous job B and 2) that of A plus one hour. Mathematically,

$$\begin{aligned} P^{\text{ready}}_j(t) = \; & \Pr\{\text{end}(j^-) = t \wedge \text{end}(j') \leq t - w^{\min}_{j',j}\} \\ & + \Pr\{\text{end}(j^-) < t \wedge \text{end}(j') = t - w^{\min}_{j',j}\} \end{aligned} \tag{A.16}$$

Indeed, in absence of time windows for j either we start j as soon as j^- finishes (first term of (A.16)), or we wait after the completion j^- until we reach appropriate time $\text{end}(j') + w^{\min}_{j',j}$ (second term of (A.16)). Since the completion time of j^- clearly depends on that of j', it finally leads to:

$$\begin{aligned} P^{\text{ready}}_j(t) = & \sum_{t' \leq t - w^{\min}_{j',j}} P^{\text{end}}_{j'}(t') \cdot P^{\text{end}}_{j^-|j'}(t, t') \\ & + \sum_{t' < t} P^{\text{end}}_{j'}(t - w^{\min}_{j',j}) \cdot P^{\text{end}}_{j^-|j'}(t', t - w^{\min}_{j',j}) \end{aligned} \tag{A.17}$$

where $P^{\text{end}}_{j^-|j'}(t, t')$, the probability previous job j^- completes at time t conditionally that job j' completes at time $t' \leq t$, can be computed recursively by following (A.9) where $P^{\text{end}}_{j'}(t') = 1$ and $P^{\text{end}}_{j'}(t'') = 0$, for $t'' \neq t'$. In particular, in case $j' = j^-$ then (A.17) reduces to $P^{\text{end}}_{j^-}(t - w^{\min}_{j',j})$, since $P^{\text{end}}_{j^-|j^-}(t, t') = 1$ if $t' = t$, zero otherwise. Note that supplementary minimum transition time constraints can be associated to j by adding terms to the max operator in (A.15).

Intractability issue. Having two jobs j', j'' on which j depends for minimum transition time, one has to take conditional probabilities $P^{\text{end}}_{j^-|j',j''}(t, t', t'')$ into account for every combination of $t', t'' \in H$. In the case the last job j_n of a solution $s = \langle j_1, \ldots, j_n \rangle$ is constrained by minimum transition times involving previous jobs $j_1, \ldots, j_{n-1}$, computing all the conditional probabilities is equivalent to enumerating all the $\mathcal{O}(h^n)$ possible scenarios. In order to keep the computation tractable, in what follows we assume that there is at most one such constraint per job.

COMPUTATIONAL COMPLEXITY. Provided a solution to n jobs, the complexity of computing (A.7) is equivalent to the one of filling up three matrices P^{ready}, P^{start} and P^{end}, each of size nh, containing respectively all the $P^{\text{ready}}_j(t)$, $P^{\text{start}}_j(t)$ and $P^{\text{end}}_j(t)$ probabilities. The computational effort required to compute each cell significantly varies depending on the presence of minimum transition time constraints.

No minimum transition times. Once the probabilities in cells $(j, 1 \cdots t)$ of P^{end} are known, the cell $P^{\text{ready}}_{(j,t)}$ can be computed in $\mathcal{O}(1)$ using (A.14). Then, using probabilities in cells $(j, 1 \cdots t)$ of P^{ready}, cell $P^{\text{start}}_{(j,t)}$ can be computed in $\mathcal{O}(h)$ according to equation (A.10)-(A.12). In fact, the double summation in the second term of (A.10) is amortized in $\mathcal{O}(h)$. Finally, probabilities $(j, 1 \cdots t)$ of P^{start} allows to compute cell $P^{\text{end}}_{(j,t)}$ in $\mathcal{O}(h)$ according to equation (A.9). A solution consisting in n jobs and a time horizon of length h leads to a worst case complexity of $\mathcal{O}(nh^2)$.

Minimum transition times. In the case where each job can have a most one minimum transition time with another job, we consider the worst case involving a sequence $\langle j_1, \ldots, j_n \rangle$ in which all $j_2, \ldots, j_n$ jobs have a minimum transition time constraint with j_1. It then requires to compute $\mathcal{O}(n^2)$ conditional probability matrices $P^{\text{end}}_{j|j'}(t, t')$ each of size h^2, each cell still computable in $\mathcal{O}(h)$. That enables the computation of P^{ready} values in $\mathcal{O}(h)$ according to equation (A.17). The overall complexity is now of $\mathcal{O}(nh^2 + n^2h^3)$. However, if we allow a job to have minimum transition time constraints with q other jobs, then the $P^{\text{end}}_{j|j'_1 \cdots j'_q}$ matrix will be of exponential size h^q.

STOCHASTIC TRANSITION TIMES. There are some contexts in which transition times could also be considered stochastic. In fact, for a fixed planning, transitions may be equivalently seen as jobs. Stochastic transition times may be thus trivially handled, by simply replacing in a planning each stochastic transition time by a job with same probability distribution. Hence we equivalently end up with sequences of jobs, but no transition times.

A.3 THE *ucl to mars 2018* CASE STUDY

During our stay at the Mars Desert Research Station (MDRS, figure A.1), our crew conducted 10 experiments (see `http://ucltomars.org/#!/crews/190/projects`) regrouped in 7 different research projects. Each researcher from the team was associated to a project, composed of a set of jobs. Each researcher acted as an operator, or machine, each being subject to constraints such as those covered by the R-SMS-T plus a few additional ones. The goal of the crew's executive officer was therefore to model and solve the associated scheduling problem globally.

A.3.1 *In-place operations*

The a priori horizon covered the thirteen 9-hour work days of the entire mission. At the end of each day, we solved an updated schedule on the remaining horizon, depending on the scientific outcomes of the

current day. For practical reasons, each researcher was only able either to perform his/her own jobs, or to assist other researchers.

DETERMINISTIC MODEL. As we were provided 7 operators instead of a single one, the problem we faced at MDRS had been modeled in-place as a deterministic *job-shop scheduling problem* (JSP), assuming processing times to be perfectly known. The modeling of all crew members' research projects merged within a global problem was inevitable, as researchers at MDRS depend on limited and shared resources. Some projects required *extra vehicular activities* (EVAs), which for security reasons require between three and five participants. EVAs usually take half a day (four hours) in total, should be planned and approved days ahead and happen at most once a day. In such context, all crew members have their schedule linked to each others even concerning research projects that do not involve EVAs.

The time horizon consists of 10 minutes time units, each operational day counting 24×6 time units. In fact, because of transition time constraints the 15 non-working hours (out of 24) must be considered as well. An example of project model is depicted in figure A.3. The model requires three distinct EVAs and combines two different projects lead by the same researcher. Because a minimum number of people is required to validate an EVA, it is however likely that the researcher attends additional EVAs. In fact, the model depicted is actually connected to the projects lead by the seven other researchers thorough these EVA jobs.

In addition to precedence and minimum transition time constraints, we also notice a 24h *maximum transition time* between completion *Treatment* and that of *Exposition2*. Maximum transition time constraints are easily handled in the deterministic problem. The stochastic formulation however requires to be adapted, as explained hereafter.

Finally, beyond feasibility the model maximizes a quality measure, depending on several preferences predefined by the crew members, such as maximizing the delay between second and third EVA in figure A.3. We refer to this solution quality function as f^{mdrs} hereafter.

SOLUTIONS. Optimizing the deterministic problem was achieved in-place using a basic local search (LS) approach, exploiting well-known sequence neighborhood operators and a simulated annealing meta-heuristic. Our LS algorithm is directly adapted from the one for stochastic VRPs of Saint-Guillain et al., 2017 , while replacing the vertices, service durations and vehicles by the jobs, processing times and machines, respectively. On Mars, a solar day lasts approximately 24h39m and is commonly called a *sol*. We refer to the day preceding the first sol of the mission as SOL0. The nth day of the mission is called SOLn. Figure A.4 shows an example of a schedule as recomputed during the mission. Naturally, in practice schedules recomputed later

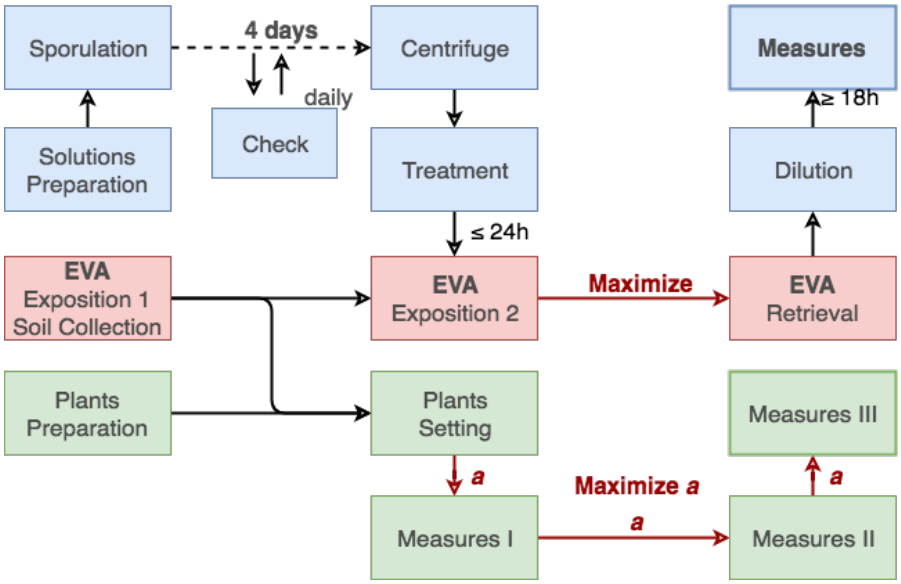


Figure A.3: Modeling of a research project combining botanics (green) and biology (blue), conducted by crew member Frédéric Peyrusson (UCLouvain, Belgium).

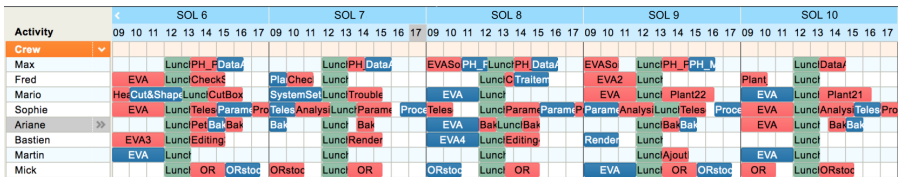


Figure A.4: Overview of the first six days of the global schedule as recomputed in-place on March 15th evening (SOL5, remaining horizon of 8 sols), based on updated data after completing five operational days.

than SOL5 differ as a new problem is reformulated at the end of each day, sometimes by integrating new specific constraints or preferences. Taking into account the initial complexity of the model, this motivated the choice for a heuristic algorithm over an exact approach.

The problem counts from 237 jobs at SOL0, to 149 jobs at SOL7. Time horizon, at SOL0, is composed of 13 days of 144 time units, each of 10 minutes.

A.3.2 *A posteriori analysis: stochastic robust approach*

This section investigates the impact of taking uncertainty into account at planning phase of a Mars analog mission, provided the robustness measure we propose.

A.3.2.1 *Solution method and solution evaluation.*

In order to study the impact of our robust formulation, independently of the technology used, we use the same solution algorithm as used for the deterministic problem.

The objective function computed by the solver at each solution s is replaced by the heuristic:

$$f(s) = f^{\text{mdrs}}(s) \times r^{\text{mdrs}}(s)$$

where $f^{\text{mdrs}}(s)$ is the quality of s as computed in the deterministic context. If the solution is not *deterministic-feasible*, that is if s does not fulfill all the constraints of the problem as described in section A.3.1 (including constraints which are specific to the MDRS, such as minimum and maximum people attendance during EVAs), then $f^{\text{mdrs}}(s) = 0$. In other words, we optimize based on the deterministic solution quality, processing times hence being assumed to be fixed to their expected values, multiplied by the robustness measure of the solution against processing time variability.

MDRS SPECIFIC CONSTRAINTS AND ISSUES. Note that following (A.8) our definition of $r(s)$ does not involve the *multiple parallel machines* context nor the *maximum transition time* constraints, whereas they were both present on projects we conducted at MDRS.

In case there is no minimum transition time constraint between jobs of a set of machines M, as it case the case at MDRS, our definition of $r(s)$ can be easily generalized:

$$r(s) = \prod_{m \in M} \Big(\sum_{t \in TW_j} P^{\text{end}}_{j^m_{\text{last}}}(t) \Big) \tag{A.18}$$

where j^m_{last} is the last job assigned by machine $m \in M$, since the execution of the jobs (and hence the $P^{\text{end}}_{j^m_{\text{last}}}(t)$ probabilities) are consequently independent between the different machines. This is true even for

EVAs, despite their minimum and maximum people requirement. For practical reasons, an EVA could only take place during mornings, from 9am to 12am. Together with the lunch activities, EVAs are the only jobs having a fixed deterministic duration (3h). The probability of respecting the scheduled people attendance to an EVA is then equivalent to the probability for all the assigned people to respect this 9am-12am time window.

Similarly to the minimum transition time constraints, maximum transitions are computationally very hard to take into account while computing an exact $r(s)$. Instead, here we use the following approximation:

$$r^{\text{mdrs}}(s) \approx \prod_{m \in M} \Big(\sum_{t \in TW_j} P^{\text{end}}_{j^m_{\text{last}}}(t) \Big) \times \Big[1 - \sum_{j \in J} \sum_{j' \in J_j^<} \Big(\sum_{t' \in H} \sum_{t \geq t' + w^{\max}_{j',j}} P^{\text{end}}_{j'}(t') P^{\text{end}}_{j}(t) \Big) \Big] \tag{A.19}$$

, by supposing independence between completion times. Here $w^{\max}_{j',j}$ is the maximum allowed transition time between completion of j' and that of j, if any, otherwise $w^{\max}_{j',j} = h$. The additional second term is one minus (an upper bound on) the approximated probability that a maximum transition time constraints is violated.

HORIZON PARTITIONING APPROXIMATION (hpa). Another issue with the MDRS case study is the computational effort required to compute $r(s)$, which critically depends on the length of the horizon. In practice, computing $r^{\text{mdrs}}(s)$ on the entire horizon takes at least several minutes, even at SOL7. Instead, we introduce the $r^{\text{mdrs}}_{\Delta}(s)$ measure, which we denote by *Horizon Partitioning Approximation*. HPA consists in the robustness of s when the horizon is partitioned in independent parts of Δ days each. For instance, $r^{\text{mdrs}}_{2}(s)$ is computed by first applying $r^{\text{mdrs}}(s)$ on days $\{1, 2\}$ only, using a two-days horizon and considering only the jobs a priori planned at days $\{1, 2\}$ by s. The same computation is then performed on days $\{3, 4\}$, by considering the days $\{1, 2\}$ to be fixed, and so on until we cover the entire horizon. A pass at days $\{d, \ldots, d + \Delta - 1\}$ computes the probability that, if everything happens as planned by s up to start of day d, all the jobs planned for days $\{d, \ldots, d + \Delta - 1\}$ get actually completed during those days. A job at day d that has a transition time constraint with a job j' from a day $d' < d$ simply sees its time window modified accordingly, since completion time of j' is supposed to be fixed. Finally, we multiply the probabilities obtained at each pass to get $r^{\text{mdrs}}_{\Delta}(s)$. Using $r^{\text{mdrs}}_{1}(s)$ then provides the probability that every job gets completed the day it is planned, which is a pessimistic approximation of the robustness of s, since in general delaying a some job during a few days does not necessarily break feasibility. The later can be viewed as optimizing a *stability* criterion (Goren and Sabuncuoglu, 2008), in the

sense that using $\Delta = 1$ we approximate the expected deviation from daily objectives.

A.3.2.2 *Experimental plan.*

Our empirical study is based on real data from the *UCL to Mars 2018* analog mission. Our benchmark is composed of the sequence of updated problems we faced at SOL1, SOL2, SOL4, SOL5, SOL7, in addition to the initial problem modeled at SOL0. Unfortunately, models faced at sols 3, 6, 8-12 were lost due to a technical issue.

However, we ignore the exact probabilities that describe the processing times of our jobs. Good predictions require a significant amount of observations, whereas we are only provided the scenario that realized during the mission. Collecting a sufficiently large set of observations is often impossible in practice and it is often both necessary and realistic to consider the real distributions as unknown (or hidden). Let X_j be the real probability distribution of j's processing time, and $\hat{X}_j$ the predicted one used by $r(s)$. As X_j is unknown, we approximate it using a normal distribution: $\hat{X}_j \sim N(\hat{\mu}_j, \hat{\sigma})$. We consider five different experimental contexts, depending on the quality of the approximations:

1. We know exactly the mean value of each distribution: $E[X_j] = \hat{\mu}_j, \forall j \in J$;
2. The approximations are fairly good: $E[X_j] \sim Un(\hat{\mu}_j \pm 10\%), \forall j \in J$;
3. The approximations are of poor quality: $E[X_j] \sim Un(\hat{\mu}_j \pm 30\%), \forall j \in J$
4. The approximations are globally underestimating: $E[X_j] \sim Un([\hat{\mu}_j - 10\%, \hat{\mu}_j + 30\%]), \forall j \in J$
5. The approximations are globally overestimating: $E[X_j] \sim Un([\hat{\mu}_j - 30\%, \hat{\mu}_j + 10\%]), \forall j \in J$

whereas in all five cases each hidden discrete distributions X_j is randomly generated according to $E[X_j]$, by using a normal distribution with 50 rolls only which results in a highly imperfect normal distribution. Note that 4. stands for a pessimistic context in which processing times will often reveal to be longer than initially estimated, in contrary to optimistic context 5.

A solution s is optimized by using either the deterministic objective function, $f = f^{\text{mdrs}}$, or the proposed stochastic measure, $f = f^{\text{mdrs}} \times r_{\Delta}^{\text{mdrs}}$, exploiting the provided $\hat{X}_j$ approximate distributions. In order to assess the quality of s, we measure its true robustness by simulating its execution on a sufficiently large number of scenarios (10^5), hence using *Sample Average Approximation* (SAA, Ahmed and Shapiro, 2002).

	1. Exact mean values			2. $\hat{\mu}\pm10\%$			3. $\hat{\mu}\pm30\%$			4. Pessimistic			5. Optimistic			
$f=$		$f^{\text{mdrs}}\times$			$f^{\text{mdrs}}\times$			$f^{\text{mdrs}}\times$			$f^{\text{mdrs}}\times$			$f^{\text{mdrs}}\times$		
	f^{mdrs}	r_1^{mdrs}	r_2^{mdrs}	f^{mdrs}	r_1^{mdrs}	r_2^{mdrs}	f^{mdrs}	r_1^{mdrs}	r_2^{mdrs}	f^{mdrs}	r_1^{mdrs}	r_2^{mdrs}	f^{mdrs}	r_1^{mdrs}	r_2^{mdrs}	δ
SOL0	0.7	84.0	79.5	0.7	74.9	67.0	0.1	63.7	54.5	0.1	5.5	24.0	31.8	92.9	94.4	7
SOL1	6.1	90.2	96.8	0.7	75.5	87.0	0.1	55.6	66.7	0.1	8.0	15.3	20.3	93.3	97.3	4
SOL2	1.4	97.5	96.3	4.0	93.6	90.3	0.1	45.7	40.3	0.1	57.7	44.5	15.5	99.1	97.5	3
SOL4	28.3	90.0	80.9	25.4	89.6	79.5	21.1	66.1	40.3	0.3	59.9	36.2	71.4	86.5	75.3	8
SOL5	2.9	89.6	88.7	0.5	91.7	91.0	2.9	99.6	99.5	0.1	0.6	1.9	41.4	99.9	99.9	3
SOL7	4.8	91.6	89.3	0.1	64.3	60.1	9.7	94.3	90.9	0.1	5.4	5.3	22.5	99.2	99.0	17
Avg.	*7.4*	*90.5*	*88.6*	*4.9*	*81.6*	*79.6*	*5.6*	*70.8*	*65.4*	*0.1*	*22.8*	*19.5*	*33.8*	*95.1*	*93.9*	7

Table A.1: Average percentage of times the solutions remain feasible despite processing time uncertainty, depending on the quality of available a priori stochastic knowledge. Column δ gives the average loss percentage of deterministic quality f^{mdrs}.

For each experimental context, from 1. to 5., the 10^5 scenarios are randomly sampled from the X_j hidden real distributions. We then measure the average proportion of the scenarios in which s remains feasible. We refer to this robustness measure as $\text{SAA}_{10^5}(s)$.

A.3.2.3 *Results.*

For each instance SOLn, we compute a set of 10 solutions by first using the deterministic objective function $f = f^{\text{mdrs}}$, then by using $f = f^{\text{mdrs}} \times r_1^{\text{mdrs}}$ and finally by using $f = f^{\text{mdrs}} \times r_2^{\text{mdrs}}$, with 30 minutes of computation time. Table A.1 shows the average results obtained by these sets of solutions as *the percentage of simulations in which the schedule remains feasible*, depending on the instance and the experimental context 1. to 5. We obtain these results by computing $\text{SAA}_{10^5}(s)$. For each instance, the δ column gives the solutions average relative difference in their deterministic attractiveness, as computed by f^{mdrs}.

Clearly, the solutions obtained using the two stochastic robust approaches strongly outperform those of the deterministic model. Obviously, stochastic models perform better under optimistic assumptions, namely exact (1.) and overestimating (5.) distributions. The largest gaps in the average robustness of both deterministic and stochastic models appear under context 1., since at that point the stochastic HPA functions r_1^{mdrs} and r_2^{mdrs} are computing values being almost equivalent to the real ones. The most preferred situation is naturally context 5. in which the mean processing times are globally overestimated, since whatever the solution is, it is likely to be more robust than under other contexts. However, it is interesting to note that even under such fortunate conditions, the deterministic model produces solutions that fail $\pm 66\%$ of the time, on average, against $\pm 5\%$ for those obtained using the proposed stochastic model. Furthermore, this improved robustness comes at the price of deteriorating by only 7% of the solution's deterministic quality (δ column) on average. Under contexts 2. and 3., average robustness is increased from the deterministic approach by more or less 76% and 65% respectively, when using r_1^{mdrs}. Finally, using $\Delta = 1$ compared to $\Delta = 2$ in r_Δ^{mdrs} reveals to be more advantageous here, although this is strongly problem dependent. In fact, in our MDRS case study involving time horizons that span from 6 to 13 days, Δ provides a parameterizable trade-off between accuracy and computational effort.

A.3.2.4 *Comparison with SAA.*

Results reported in Table A.1 clearly motivate the use of a robust formulation. However, the design and implementation of r^{mdrs} and r_Δ^{mdrs} can only be justified if it allows better average results compared to the Sample Average Approximation (SAA) method.

$f =$		1.	2.	3.	4.	5.	δ
$f^{\text{mdrs}} \times r_1^{\text{mdrs}}$		*90.5*	*81.6*	*70.8*	*22.8*	*95.2*	*7*
$f^{\text{mdrs}} \times$	SAA_{10^2}	89.4	78.6	47.3	8.3	96.6	15
	$\text{SAA}_{5.10^2}$	90.6	79.8	49.7	16.7	97.6	14
	SAA_{10^3}	89.7	78.3	56.2	19.4	97.1	12
	$\text{SAA}_{5.10^3}$	88.8	79.3	62.3	21.6	98.2	11
	SAA_{10^4}	87.0	76.1	55.7	19.3	96.8	14

Table A.2: Average results obtained by using the SAA-based method, depending on the experimental context (from 1. to 5.), compared to average results obtained by $f \times r_1^{\text{mdrs}}$. Different scenario pool sizes are considered, from 10^2 to 10^4.

$f =$		1.	2.	3.	4.	5.	δ
$f^{\text{mdrs}} \times r_1^{\text{mdrs}}$		*89.9*	*81.0*	*68.1*	*19.6*	*95.1*	*8*
$f^{\text{mdrs}} \times$	$\text{SAA}_{5.10^2}$	86.9	74.7	46.1	12.4	96.5	12
	SAA_{10^3}	85.7	74.5	51.8	12.1	96.2	14
	$\text{SAA}_{5.10^3}$	69.5	56.5	39.4	8.6	86.3	11

Table A.3: Average results obtained by using the SAA-based method, the computation time being restricted to 10 minutes.

Table A.2 reports the average results obtained by repeating all the experiments, while using $f \times \text{SAA}_N$ as LS objective function, with the number N of sampled scenario varying from 10^2 up to 10^4. Both accuracy and computational efficiency of SAA depends on N, which is in fact problem dependent. In our case, $N = 5.10^3$ seems to constitute the best compromise on average, when computational time is limited to 30 minutes. Despite the interesting results obtained by $\text{SAA}_{5.10^3}$, it is however outperformed by r_1^{mdrs} in contexts 2., 3. and 4. In terms of robustness, these contexts are of the utmost importance for anyone concerned by the issues and limitations of processing time estimations.

The superiority of a closed-form function, such as r_1^{mdrs}, over SAA is closely related to the available computation time. It is likely that, provided a couple of hours rather than of 30 minutes, SAA would eventually outperform r_1^{mdrs}. In contrary, reducing computation time to 10 minutes tends to significantly reinforce the superiority of r_1^{mdrs}, as depicted in Figure A.3. We note that, as we reduce computation time, SAA obtains better results by reducing the number of samples, hence improving diversification in the LS process.

A.3.2.5 *Accuracy of r^{mdrs}.*

Moving from the theoretical core problem to the real one faced at MDRS naturally introduces limitations, leading to the proposed sim-

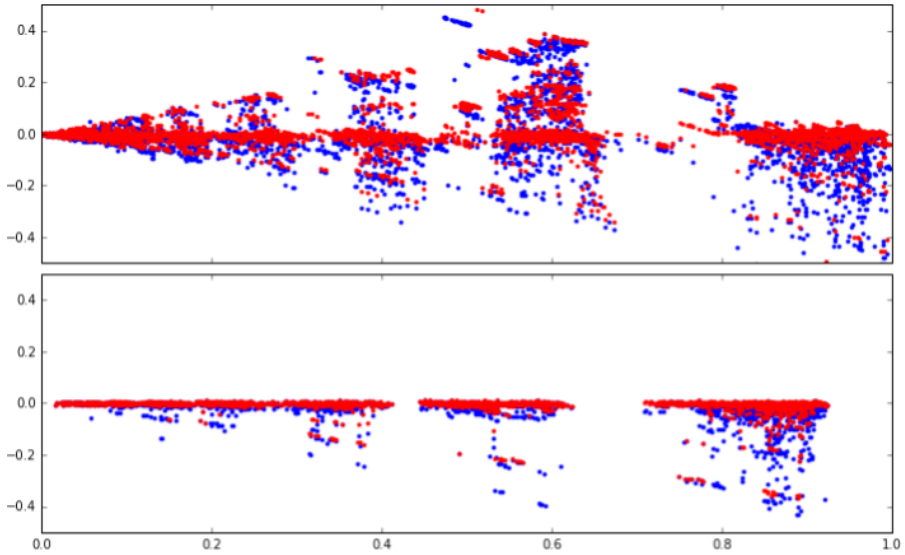


Figure A.5: Average deviation of r_1^{mdrs} (blue) and r_2^{mdrs} (red) robustness measures (y-axis), depending on the solutions' true robustness (x-axis). Top: SOL0. Bottom: SOL7.

plifying assumptions. In Figure A.5, we show how this impacts the accuracy of the robustness functions r_1^{mdrs} and r_2^{mdrs}, on both instances SOL0 and SOL7. For every incumbent solution s encountered by the LS algorithm during the previous experiments ($\pm$8700 solutions for SOL0), we recomputed the true robustness of s under experimental context 1., thus by using SAA_{10^5}. Figure A.5 shows how the value $r_\Delta^{\text{mdrs}}(s) - \text{SAA}_{10^5}(s)$ evolves with respect to $\text{SAA}_{10^5}(s)$.

We naturally observe that the accuracy of the robustness measure, as computed by r_1^{mdrs} and r_2^{mdrs}, first depends on the size of the problem: there is a clear difference between SOL0 (237 jobs) and SOL7 (149 jobs). We note that r_2^{mdrs} seems globally more accurate than r_1^{mdrs}. Having a closer look reveals that under SOL0 (*resp.* SOL7), around 82% (*resp.* 99%) of values computed by r_2^{mdrs} are comprised in $\text{SAA}_{10^5}(s) \pm 0.01$, whereas only 75% (*resp.* 99%) for r_1^{mdrs}.

The average accuracy of r_Δ^{mdrs} globally decreases as the true robustness of the solutions increases. This suggests a hybrid approach, mixing r_Δ^{mdrs} in the early stage of the LS process, whereas SAA (e.g. SAA_{5000}) as soon as the robustness of the incumbent solution exceeds some predefined threshold. Finally, we observe that under SOL7, r_Δ^{mdrs} is globally not overestimating the true robustness. As a matter of fact, while examining the other SOLn instances, the proportion of overestimations tends to decrease with the problem size (*i.e.* the number of jobs).

A.3.2.6 *Length of the horizon and deterministic quality.*

It is interesting that, in the experimental results, no relation appears between the number of operational days and the deterministic quality

of a solution. Our first intuition tells us that the shorter is the horizon, the better the deterministic quality, because the more likely we respect the a priori schedule. We think that what actually happens is that the length of the horizon, in each instance, is compensated by the urgency of the tasks. In fact, when 10 days remain, one can afford postponing tasks because of a poor decisions. When there are only 2 days left, things start to be too urgent and postponing may not remain an option anymore. In the end, both compensate so that the average quality of a deterministic planning may after all not depend on the length of the horizon.

A.4 CONCLUSIONS AND RESEARCH DIRECTIONS

In the context of a recent Mars analog mission, we propose robust models for daily decisions in operations scheduling. Simulations show that our method, by taking the processing time uncertainty into account when designing a schedule, is able to produce solutions that are significantly more reliable than those obtained using a classical deterministic model, even when the available stochastic knowledge is of very poor quality. Even in a context where all the average processing times are globally overestimated, our experiments show that the probability of a mission success is multiplied by three. The solutions' robustness comes at a relatively low price, their quality being impacted by 7% on average, whereas the probability to stay feasible is significantly increased.

We explore two fundamentally different approaches for evaluating the robustness of a solution: the proposed closed-form formulas and the well known Sample Average Approximation method. In particular, it showed promising results for the future onboard scheduler of the M2020 rover Chi et al., 2019. Depending on the problem and available computation time, results suggest that the strengths of both techniques could be combined into a hybrid algorithm.

Project scheduling is a problem daily faced by aerospace engineers and managers, and each problem is specific. Our current understanding of the problem could be improved by conducting further experiments, on a broader set of operational contexts. In fact, exploiting available data from different projects is likely to require new specific, exotic constraints to be considered at planning and optimization phases, leading to a more comprehensive model.

Existing techniques for robust (*a.k.a.* proactive) scheduling are mostly *redundancy-based*, or make use of *temporal protection* (Herroelen and Leus, 2005). Based on random variables, our method considers the original set of tasks without duplicating nor modifying the data, and searches for the expected best sequencing of the tasks. This makes our approach compatible with the two previously cited ones, and experiments should be conducted while mixing for example with

redundancy. Further research should also be considered for alternative computational models such as Bayesian networks (Darwiche, 2009), which naturally apply to our random objective function, and for alternative representations of the uncertainty, e.g. by using fuzzy numbers instead of random variables (Huang and Teghem, 2012).

APPLICATION TO OTHER DOMAINS. Whereas the paper is focused on the Mars scenario, which provided the case study, it could be of interest in many other domains. Consider for instance the case of the biotechnology industrial domain. In biotech companies, scheduling the manufacturing projects (*e.g.* production of vaccines, drugs, *etc.*) is a problem for which our approach is potentially particularly valuable. The main reasons are that: 1) their tasks must be performed by humans, hence having highly variable processing times, and 2) they must cope with really strict operational constraints (the so-called GMPs). In fact, we claim that it applies to any operational context for which planning or scheduling involves time uncertainty and hard operational constraints.

A.4.0.1 *Acknowledgements.*

This work could not have been conducted without the involvement of the entire *UCL to Mars 2018* crew: Bastien Baix, Frédéric Peyrusson, Martin Roumain, Ariane Sablon, Mario Sundic, Sophie Wuyckens and Maximilien Richald. In particular, M. Richald did an enormous amount of work to bring us at MDRS. We also thank the Mars Society, in particular Shannon Rupert and the Mission Support personnel, for creating the conditions that make rotations at the MDRS realistic Mars analog sojourns.

B

A SIMULATION-OPTIMIZATION FRAMEWORK FOR CITY LOGISTICS: APPLICATION ON MULTIMODAL LAST-MILE DELIVERY

City Logistics has attracted considerable interest from the operations research and logistics communities during last decades. It resulted in a broad variety of promising approaches from different fields of combinatorial optimization. However, research on urban freight transportation is currently slowing down due to two different lacks, limiting the exploratory capacity and compromise the technology transfer to the industry. First, the majority of the instances in the literature are based on the generalization of classical instances, often not created for urban applications, or on artificial data, i.e., data not coming from any historical or empirical datasets. Thus, the validation of models and methods becomes more difficult, being the results not directly compared to real or realistic settings. Second, even when some data sources become available, there is no standard way to mixing data gathered from different sources and, from them, generate new instances for urban applications. This paper aims to overcome these issues, proposing a simulation-optimization framework for building instances and assess operational settings. To illustrate the usefulness of the framework, we conduct a case study, in order to evaluate the impact of multimodal delivery options to face the demand from e-commerce, in an urban context as Turin (Italy).

CONTRIBUTIONS. The contribution of paper Perboli et al., 2018, in which the current appendix is based, is twofold. First, we propose to mitigate the two limitations mentioned at Section 4.3.1 by introducing a new standard optimization-simulation framework for City Logistics. Whereas the framework generalizes to many types of routing problems encountered in urban areas, its generality also allows to describe and combine requirements coming from different stakeholders. We categorize the sources of information and we present a tool able to mix data gathered from different sources. So we can generate new instance sets which are realistic, i.e., they include all the characteristics of the original datasets.

Second, we apply our framework to a case study focused on the online urban freight distribution in the city of Turin (Italy). This study concerns the application of the proposed simulation-optimization framework to address the Dynamic and Stochastic Vehicle Routing Problem with Time Windows (DS-VRPTW) problem. We analyse how the solution quality in realistic urban scenarios is sensible to various

stakeholder parameters, such as customers geographical distribution, the available types of vehicles and their limitations, the use of lockers for delivering part of the demand. Our experimental plan leads to a broad variety of realistic benchmarks, each of these being specialized in a particular operational context in the online urban collection of parcels. This portfolio of benchmarks is made available to the community under a simple common format, in order to reuse them in different case studies.

ORGANISATION. The paper introduction, as well as the literature for vehicle routing case studies and applications in realistic urban areas, can be found at Section 4.3 of the thesis. Section B.1 describes the framework we propose in order to analyse realistic urban freight collection problems. Then, Section B.2 shows how our framework can be exploited to realize a concrete case study of online freight collection in a realistic urban context. Note that the case study is the production of Guido Perboli, Mariangela Rosano and Pietro Rizzo, rather than a original contribution of the author of the current thesis. Conclusions and perspectives are discussed in Section B.3, also in appendix.

B.1 A SIMULATION-OPTIMIZATION FRAMEWORK FOR VRP IN URBAN AREAS

The simulation-optimization framework proposed in this section is depicted in Figure B.1. According to Crainic et al., 2017, this framework applies a sequential simulation-optimization, where the simulations are numerical and based on the Monte-Carlo method. The simulation is implemented in Python, while the optimization modules can be defined directly in Python by the Pyomo modelling tool, including the PySP library for Stochastic Programming problems (Hart et al., 2011; Watson et al., 2012) or can be integrated as external modules.

Thus, the framework is composed of the following modules:

1. Data fusion and operational context description;
2. Scenario generation and Simulation;
3. Optimization;
4. Context modification.

B.1.1 *Data fusion and operational context description*

The first phase of the framework consists in describing both the problem studied and the operational context, which may consider different types of data sources. We define the operational context using the following five sources of information: city network graph, vehicles

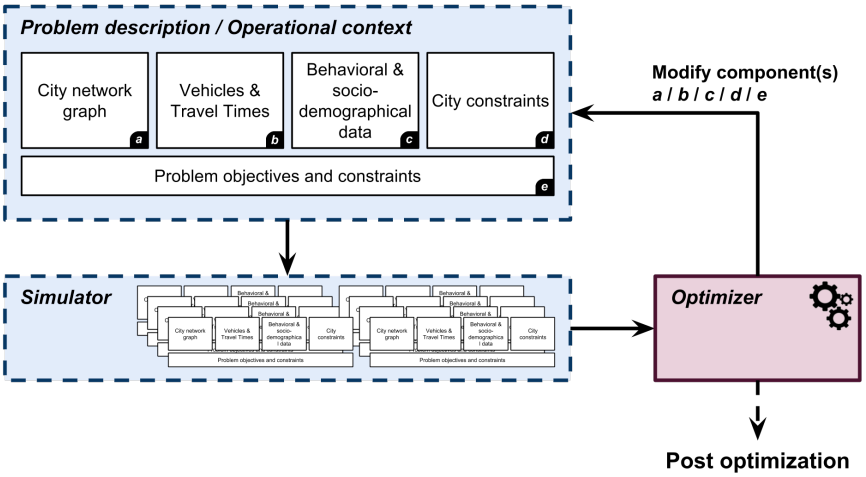


Figure B.1: The simulation-optimization framework.

and travel times, behavioural data (e.g., users choice preferences), socio-demographical data and city constraints (e.g., limited traffic zones, specific restrictions for certain vehicles, etc.) and problem objectives and constraints. Some data may be stochastic, i.e., they can be described by random variables, whenever some component of the operational context is uncertain (e.g., service or travel times, customer demand or presence, etc.). The problem is then fully defined by the problem objectives and constraints data type. The framework requires as input a problem (or operational context) description consisting of five types of data:

a *City network graph and maps.* They are represented by complete directed graphs over a set of depots and customers. Ideally, vertices should be associated geographical coordinates so that to be visualized on real maps. The city network graph is usually obtained using raw data from cartography and the companies, including maps and empirical distributions of customers and depots. Amongst the four main stakeholders identified by Kim et al., 2015 (residents, carriers, shippers and local administrators), the city network graph explains the baseline geographical attributes and means of the residents (customer locations), the shippers (the location of the depots) and the carriers (the available road network).

b *Vehicle fleet and travel times.* They include the specificities of the vehicle types, as capacity, speed, fuel consumption, etc., as well as their respective travel times and costs matrices. Vehicle fleet and travel times capture the means supplied by the shippers. In practice, these are provided by the company and possibly combined with data from external sources (such as sensors spread over the city network). Time dependence and/or uncertainty in the travel times/costs, if any, may also be described here together

with other uncertainties (e.g., vehicle breakdown probability distributions).

c *Behavioural and socio-demographic data.* They include information concerning the density and the purchasing behaviours of final customers for a specific market. Thus, they clearly describe the residents stakeholders in all of their possible attributes. In a static context, these capture the customers' constraints (e.g., time windows, demands, origin-destination matrices, etc.). In dynamic applications, any stochastic knowledge about the customer habits can be described here (e.g., demand or service time probability distributions, etc.).

d *City constraints.* Regulations imposed by the local administrators, such as access time windows (e.g., forbidding trucks during rush hours), vehicle weight restrictions (e.g., no heavy truck in the city centre), etc. City constraints clearly represent the administrators in all the regulations that could be imposed on the other stakeholders (e.g., the carriers).

e *Problem objectives and constraints.* Describe the problem itself in terms of constraints, preferences, as well as the objective function to be optimized. They can be defined by declaring the specific optimization module including its interface with the scenarios or using a MIP solver by the Pyomo modelling tool.

This partitioning of the data into five distinct types allows to easily study the impact of modifying a specific aspect of the operational context. Furthermore, it provides the possibility of combining/reusing data from existing case studies, hence alleviating the full data unavailability issue discussed in Section 4.3. For instance, provided a real-world case study on a classical CVRP, modifying only components *d* and *e* of the problem allows to study the impact on the total carbon emissions of restricting the access of the city centre to green vehicles. From a VRPTW, one can consider a Dial-a-Ride Problem (DARP) by adapting component *c* to specify whether each location should be a pickup or a delivery location and setting maximum ride times. Furthermore, filling component *c* with customer demand probability distributions permits to study Stochastic VRPs, whereas updating component *b* could allow studying the impact of taking travel time variability into account, as well as to use empirical distributions coming from other studies, letting to anonymize industrial data. Similarly, the potential benefits of redesigning part of the road network can be considered by applying those modifications to the component *a*.

B.1.2 *Scenario generation and Simulation*

Once both the problem and the operational context are well defined, a broad set of scenarios is generated by using a high-level scenario generator, which allows the researchers to develop specific scenarios for different frameworks. Each scenario represents a particular realization of all the random variables involved in the problem data. In other words, each scenario is the description of a particular operational day. If the problem and operational day description contain no uncertain data, the scenario becomes the description itself. Otherwise, a set of instances are generated using Monte-Carlo sampling. The framework let the user define deterministic operational scenarios or stochastic ones with associated a scenario tree to each simulation scenario.

The present version of the simulator implements a Monte Carlo method, a module for georeferencing the data and a post-optimization software. In more detail, the method works as follows:

- The Monte Carlo simulation module repeats the following process for a given number $|I|$ of iterations.
  - Given the different data of the operational context as well as eventual distributions of the data themselves, the simulator generates a series of city scenarios.
  - The chosen optimization module is executed in each scenario.
  - A first statistical analysis on the aggregate results of the scenario-based optimization of a single iteration of the Monte Carlo simulation is performed. These data are used in order to check if one or more unrealistic or extreme situations have been introduced in the simulation itself.
  - In order to make a more accurate definition of travel times and cost matrices, the georeference module is used. The georeference feature is implemented by means of Google Earth APIs and it is also used to graphically represent the results of the simulation itself.
- The distribution of the simulation-based optimization solutions is computed and a series of statistical data are collected.
- A post-optimization software module is devoted to computing additional Key Performance Indicators (e.g., CO2 and NOx emissions, stop per working hour, service and travel times).

B.1.3 *Optimization*

During this phase, each scenario is solved using a dedicated optimization algorithm that we consider here as a black box. Provided

that the solver outputs the Key Performance Indicators required by the case study into consideration, the post-optimization analysis is conducted. In order to cope with different contexts in urban areas, this simulation-optimization framework is composed of different building elements addressing the following problems:

- Mathematical model generated by the Pyomo modelling tool;
- VRPTW combined with the load balancing;
- Stochastic TSP;
- Dynamic Stochastic VRPTW solved by the optimization algorithm proposed by Saint-Guillain et al., 2015.

B.1.4 *Context modification*

Eventually, the structure of the operational context description makes it easily modifiable. During this phase, some properties of the description are modified, leading to a new operational context to be analysed by reiterating through phases 2 to 4.

B.2 CASE STUDY: ONLINE URBAN FREIGHT COLLECTION IN TURIN

In order to demonstrate the potentialities of the proposed simulation-optimization framework, we adopt it in the case study of the city of Turin (Italy). Our aim is twofold:

- analyse the impact of multimodal delivery options to face the demand generated by the e-commerce.
- highlight the importance of considering real benchmark data set for DS-VRPTW coming from different sources and stakeholders.

B.2.1 *Operational contexts and benchmark generation*

Online shopping is rapidly increasing the freight flows which transit into the urban areas. According to Cardenas et al., 2016; Copenhagen Economics, 2013; FTI Consulting, 2011, while the business to consumer (B2C) segment of e-commerce represents around 30% of the e-commerce turnover, they generate 56% of all e-commerce shipments. Moreover, e-commerce involves individually fragmented and time-sensitive orders of generally small-sized items, leading to more traffic in urban areas and negative externalities on the environment (Taniguchi and Kakimoto, 2004). These are challenging factors for City Logistics applications, which are more and more focused on the integration of different delivery options (e.g., cargo bikes, drones, lockers, etc.). In fact, our paper addresses this topic, considering the following four benchmarks:

- Benchmark 1 (B1). Only traditional vehicles (i.e., fossil-fuelled vans) are used to manage the parcel delivery in urban areas.
- Benchmark 2 (B2). Outsourcing of classes of parcels to green carrier subcontractors (i.e., they use bikes and cargo bikes) is a common practice to obtain operational and economic efficiency and customer proximity while reducing the environmental impact of logistics activities (Perboli et al., 2017). Thus, in the B2 we consider that a green subcontractor delivers the parcels up to 6 kg in the central and semi-central areas of Turin. On the contrary, the traditional carrier manages all remaining parcels.
- Benchmark 3 (B3). We consider the adoption of delivery lockers. They represent self-service delivery location, in which the customer can pick up or return its parcel, according to the best and convenient time for him. In practice, these can be seen as special "super-customers" that aggregate the daily demands of a subsets of the actual customers.
- Benchmark 4 (B4). In this benchmark, we consider the integration of the vans with both bikes and lockers.

These specific benchmarks derive from the combination of three parameters defined "a priori": the size of the traditional vehicles' fleet, the size of the green vehicles' fleet and the number of lockers. These data are provided by an international parcel delivery companies and an international e-commerce operator, which acting in Turin. Other input data considered in the DS-VRPTW are:

- City network graph and maps. We consider a 2.805 x 2.447 km area in Turin, which includes the centre of the city and a semi-central area, as in Perboli et al., 2017 (see Figure B.2). Moreover, the list of the depots, the locations of lockers and of the potential customers inside the selected area are considered. Concerning the depots, we contemplate a distribution centre located on the outskirts of the urban zone and a mobile depot in the city centre. The former supplies the traditional carrier, while the second represents a satellite facility for the green carrier. In addition, the list of all the roads inside the city area is also required. Such list is arranged as a network of road-segments, each road-segment is defined as a sequence of two connected points, i.e., the crossroads. The information concerning the roads was extracted from the shapefiles made available by the local public authority in Turin. For each road-segment, the average daily speed is measured by speed-sensors. Each element of the mentioned lists is defined with a unique identification number and its real GPS coordinates.
- Vehicle fleet and travel times. As mentioned above, we consider two type of vehicle fleets: vans and cargo bikes. The parcel

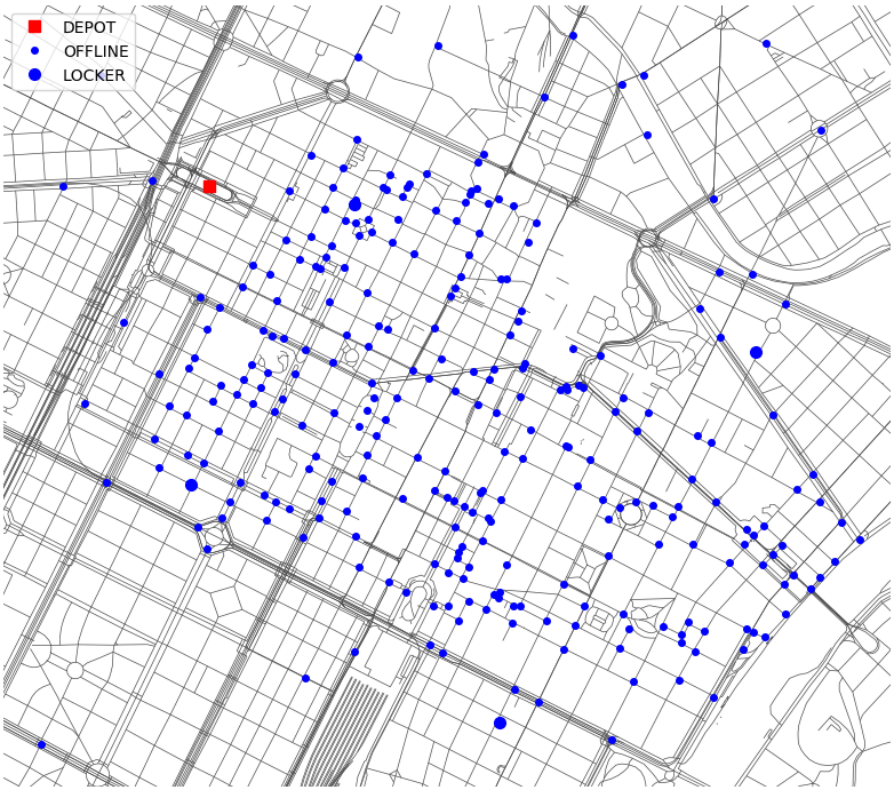


Figure B.2: Area considered in the case study. Note that in the figure the mobile depot (square) and a set of offline customers (circles) and lockers (crosses) are represented.

delivery company interviewed provides the characteristics of vehicle fleets (e.g., capacity, service time, speed). The service-time is a vector containing the information for each type of parcel handled, for the upload from the depot and for the unload into the locker. According to Perboli et al., 2017, we consider three classes of parcels: mailers (i.e., parcel with a weight up to 3 kg) small parcels (i.e., parcel with a weight between 3 and 6 kg) and large deliveries (i.e., parcel with weight over 6 kg). The expected number of parcels for each class, expressed as a percentage of the total number of parcels delivered, are shown in the Table B.1.

- Behavioural and socio-demographic data. The horizon size is given here. We consider an 8-hours working day, from 9:00 to 17:00. The time-unit considered is 1 minute and the time-horizon is split into four time-buckets with the same length. For each potential customer, the demand expressed as parcel's volume is provided, together with the time-window for the service. The time-windows are assigned considering the percentage of prime members (i.e., those whose requests are prioritized restricting the time-window to the first two time-buckets). Then, the expected behaviour of each potential customer of the DS-VRPTW is described. It gives the probability that, for each customer location i and each time-unit t of the time-horizon, an online request (i.e., picking up a parcel) appears at time t for location i.

- City constraints. We do not consider any specific city constraint.

- Problem objectives and constraints. The objective is first to maximize the (expected) number of online requests satisfied by the end of the horizon, and second minimize the total distance travelled by the vehicles.

The operational context defines the number of potential customers in the city map, the number of offline customers selected among the potential ones and the percentage of prime members. In this simulation, we generate three different-sized operational contexts with respectively 500, 250 and 100 potential customers. Each context contains 70% of offline customers and 25% of prime members. These potential customers are randomly picked from the pool of potential customers listed in the input data and then anonymized for confidentiality matters, by offsetting the Cartesian coordinate system. Once the potential customers are defined, it is possible to compute the matrix of the mutual distances among the customers and the depots on the map. Such distances are computed applying the Dijkstra's shortest path to the network of road-segments specified in the input data. The high level of detail in the network, coupled with the haversine formula used to estimate the distance between each pair of points that compose a road-segment, provide us an outcome, which is much more accurate than a simple

application of the Manhattan distance. The obtained results are in line with the one provided by the most common web-mapping service Google Maps. From the distance matrix is then possible to compute the travel-times between pairs of locations, by using the measured road-segments' speeds available in the input data. The set of online requests appearing during the daily time-horizon is defined by considering three different degrees of dynamism: 15%, 30% and 45%. Three sub-contexts are thus defined for each operational context, according to the degree of dynamism assignation. For each sub-context, a set of n instances is sampled by generating n Poisson Random Variates (PRVs) with parameter λ dependent on the degree of dynamism considered. Each PRV i represents the effective number of online requests that appear in the Instance i. The accorded set of online customers is finally randomly selected from the list of potential ones, allowing multiple requests for the same customers, but provided that they appear at different moments (i.e., time units). Each scenario, which in the case of DS-VRPTW corresponds to a sequence of revealed online requests along the day together with their specific reveal times and locations, is then independently solved by the optimizer. All the instances are generated and classified in classes (i.e., the benchmarks presented above), depending on the operational context, as described in Section B.1. The benchmarks are available online on the git repository available at the address `https://bitbucket.org/orogroup/city-logistics.git`.

Table B.1 resumes the values of the input data considered in our analysis. This information derive from interviews with Chief Executive Officer (CEO) and logistic director of an international parcel delivery company and of an e-commerce company operating in Turin. For further information about these data, the interested reader could refer to Perboli et al., 2017. Moreover, the tests are conducted using real data concerning the customer distribution and daily volumes of deliveries in Turin between 2014 and 2015, provided by the international parcel delivery company that operates in Italy and is involved in the URBan Electronic LOGistics (URBeLOG, URBeLOG, 2015).

B.2.2 *Specific optimization problem definition*

Ritzinger et al., 2016 provide a recent review on DS-VRP(TW)s. At any moment of the operational day, a DSS is then responsible for maintaining a feasible solution (i.e., satisfying the previously accepted requests) while dealing with the appearance of online requests. In such a context, the objective is usually expressed either in terms of expected operating costs, such as travel distances, additional vehicles, and expensive penalties whenever an online request cannot be accepted, or in terms of expected profit when one gets rewarded at each satisfied request.

Table B.1: Input data

Classes of parcel			
Class	*Weight range*	*% on total parcels*	
Mailer	0-3 kg	57%	
Small delivery	3-6 kg	13%	
Large delivery	> 6 kg	30%	
Capacity			
Vehicle	*Parcel size max*	*Capacity*	*Coverage*
Locker	6 kg	20* *parcels*	1 km
Van	70 kg	700 kg	NA
Cargo bike	6 kg	70 kg	NA
Speed in urban area		**Setup time**	
Vehicle	*Speed*	Load locker	15 min
Van	40 km/h	Load bikes at mobile depot	15 min
Cargo bike	20 km/h		
Service time to deliver each class of parcels			
Vehicle	*Mailer*	*Small delivery*	*Large delivery*
Van	4 min	4 min	5 min
Cargo bike	2 min	2 min	NA

* max number of parcel per day. Note that part of the locker is actually filled with the parcels of the previous three days

As introduced above, we adopt the proposed simulation-optimization framework to address the DS-VRPTW problem that we define as follows. Given a discrete horizon of length h, a depot location and a set of n customer locations, we define the set $R = \{1, \dots, n\} \times \{1, \dots, h\}$ of potential requests, that is, one potential request at each time unit for each customer location. We assume the probability of each potential request to appear to be known, together with its own demand, service time and time window in case it actually appears. Whenever it happens and by the end of the current time unit, the request must be either *accepted* or *rejected*. In case it is accepted, the request must be guaranteed to be satisfied according to its time window and the vehicles capacity constraints. A function $c : R \to \mathbb{R}_+$ defines the penalty cost inquired whenever a request $r \in R$ is rejected. Provided a finite set of capacitated vehicles, the asymmetric travel times matrix between all pairs of locations and the set of potential requests, the goal (at each time unit) is to operate the fleet of vehicles such that the expected total penalty cost is minimized by the end of the horizon.

Generally speaking, VRPs aims at modelling and solving a real-life common operational problem, in which a known set of geographically distributed customer (pickup) demands must be satisfied using a fleet of capacitated vehicles. The VRPTW introduces a time dimension by restricting each customer visit in a predefined interval. The objective is to find an optimal feasible solution, where optimality is classically defined in terms of travel costs. In urban applications, some additional characteristics must be taken into account: the *dynamic of the customers*, i.e., the customer requests are not known in advance, but are instead revealed as the operations go, and the *stochastic nature of some parameters*, i.e., some attributes are random variables. For the aforementioned reasons, we incorporate as optimization model the Dynamic Stochastic VRP with Time windows (DS-VRPTW) solved by the algorithm described in Saint-Guillain et al., 2015. Based on Monte Carlo sampling, the main idea of the Global Stochastic Assessment (GSA) algorithm aims at maintaining a unique feasible current solution being continuously optimized with respect to a restricted pool of sampled scenarios, while preserving nonanticipativity constraints in the evaluation function. A classical local search approach is used, exploiting well-known VRP neighbourhood operators such as relocate, swap, inverted 2-opt and cross-exchange (Kindervater and Savelsbergh, 1997, Taillard et al., 1997) to construct neighbouring solutions. A diversification mechanism is provided by regularly renewing the scenario pool, hence modifying the shape of the evaluation function, making needless the use of any other meta-heuristic. Note that the algorithm implements a relocation strategy, allowing the vehicles to anticipatively travel and possibly wait at promising strategical (customer) locations, even when these do not require a service (yet, if any).

Concerning the related DS-VRPTW applications, only a few realistic case studies involving the DS-VRPTW are present in literature Hvattum et al., 2006; Schilde et al., 2011. In Bent and Van Hentenryck, 2007, the generated benchmark, later extended by Saint-Guillain et al., 2015, is highly artificial and based on Solomon's instances. A realistic benchmark is considered in Hvattum et al., 2006, based on a real-world case observed in a leading distribution company in Linjegods, Norway. Whereas the customer distributions are based on real-world data, the benchmark assumes Euclidean distances between them. Unfortunately, the benchmark is not available anymore. The dynamic and stochastic dial-a-ride problem considered in Schilde et al., 2011 is a generalization of the DS-VRPTW where each customer consists in both a pickup and a delivery request, associated with a maximal ride time constraint. In their study, the authors generated their benchmarks based on real-world assumptions, using daily operation performed by the Austrian Red Cross during the year 2004. Up to our knowledge there is still no standard benchmark for realistic (urban) dynamic and stochastic VRPTWs. The benchmarks available in the literature are either too artificial or do not include the probabilistic knowledge required in dynamic and stochastic VRPTWs.

B.2.3 *Numerical analysis*

In this section, computational tests of the simulation-optimization framework on the DS-VRPTW are described. The experimental plan is composed of a set of randomly generated test problems. For each benchmark and each operational context, we performed 10 independent runs. Thus, we obtained totally 360 instances, which were independently solved by the optimizer.

To evaluate the results we measured different Key Performance Indicators (KPIs), according to the following standpoints:

- Economic Sustainability. As defined in Perboli et al., 2017, the carrier incurs in operating costs related to fleet management and maintenance, and personnel costs. These costs are increased by a margin equal to 15% when the fleet is managed by an external firm subcontractor. Moreover, typical contract scheme in the parcel delivery industry imposes the conversion from a cost per kilometre to a cost per stops. Thus, the KPI measured are:
  - Cost per stop (internal fleet), in the case in which the fleet of vehicles is owned by the carrier (CpsI).
  - Cost per stop (external fleet) in the case in which the fleet of vehicles is owned by the subcontractor (CpsE).

  For further details about the computation of operating costs and each cost item, see Perboli et al., 2017.

- Environmental Sustainability. In order to evaluate the impact of the adoption of green delivery means on the environment, we computed the CO_2 savings (CO_2EMsav) as the kilograms of CO_2 not emitted in the B2, B3, and B4. Moreover, as the externalities have a social cost that impacts on the economic efficiency of the logistics operator, we express the emissions saved (compared with the B1) in monetary terms by applying the carbon tax, based on the average price paid for CO_2 emissions (Perboli et al., 2017). This KPI is the environmental costs saving (CO_2CSsav). Note that according to the regulation ISO/TS 14067:2013 we consider the total amount and costs of Green Houses Gas (GHG) emitted directly or indirectly by the overall parcel delivery chain.

- Operational Sustainability. It is referred to the operational performance and efficiency of each operator involved in the urban parcel distribution. Generally, it is expressed in terms of number of parcels delivered per hour (nD/h)

- Social Sustainability. It is strictly related to the operational sustainability, as the fulfilment of the increasing demand of time-sensitive and online deliveries and the high service quality required by the final customers affect the working conditions of the drivers.

To provide the reader an easier understanding of the results, we computed the percentage of each KPI compared with the reference benchmark B1, as shown in Figure B.3. Thus, Figure B.3 depicts the performance of the traditional courier in the B2, B3, and B4. The values are computed as percentage variation of each KPI with respect to the value of the same KPI in the Benchmark $B1$. In particular, the Δoperating costs and Δenvironmental costs show the percentages of costs savings, both operating and environmental, that the traditional carrier obtains when the parcels up to 6 kg are outsourced to the green carrier or delivered by means of the lockers. While, the item Δefficiency represents the loss of efficiency that affects the traditional carrier due to the reduced number of deliveries and the high saturation of vans, particularly in B2.

Figure B.3 highlights improvement of both economic and environmental sustainability when green delivery options (cargo bikes and lockers) are introduced. In particular, in B2 the adoption of cargo bikes and the optimization of routes lead to a reduction of the vans used of about 32% and of the kilometres travelled, with consequent benefits in terms of reduction of operating costs (-37%). At the same time a reduction of the CO_2 emission on average of 303 kg, is registered, which correspond to a decrease of 40% in the environmental costs.

Figure B.4 reports the number of deliveries per hour of traditional vans and green vehicles in the different operational contexts, segmenting the results according to the number of customers in the scenarios

and the degree of dynamism. For the operational context $B1$ the green carrier has no bar because it is not present in it. Thus, the number of deliveries per hour (nD/h) are given for the traditional vans only. They are reported in order to provide a reference value while comparing the results in the operational contexts $B2$ and $B4$. The values of $B3$ are not given because no green vehicle is usable in this operation context. As figured out in Perboli et al., 2017, the outsourcing of the small parcels (mailers and small deliveries) to the green carrier, the traditional one incurs in a reduction of efficiency of 80% at maximum. This means, for example, a reduction of the number of deliveries per hour from 126 to 25 in 10 working days when there are 100 customers locations and 30% of dynamism. Figure B.4 shows how the green carrier reaches the highest number of parcels delivered when the degree of dynamism is equal to 45%. Similarly, when the deliveries are managed by means of lockers, there is an improvement of the economic and environmental sustainability. However, here the reduction of the costs, both operative (-25%) and environmental (-21%), is lower than B2. The reason is that, although there is a reduction in the kilometres travelled by the vans to serve the home deliveries directly to the customers, these vehicles are still adopted to reach and supply the lockers. When the number of customers to serve and the degree of dynamism are both to a low level, the adoption of lockers leads the highest decrease of the number of deliveries managed by the traditional carrier (-38%). This impact on the efficiency corresponds to costs savings of the same order. On the contrary, although the presence of the lockers, when the degree of dynamism is high and, i.e., the online requests increase, they are served by the traditional carrier. In fact, when the class of customers locations and the degree of dynamism are of respectively 500 and 45%, the loss of efficiency for the traditional carrier reaches the minimum value (-12%). A significant finding is that, combining all the delivery options (B4), the highest reduction of emissions and operating costs is reached. In particular, this reduction becomes more evident when there is a low number of customers. Thus, they are served by environmental-friendly delivery modes, while a very few number of parcels is delivered by the traditional carrier. On the contrary, when we consider 500 customers, the performance of the traditional and the green carriers in terms of efficiency are similar to those achieved in the B2. This means that in case of high demand the lockers saturate quickly. Thus, bikes and vans (particularly) are used to cope the most considerable part of the deliveries, as more flexible.

The results obtained figured out that we can minimize the number of rejected requests by adopting the optimization solver. In fact, only 1, 2 and 9 requests are rejected respectively in B2, B3, and B4. This rejection happens when the classes of customer locations and/or the degree of dynamism are medium-high. On the contrary, in the other instances all the online requests are fulfilled.

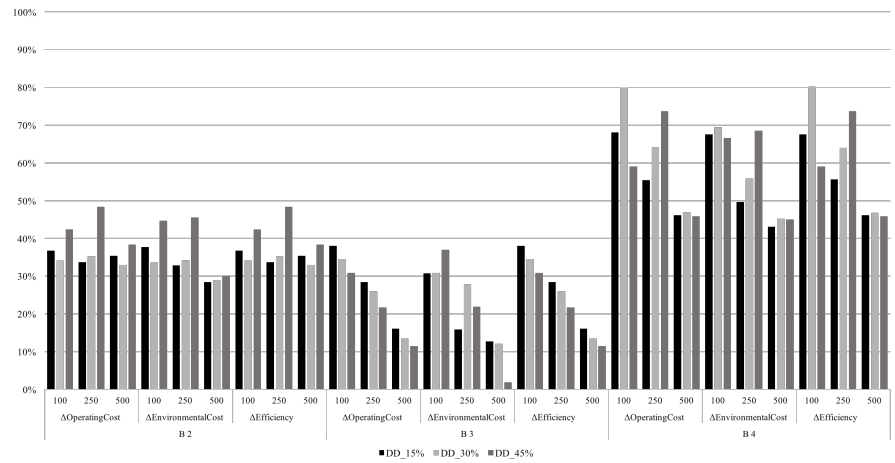


Figure B.3: Performance of the traditional carrier, when cargo bikes and lockers are adopted.

As mentioned above, the operational sustainability is strictly related to the social sustainability. The integration of traditional delivery mode with the two new options (i.e., bikes and lockers) could have a positive impact on the social sustainability. In fact, at present, the drivers are hard-pressed to face the high demand of home-deliveries, respecting the time windows. Moreover, their working conditions are affected by a broad range of issues, as traffic and congestion, unavailability of loading/unloading zone, as well as second-time deliveries because the customer is not at home. All these problems make difficult in a regular working shift the achievement of the 80 deliveries per day imposed by the common practice in the industry (Perboli et al., 2017). Thus, considering the revenues based on the number of deliveries and the penalties in case of not fulfilment, these problems impose pressure on the drivers of the traditional carrier company. On the contrary, the reduction of the number of parcels that the traditional carrier have to deliver, combined with the optimization of the routes and the reduction of vehicles on road, lead to a less and more balanced workload and the improvement of the working conditions. However, a necessary fundamental condition is that this integration must be made in a reasonable manner. In fact, as stated in Perboli et al., 2017, the loss of efficiency for the traditional carrier must be contained and balanced by an increase in service quality led by the bikes and lockers and by a continuous process of optimization and monitoring of the activities in the overall last-mile chain.

B.3 CONCLUSIONS AND PERSPECTIVES

In this paper we presented a new simulation-optimization framework for building instances and assess operational settings. This research topic arose from the emerged lack of an available realistic benchmark

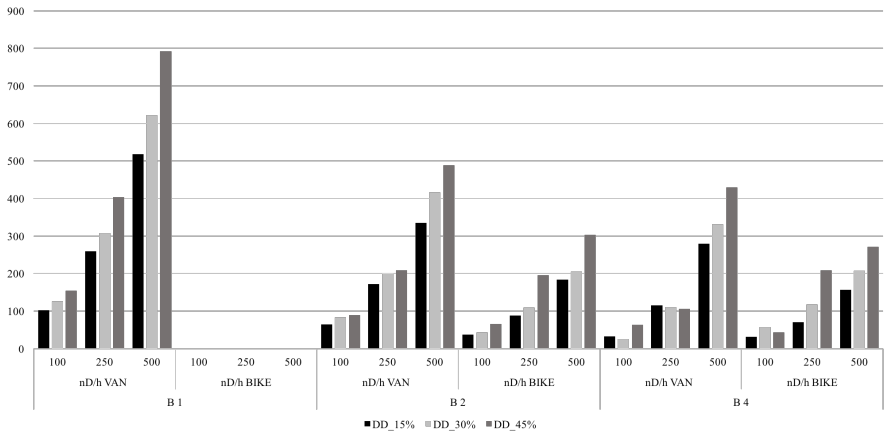


Figure B.4: Performance of the green carrier. *B*3 is not reported, not involving any green vehicle for the delivery.

for VRP in City Logistics applications. By proposing a standard representation of many types of vehicle routing problems (including City VRPs), our framework allows to mitigate that issue by making easier to share, adapt, reuse and even merge data and benchmarks from different sources or studies. We illustrated the proposed framework by applying it to an online parcel delivery problem in the medium-sized city of Turin (Italy). The novelty of our contribution is that the realism of case study is guaranteed by the introduction in the framework of different real data sources and stakeholder requirements. In addition, we considered the integration of different deliveries modes (i.e., cargo bikes and lockers), reflecting the current practices in the city, which are devoted to the adoption of green delivery options. The experimental plan conducted highlighted that the switch to vehicles with a low environmental impact and to lockers, could lead an improvement in the economic efficiency of the business model of the traditional carrier and in the working conditions of the drivers. Moreover, the bikes represent the most suitable vehicles to face the online requests of deliveries, due to their high flexibility. Furthermore, the adoption of environmental-friendly vehicles could result in benefits in terms of CO_2 emissions reduction. At the same time, this integration of different deliveries options could cause a loss of efficiency for traditional carriers. An important outcome obtained is that a multimodal last-mile delivery achieved by means the integration of all the delivery options considered allows reaching the lowest levels of emissions when the number of the customers is low/medium. On the contrary, vans and bikes represent the most appropriate means to deal with high demand, while still pursuing environmental benefits. Finally, focusing on the framework proposed, the most relevant advantage resides in the possibility to generate diversified operational contexts, customized degree of dynamism, huge sets of instances and tailored classes of benchmark to test any kind framework. Thus, it represents

an important contribution to the scientific research community that could adopt it to analyse different City Logistics applications.

Future development will be the usage of the simulation-optimization tool to validate a new system of two-echelon neighbourhood exchange points integrating more types of delivery options, as well as integrate into the simulation module a discrete event simulator.

BIBLIOGRAPHY

Ahmed, Shabbir and Alexander Shapiro (2002). "The sample average approximation method for stochastic programs with integer recourse." In: *E-print: http://www.optimization-online.org*.

Angulo, Gustavo, Shabbir Ahmed, and Santanu S Dey (2016). "Improving the integer L-shaped method." In: *INFORMS Journal on Computing* 28, pp. 483–499.

Applegate, David and William Cook (1991). "A computational study of the job-shop scheduling problem." In: *ORSA Journal on computing* 3.2, pp. 149–156.

Asmussen, Søren and Peter W Glynn (2007). *Stochastic Simulation: Algorithms and Analysis*. Vol. 57. Springer.

Balinski, Michel L and Richard E Quandt (1964). "On an integer program for a delivery problem." In: *Operations Research* 12.2, pp. 300–304.

Bard, Jonathan F., George Kontoravdis, and Gang Yu (2002). "A Branch-and-Cut Procedure for the Vehicle Routing Problem with Time Windows." In: *Transportation Science* 36.2, pp. 250–269. ISSN: 0041-1655. DOI: 10.1287/trsc.36.2.250.565.

Bartholdi III, John J et al. (1983). "A minimal technology routing system for meals on wheels." In: *Interfaces* 13.3, pp. 1–8.

Bautista, Joaquín, Elena Fernández, and Jordi Pereira (2008). "Solving an urban waste collection problem using ants heuristics." In: *Computers and Operations Research* 35.9, pp. 3020–3033. ISSN: 03050548. DOI: 10.1016/j.cor.2007.01.029.

Beck, J Christopher, Patrick Prosser, and Evgeny Selensky (2003). "Vehicle Routing and Job Shop Scheduling : What 's the Difference ?" In: *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 267–276.

Bektas, Tolga and Gilbert Laporte (2011). "The Pollution-Routing Problem." In: *Transportation Research Part B: Methodological* 45.8, pp. 1232–1250. ISSN: 03772217. DOI: 10.1016/j.ejor.2013.08.002.

Bent, Russell W and Pascal Van Hentenryck (2004a). "Regrets only! online stochastic optimization under time constraints." In: *AAAI*, pp. 501–506.

– (2004b). "Scenario-based planning for partially dynamic vehicle routing with stochastic customers." In: *Operations Research* 52.6, pp. 977–987.

– (2004c). "The Value of Consensus in Online Stochastic Scheduling." In: *ICAPS* 1, pp. 219–226.

Bent, Russell W and Pascal Van Hentenryck (2007). "Waiting and Relocation Strategies in Online Stochastic Vehicle Routing." In: *IJCAI*, pp. 1816–1821.

Bent, Russell W, Irit Katriel, and Pascal Van Hentenryck (2005). "Sub-optimality approximations." In: *Principles and Practice of Constraint ...* Pp. 1–15.

Beraldi, Patrizia et al. (2005). "Efficient neighborhood search for the probabilistic pickup and delivery travelling salesman problem." In: *Networks* 45.4, pp. 195–198.

Berhan, Eshetie et al. (2014). "Stochastic Vehicle Routing Problem: A Literature Survey." In: *Journal of Information & Knowledge Management* 13.03, p. 1450022. ISSN: 0219-6492. DOI: 10.1142/S0219649214500221.

Bertsimas, Dimitris J (1988). "Probabilistic combinatorial optimization problems." PhD thesis. Massachusetts Institute of Technology.

– (1992). "A vehicle routing problem with stochastic demand." In: *Operations Research* 40.3, pp. 574–585.

Bertsimas, Dimitris J and Louis H Howell (1993). "Further results on the probabilistic traveling salesman problem." In: *European Journal of Operational Research* 65.1, pp. 68–95.

Bertsimas, Dimitris J and G Van Ryzin (1991). "A stochastic and dynamic vehicle routing problem in the Euclidean plane." In: *Operations Research*.

Bertsimas, Dimitris J and David Simchi-Levi (1996). "A new generation of vehicle routing research: robust algorithms, addressing uncertainty." In: *Operations Research* 44.2, pp. 286–304.

Bertsimas, Dimitris J, Garrett Van Ryzin, and G Van Ryzin (1993). "Stochastic and dynamic vehicle routing in the Euclidean plane with multiple capacitated vehicles." In: *Operations Research* 41.1, pp. 60–76.

Bertsimas, Dimitris J, Philippe Chervi, and Michael Peterson (1995). "Computational approaches to stochastic vehicle routing problems." In: *Transportation science* 29.4, pp. 342–352.

Bianchi, Leonora and Ann M Campbell (2007). "Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem." In: *European Journal of Operational Research* 176.1, pp. 131–144. ISSN: 03772217. DOI: 10.1016/j.ejor.2005.05.027.

Bianchi, Leonora, Luca Maria Gambardella, and Marco Dorigo (2002). "An ant colony optimization approach to the probabilistic traveling salesman problem." In: *International Conference on Parallel Problem Solving from Nature*, pp. 883–892.

Bianchi, Leonora, Joshua Knowles, and Neill E Bowler (2005). "Local search for the probabilistic traveling salesman problem: correction to the 2-p-opt and 1-shift algorithms." In: *European Journal of Operational Research* 162.1, pp. 206–219.

Birge, John R and François Louveaux (2011). *Introduction to stochastic programming*. ISBN: 9781461402367.

Bowler, Neill E, Thomas M Fink, and Robin C Ball (2003). "Characterisation of the probabilistic travelling salesman problem." In: *Physical Review E* 68.3, p. 036703. DOI: 10.1103/PhysRevE.68.036703. arXiv: 0011023 [physics].

Branke, Jürgen et al. (2005). "Waiting Strategies for Dynamic Vehicle Routing." In: *Transportation Science* 39.3, pp. 298–312. ISSN: 0041-1655. DOI: 10.1287/trsc.1040.0095.

Braun, Mikio L and Joachim M Buhmann (2002). "The noisy Euclidean traveling salesman problem and learning." In: *Advances in neural information processing systems*, pp. 351–358.

Campbell, Ann M and Barrett W Thomas (2008a). "Challenges and advances in a priori routing." In: *The vehicle routing problem: latest advances and new challenges* 43, pp. 123–142.

– (2008b). "Probabilistic traveling salesman problem with deadlines." In: *Transportation Science* 42.1, pp. 1–27.

Cardenas, Ivan et al. (2016). "Spatial characteristics of failed and successful E-commerce deliveries in Belgian cities." In: *Information Systems, Logistic and Supply Chain Conference*.

Cattaruzza, Diego et al. (2017). "Vehicle routing problems for city logistics." In: *EURO Journal on Transportation and Logistics* 6.1, p. 51. ISSN: 2192-4384. DOI: 10.1007/s13676-014-0074-0.

Chaari, Tarek et al. (2014). "Scheduling under uncertainty: Survey and research directions." In: *2014 International Conference on Advanced Logistics and Transport (ICALT)*, pp. 229–234. ISSN: 9781479948390. DOI: 10.1109/ICAdLT.2014.6866316.

Chang, Tsung-Sheng and Hui-Mei Yen (2012). "City-courier routing and scheduling problems." In: *European Journal of Operational Research* 223.2, pp. 489–498. ISSN: 03772217. DOI: 10.1016/j.ejor.2012.06.007.

Chao, I-Ming, Bruce L. Golden, and Edward A. Wasil (1996). "The team orienteering problem." In: *European Journal of Operational Research* 88.3, pp. 464–474. ISSN: 03772217. DOI: 10.1016/0377-2217(94)00289-4.

Chapleau, Luc, Jacques-A Ferland, and Jean-Marc Rousseau (1985). "Clustering for routing in densely populated areas." In: *European Journal of Operational Research* 20.April 1984, pp. 48–57.

Chi, Wayne et al. (2019). "Optimizing Parameters for Uncertain Execution and Rescheduling Robustness." In: *29th International Conference on Automated Planning and Scheduling (ICAPS)*.

Christiansen, Christian H. and Jens Lysgaard (2007). "A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands." In: *Operations Research Letters* 35.6, pp. 773–781. ISSN: 01676377. DOI: 10.1016/j.orl.2006.12.009.

Chung, Kai Lai (2001). *A course in probability theory*. Academic press.

Copenhagen Economics (2013). *E-commerce and delivery*.

Cordeau, Jean-François and Gilbert Laporte (2003). "The dial-a-ride problem (DARP): Variants, modeling issues and algorithms." In: *4OR: A Quarterly Journal of Operations Research* 1.2, pp. 89–101.

Cordeau, Jean-François et al. (2007). "Vehicle routing." In: *Transportation* 14, pp. 367–428.

Crainic, T G, G Perboli, and M Rosano (2017). *Simulation of intermodal freight transportation systems: a taxonomy*. Tech. rep. CIRRELT.

Darwiche, Adnan (2009). *Modeling and reasoning with Bayesian networks*. Cambridge university press.

Dolan, Elizabeth D and Jorge J Moré (2002). "Benchmarking optimization software with performance profiles." In: *Mathematical programming* 91.2, pp. 201–213.

Donati, Alberto V. et al. (2008). "Time dependent vehicle routing problem with a multi ant colony system." In: *European Journal of Operational Research* 185.3, pp. 1174–1191. ISSN: 03772217. DOI: 10.1016/j.ejor.2006.06.047.

Dror, Moshe, Gilbert Laporte, and Pierre Trudeau (1989). "Vehicle routing with stochastic demands: Properties and solution frameworks." In: *Transportation science* 23.3, pp. 166–176.

Dulac, Gilles, Jacques A. Ferland, and Pierre A. Forgues (1980). "School bus routes generator in urban surroundings." In: *Computers and Operations Research* 7.3, pp. 199–213. ISSN: 03050548. DOI: 10.1016/0305-0548(80)90006-4.

Dupacova, J, N Growe-Kuska, and W Romish (2003). "Scenario reduction in stochastic programming An approach using probability metrics." In: *Mathematical programming* 95.3, pp. 493–511.

Eglese, Richard, Will Maden, and Alan Slater (2006). "A Road Timetable™ to aid vehicle routing and scheduling." In: *Computers and Operations Research* 33.12, pp. 3508–3519. ISSN: 03050548. DOI: 10.1016/j.cor.2005.03.029.

Erdogan, Sevgi and Elise Miller-Hooks (2012). "A Green Vehicle Routing Problem." In: *Transportation Research Part E: Logistics and Transportation Review* 48.1, pp. 100–114. ISSN: 13665545. DOI: 10.1016/j.tre.2011.08.001.

Escuín, David, Carlos Millán, and Emilio Larrodé (2012). "Modelization of Time-Dependent Urban Freight Problems by Using a Multiple Number of Distribution Centers." In: *Networks and Spatial Economics* 12.3, pp. 321–336. ISSN: 1566113X. DOI: 10.1007/s11067-009-9099-6.

FTI Consulting (2011). *Intra-community cross-border parcel delivery London*.

Faulin, Javier et al. (2011). "Solving the capacitated vehicle routing problem with environmental criteria based on real estimations in road transportation: A case study." In: *Procedia-Social and Behavioral*

Sciences 20, pp. 323–334. ISSN: 18770428. DOI: 10.1016/j.sbspro.2011.08.038. arXiv: arXiv:1011.1669v3.

Fichter, Donn (1964). *Individualized automatic transit and the city*.

Flatberg, Truls et al. (2007). "Dynamic and stochastic vehicle routing in practice." In: *Dynamic Fleet Management*. Springer, pp. 41–63.

Fleischmann, Bernhard et al. (2004a). "Dynamic Vehicle Routing Based on Online Traffic Information." In: *Transportation Science* 38.4, pp. 420–433. ISSN: 0041-1655. DOI: 10.1287/trsc.1030.0074.

Fleischmann, Bernhard, Martin Gietz, and Stefan Gnutzmann (2004b). "Time-Varying Travel Times in Vehicle Routing." In: *Transportation Science* 38.2, pp. 160–173. ISSN: 0041-1655. DOI: 10.1287/trsc.1030.0062.

Gauvin, Charles, Guy Desaulniers, and Michel Gendreau (2014). "A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands." In: *Computers & Operations Research* 50, pp. 141–153. ISSN: 03050548. DOI: 10.1016/j.cor.2014.03.028.

Gendreau, Michel, Gilbert Laporte, and René Séguin (1995). "An exact algorithm for the vehicle routing problem with stochastic demands and customers." In: *Transportation Science* 29.2, pp. 143–155.

– (1996a). "A Tabu Search Heuristic for the Vehicle Routing Problem with Stochastic Demands and Customers." In: *Operations Research* 44.3, pp. 469–477. ISSN: 0030-364X. DOI: 10.1287/opre.44.3.469.

– (1996b). "Stochastic vehicle routing." In: *European Journal of Operational Research* 88.1, pp. 3–12.

Gendreau, Michel et al. (2016). "Future Research Directions in Stochastic Vehicle Routing." In: 1655.October, pp. 1–11. ISSN: 1526-5447. DOI: 10.1287/trsc.1090.0306.

Ghiani, Gianpaolo et al. (2009). "Anticipatory algorithms for same-day courier dispatching." In: *Transportation Research Part E: Logistics and Transportation Review* 45.1, pp. 96–106. ISSN: 13665545. DOI: 10.1016/j.tre.2008.08.003.

Goren, Selcuk and Ihsan Sabuncuoglu (2008). "Robustness and stability measures for scheduling: Single-machine environment." In: *IIE Transactions* 40.1, pp. 66–83. ISSN: 0740817X. DOI: 10.1080/07408170701283198.

Gounaris, Chrysanthos E et al. (2014). "An adaptive memory programming framework for the robust capacitated vehicle routing problem." In: *Transportation Science* 50.4, pp. 1239–1260.

Hall, N G and M J Magazine (1994). "Maximizing the value of a space mission." In: *European Journal of Operational Research* 78.2, pp. 224–241. DOI: 10.1016/0377-2217(94)90385-9.

Hart, William E, Jean-Paul Watson, and David L Woodruff (2011). "Pyomo: modeling and solving mathematical programs in Python." In: *Mathematical Programming Computation* 3.3, pp. 219–260.

Heilporn, Géraldine, Jean-François Cordeau, and Gilbert Laporte (2011). "An integer L-shaped algorithm for the Dial-a-Ride Problem

with stochastic customer delays." In: *Discrete Applied Mathematics* 159.9, pp. 883–895.

Helgason, Thorkell and Stein W Wallace (1991). "Approximate scenario solutions in the progressive hedging algorithm." In: *Annals of Operations Research* 31, pp. 425–444.

Henchiri, Abir, Monia Bellalouna, and Walid Khaznaji (2014). "A probabilistic traveling salesman problem: a survey." In: *FedCSIS Position Papers*. Vol. 3, pp. 55–60. DOI: 10.15439/2014F381.

Herroelen, Willy and Roel Leus (2005). "Project scheduling under uncertainty: Survey and research potentials." In: *European Journal of Operational Research* 165.2, pp. 289–306. ISSN: 03772217. DOI: 10.1016/j.ejor.2004.04.002.

Hjorring, Curt and John Holt (1999). "New optimality cuts for a single-vehicle stochastic routing problem." In: *Annals of Operations Research* 86, pp. 569–584.

Ho, Sin C and Dag Haugland (2011). "Local search heuristics for the probabilistic dial-a-ride problem." In: *Or Spectrum* 33.4, pp. 961–988.

Hollis, B. L., M. A. Forbes, and B. E. Douglas (2006). "Vehicle routing and crew scheduling for metropolitan mail distribution at Australia Post." In: *European Journal of Operational Research* 173.1, pp. 133–150. ISSN: 03772217. DOI: 10.1016/j.ejor.2005.01.005.

Hu, Xiangpei et al. (2009). "A computer-enabled solution procedure for food wholesalers' distribution decision in cities with a circular transportation infrastructure." In: *Computers and Operations Research* 36.7, pp. 2201–2209. ISSN: 03050548. DOI: 10.1016/j.cor.2008.08.020.

Huang, Shi-Yu and Jaques Teghem (2012). *Stochastic versus fuzzy approaches to multiobjective mathematical programming under uncertainty*. Vol. 6. Springer Science & Business Media.

Huang, Yixiao et al. (2012). "A study on carbon reduction in the vehicle routing problem with simultaneous pickups and deliveries." In: *Proceedings of 2012 IEEE International Conference on Service Operations and Logistics, and Informatics, SOLI*, pp. 302–307. DOI: 10.1109/SOLI.2012.6273551.

Hvattum, Lars Magnus, Arne Løkketangen, and Gilbert Laporte (2006). "Solving a Dynamic and Stochastic Vehicle Routing Problem with a Sample Scenario Hedging Heuristic." In: *Transportation Science* 40.4, pp. 421–438. ISSN: 0041-1655. DOI: 10.1287/trsc.1060.0166.

Ichoua, Soumia, Michel Gendreau, and Jean Yves Potvin (2003). "Vehicle dispatching with time-dependent travel times." In: *European Journal of Operational Research* 144.2, pp. 379–396. ISSN: 03772217. DOI: 10.1016/S0377-2217(02)00147-9.

– (2006). "Exploiting knowledge about future demands for real-time vehicle dispatching." In: *Transportation Science* 40.2, pp. 211–225. ISSN: 0041-1655. DOI: 10.1287/trsc.1050.0114.

Jaillet, Patrick (1985). "Probabilistic traveling salesman problems." PhD thesis. Massachusetts Institute of Technology.

– (1987). "Stochastic routing problems." In: *Stochastics in Combinatorial Optimization, Word Scientific, Singapore*, pp. 197–213.

– (1988). "A Priori Solution of a Traveling Salesman Problem in Which a Random Subset of the Customers Are Visited." In: *Operations Research* 36.6, pp. 929–936. ISSN: 0030-364X. DOI: 10.1287/opre.36.6.929.

Jaillet, Patrick and A Odoni (1988). "The probabilistic vehicle routing problem." In: *Vehicle routing: methods and studies. North Holland, Amsterdam*.

Jezequel, Antoine (1985). "Probabilistic vehicle routing problems." PhD thesis. Massachusetts Institute of Technology.

Kenyon, Astrid S and David P Morton (2003). "Stochastic vehicle routing with random travel times." In: *Transportation Science* 37.1, pp. 69–82.

Kim, Gitae et al. (2015). "City Vehicle Routing Problem (City VRP): A Review." In: *IEEE Transactions on Intelligent Transportation Systems* 16.4, pp. 1654–1666. ISSN: 15249050. DOI: 10.1109/TITS.2015.2395536. arXiv: 1502.07718.

Kim, Seongmoon, Mark E Lewis, and Chelsea C White (2005). "Optimal Vehicle Routing With Real-Time Traffic Information." In: *IEEE Transactions on Intelligent Transportation Systems* 6.2, pp. 178–188. ISSN: 1524-9050. DOI: 10.1109/TITS.2005.848362.

Kindervater, Gerard A P and Martin W P Savelsbergh (1997). "Vehicle routing: handling edge exchanges." In: *Local search in combinatorial optimization*, pp. 337–360.

King, Alan J AJ and SW Stein W Wallace (2012). *Modeling with Stochastic Programming*. Springer New York. ISBN: 9780387878164.

Kirkpatrick, Scott, C Daniel Gelatt, and Mario P Vecchi (1983). "Optimization by simulated annealing." In: *Science* 220.4598, pp. 671–680.

Kong, Linghe et al. (2012). "Evaluation of Urban Vehicle Routing Algorithms." In: *International Journal of Digital Content Technology and its Applications* 6.23, pp. 790–799. ISSN: 1975-9339. DOI: 10.4156/jdcta.vol6.issue23.92.

Kovacs, Attila a. et al. (2014). "Vehicle routing problems in which consistency considerations are important: A survey." In: *Networks* 64.3, pp. 192–213. ISSN: 00283045. DOI: 10.1002/net.21565.

Kuo, Yiyo (2010). "Using simulated annealing to minimize fuel consumption for the time-dependent vehicle routing problem." In: *Computers and Industrial Engineering* 59.1, pp. 157–165. ISSN: 03608352. DOI: 10.1016/j.cie.2010.03.012.

Kuo, Yiyo and Chi-Chang Wang (2011). "Optimizing the VRP by minimizing fuel consumption." In: *Management of Environmental*

Quality: An International Journal 22.4, pp. 440–450. ISSN: 1477-7835. DOI: 10.1108/14777831111136054.

Kwon, Yong Ju, Young Jae Choi, and Dong Ho Lee (2013). "Heterogeneous fixed fleet vehicle routing considering carbon emission." In: *Transportation Research Part D: Transport and Environment* 23, pp. 81–89. ISSN: 13619209. DOI: 10.1016/j.trd.2013.04.001.

Laporte, Gilbert and François V Louveaux (1993). "The integer L-shaped method for stochastic integer programs with complete recourse." In: *Operations Research Letters* 13.3, pp. 133–142. ISSN: 0167-6377. DOI: 10.1016/0167-6377(93)90002-X.

Laporte, Gilbert and Yves Nobert (1983). "A branch and bound algorithm for the capacitated vehicle routing problem." In: *Operations-Research-Spektrum* 5.2, pp. 77–85.

Laporte, Gilbert, François Louveaux, and Hélène Mercure (1992). "The vehicle routing problem with stochastic travel times." In: *Transportation science* 26.3, pp. 161–170.

Laporte, Gilbert, Francois V Louveaux, and Hélene Mercure (1994). "A priori optimization of the probabilistic traveling salesman problem." In: *Operations Research* 42.3, pp. 543–549.

Laporte, Gilbert, François Louveaux, and Luc van Hamme (2002). "An Integer L-Shaped Algorithm for the Capacitated Vehicle Routing Problem with Stochastic Demands." In: *Operations Research* 50.3, pp. 415–423.

Lei, Hongtao, Gilbert Laporte, and Bo Guo (2011). "The capacitated vehicle routing problem with stochastic demands and time windows." In: *Computers & Operations Research* 38.12, pp. 1775–1783. ISSN: 03050548. DOI: 10.1016/j.cor.2011.02.007.

Lenstra, Jan K and A H G Rinnooy Kan (1979). "Computational complexity of discrete optimization problems." In: *Annals of Discrete Mathematics* 4, pp. 121–140.

Li, Xiangyong, Peng Tian, and Stephen C.H. H Leung (2010). "Vehicle routing problems with time windows and stochastic travel and service times: models and algorithm." In: *International Journal of Production Economics* 125.1, pp. 137–145. ISSN: 09255273. DOI: 10.1016/j.ijpe.2010.01.013.

Liao, Tsai-Yun and Ta-Yin Hu (2011). "An object-oriented evaluation framework for dynamic vehicle routing problems under real-time information." In: *Expert Systems with Applications* 38.10, pp. 12548–12558. ISSN: 09574174. DOI: 10.1016/j.eswa.2011.04.041.

Maggioni, Francesca, Guido Perboli, and Roberto Tadei (2014). "The multi-path traveling salesman problem with stochastic travel costs: Building realistic instances for city logistics applications." In: *Transportation Research Procedia* 3.July, pp. 528–536. ISSN: 23521465. DOI: 10.1016/j.trpro.2014.10.001.

Margot, François (2010). "Symmetry in integer linear programming." In: *50 Years of Integer Programming 1958-2008*. Springer, pp. 647–686.

Martin, Robert C (2002). *Agile software development: principles, patterns, and practices*. Prentice Hall.

Matis, Peter (2010). "Finding a solution for a complex street routing problem using the mixed transportation mode." In: *Transport* 25.1, pp. 29–35. ISSN: 16484142. DOI: 10.3846/transport.2010.05.

Maxwell, Matthew S. et al. (2010). "Approximate dynamic programming for ambulance redeployment." In: *INFORMS Journal on Computing* 22.2, pp. 266–281. ISSN: 10919856. DOI: 10.1287/ijoc.1090.0345.

Melgarejo, Penélope Aguiar, Philippe Laborie, and Christine Solnon (2015). "A time-dependent no-overlap constraint: Application to urban delivery problems." In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. ISBN: 9783319180076. DOI: 10.1007/978-3-319-18008-3_1.

Mendoza, Jorge E and Bruno Castanier (2011). "Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands." In: *Transportation science* 45.3, pp. 346–363.

Miller, C. E., A. W. Tucker, and R. A. Zemlin (1960). "Integer Programming Formulation of Traveling Salesman Problems." In: *Journal of the ACM* 7.4, pp. 326–329. ISSN: 00045411. DOI: 10.1145/321043.321046.

Minis, I, K Mamasis, and V Zeimpekis (2012). "Real-time management of vehicle breakdowns in urban freight distribution." In: *Journal of Heuristics* 18.3, pp. 375–400. ISSN: 13811231. DOI: 10.1007/s10732-011-9191-1.

Mitrović-Minić, Snežana and Gilbert Laporte (2004). "Waiting strategies for the dynamic pickup and delivery problem with time windows." In: *Transportation Research Part B: Methodological* 38.7, pp. 635–655. ISSN: 01912615. DOI: 10.1016/j.trb.2003.09.002.

Morales, JC (2006). "Planning robust freight transportation operations." PhD thesis. Georgia Institute of Technology.

Muñuzuri, Jesus et al. (2013). "Estimating the extra costs imposed on delivery vehicles using access time windows in a city." In: *Computers, Environment and Urban Systems* 41, pp. 262–275. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2012.05.005.

Novoa, C et al. (2006). "A set-partitioning-based model for the stochastic vehicle routing problem." In: *Lehigh University*.

Perboli, G, M Rosano, and L Gobbato (2017). *Parcel delivery in urban areas: do we need new business and operational models for mixing traditional and low-emission logistics?* Tech. rep. CIRRELT 2017-02 Montréal (Canada).

Perboli, G. et al. (2018). "Simulation-optimisation framework for City Logistics: An application on multimodal last-mile delivery." In: *IET Intelligent Transport Systems* 12.4, pp. 262–269. ISSN: 1751956X. DOI: 10.1049/iet-its.2017.0357.

Perboli, Guido, Roberto Tadei, and Daniele Vigo (2011). "The Two-Echelon Capacitated Vehicle Routing Problem: Models and Math-

Based Heuristics." In: *Transportation Science* 45.3, pp. 364–380. ISSN: 0041-1655. DOI: 10.1287/trsc.1110.0368.

Perrier, Nathalie, André Langevin, and Ciro-Alberto Amaya (2008). "Vehicle Routing for Urban Snow Plowing Operations." In: *Transportation Science* 42.1, pp. 44–56. ISSN: 0041-1655. DOI: 10.1287/trsc.1070.0195.

Pessoa, Artur, Marcus Poggi De Aragão, and Eduardo Uchoa (2008). "Robust branch-cut-and-price algorithms for vehicle routing problems." In: *The vehicle routing problem: Latest advances and new challenges*. Springer, pp. 297–325.

Pillac, Victor, Christelle Guéret, and Andrés L. Medaglia (2012). "An event-driven optimization framework for dynamic vehicle routing." In: *Decision Support Systems* 54.1, pp. 414–423. ISSN: 01679236. DOI: 10.1016/j.dss.2012.06.007.

Pillac, Victor et al. (2013). "A review of dynamic vehicle routing problems." In: *European Journal of Operational Research* 225.1, pp. 1–11.

Powell, Warren B (2009). "What you should know about approximate dynamic programming." In: *Naval Research Logistics (NRL)*.

Psaraftis, Harilaos N (1980). "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem." In: *Transportation Science* 14.2, pp. 130–154.

– (1995). "Dynamic vehicle routing: Status and prospects." In: *annals of Operations Research* 61.1, pp. 143–164.

Psaraftis, Harilaos N, Min Wen, and Christos A Kontovas (2015). "Dynamic vehicle routing problems: Three decades and counting." In: *Networks*.

Quak, Hans J and M. B. M. de Koster (2009). "Delivering Goods in Urban Areas: How to Deal with Urban Policy Restrictions and the Environment." In: *Transportation Science* 43.2, pp. 211–227. ISSN: 0041-1655. DOI: 10.1287/trsc.1080.0235.

Rabideau, Gregg and Ed Benowitz (2017). "Prototyping an Onboard Scheduler for the Mars 2020 Rover." In: *Proceedings of the International Workshop on Plaanning and Scheduling for Space, IWPSS* Iwpss 2017.

Ritzinger, Ulrike, Jakob Puchinger, and Richard F Hartl (2016). "A survey on dynamic and stochastic vehicle routing problems." In: *International Journal of Production Research* 54.1, pp. 215–231. ISSN: 0020-7543. DOI: 10.1080/00207543.2015.1043403.

Rossi, F A and I Gavioli (1987). "Aspects of heuristic methods in the probabilistic traveling salesman problem." In: *Advanced school on stochastics in combinatorial optimization*, pp. 214–227.

Saberi, Meead and I. Omer Verbas (2012). "Continuous Approximation Model for the Vehicle Routing Problem for Emissions Minimization at the Strategic Level." In: *Journal of Transportation Engineering* 138.11, pp. 1368–1376. ISSN: 0733-947X. DOI: 10.1061/(ASCE)TE.1943-5436.0000442.

Saint-Guillain, Michael (2019). "Robust Operations Management on Mars." In: *29th International Conference on Automated Planning and Scheduling (ICAPS'19)*. Berkeley, CA, USA.

Saint-Guillain, Michael, Yves Deville, and Christine Solnon (2015). "A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW." In: *12th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2015)*. Springer International Publishing, pp. 357–374.

Saint-Guillain, Michael, Christine Solnon, and Yves Deville (2017). "The Static and Stochastic VRP with Time Windows and both random Customers and Reveal Times." In: *Evolutionary Algorithms and Meta-heuristics in Stochastic and Dynamic Environments*. Springer International Publishing, pp. 110–127. ISBN: 9783319557922. DOI: 10.1007/978-3-319-55792-2.

Santos, Luís, João Coutinho-Rodrigues, and John R. Current (2008). "Implementing a multi-vehicle multi-route spatial decision support system for efficient trash collection in Portugal." In: *Transportation Research Part A: Policy and Practice* 42.6, pp. 922–934. ISSN: 09658564. DOI: 10.1016/j.tra.2007.08.009.

Schilde, Michael, Karl F Doerner, and Richard F Hartl (2011). "Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports." In: *Computers and Operations Research* 38.12, pp. 1719–1730. ISSN: 03050548. DOI: 10.1016/j.cor.2011.02.006.

Schmid, Verena (2012). "Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming." In: *European Journal of Operational Research* 219.3, pp. 611–621. ISSN: 03772217. DOI: 10.1016/j.ejor.2011.10.043.

Schüpbach, Kaspar and Rico Zenklusen (2013). "An adaptive routing approach for personal rapid transit." In: *Mathematical Methods of Operations Research* 77.3, pp. 371–380. ISSN: 14322994. DOI: 10.1007/s00186-012-0403-8.

Secomandi, Nicola (2000). "Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands." In: *Computers & Operations Research* 27.

Secomandi, Nicola and F Margot (2009). "Reoptimization approaches for the vehicle-routing problem with stochastic demands." In: *Operations Research* 57, pp. 214–230.

Séguin, René (1994). "Problemes stochastiques de tournees de vehicules." In: *Université de Montréal*.

Shapiro, Alexander, Darinka Dentcheva, and Andrzej P Ruszczy\'nski (2009). *Lectures on stochastic programming: modeling and theory*. Vol. 9. SIAM.

Shaw, Paul (1998). "Using constraint programming and local search methods to solve vehicle routing problems." In: *Principles and Practice of Constraint Programming—CP98*. Springer, pp. 417–431.

Sifa, Zheng et al. (2011). "Urban pickup and delivery problem considering time-dependent fuzzy velocity." In: *Computers and Industrial Engineering* 60.4, pp. 821–829. ISSN: 03608352. DOI: 10.1016/j.cie.2011.01.020.

Slyke, RM Van and R Wets (1969). "L-shaped linear programs with applications to optimal control and stochastic programming." In: *SIAM Journal on Applied Mathematics* 17.4, pp. 638–663.

Solomon, Marius M (1987). "Algorithms for the vehicle routing and scheduling problems with time window constraints." In: *Operations research* 35.2.

Sungur, Ilgaz and Yingtao Ren (2010). "A model and algorithm for the courier delivery problem with uncertainty." In: *Transportation science* 44.2, pp. 193–205.

Tadei, Roberto, Guido Perboli, and Francesca Perfetti (2014). "The multi-path Traveling Salesman Problem with stochastic travel costs." In: *EURO Journal on Transportation and Logistics*, pp. 1–21. ISSN: 2192-4376, 2192-4384. DOI: 10.1007/s13676-014-0056-2.

Taillard, Éric et al. (1997). "A tabu search heuristic for the vehicle routing problem with soft time windows." In: *Transportation Science* 31.2, pp. 170–186.

Taniguchi, Eiichi and Yasushi Kakimoto (2004). "Modelling effects of e-commerce on urban freight transport." In: *Logistics Systems for Sustainable Cities*. Chap. Chapter 10, pp. 135–146. DOI: 10.1108/9780080473222-010.

Taniguchi, Eiichi and Russell Thompson (2002). "Modeling City Logistics." In: *Transportation Research Record* 1790.1, pp. 45–51. ISSN: 0361-1981. DOI: 10.3141/1790-06.

Toth, Paolo and Daniele Vigo (2014). *Vehicle Routing: Problems, Methods, and Applications*. Vol. 18. SIAM.

Turner, Rodney J et al. (2010). *Perspectives on projects*. Routledge.

URBeLOG (2015). *Project Web Site*.

Ubeda, S., F. J. Arcelus, and J. Faulin (2011). "Green logistics at Eroski: A case study." In: *International Journal of Production Economics* 131.1, pp. 44–51. ISSN: 09255273. DOI: 10.1016/j.ijpe.2010.04.041.

Van Hentenryck, Pascal and Russell W Bent (2009). *Online stochastic combinatorial optimization*. The MIT Press.

Van Hentenryck, Pascal, Russell W Bent, and Eli Upfal (2009). *Online stochastic optimization under time constraints*. Vol. 177. 1, pp. 151–183. ISBN: 1047900906055. DOI: 10.1007/s10479-009-0605-5.

Verweij, Bram et al. (2003). "The sample average approximation method applied to stochastic routing problems: a computational study." In: *Computational Optimization and Applications* 24.2-3, pp. 289–333.

Vigo, Daniele (1996). "A heuristic algorithm for the asymmetric capacitated vehicle routing problem." In: *European Journal of Operational Research* 89.1, pp. 108–126. ISSN: 03772217. DOI: 10.1016/S0377-2217(96)90060-0.

Voccia, Stacy A, Ann M Campbell, and Barrett W Thomas (2013). "The probabilistic traveling salesman problem with time windows." In: *{EURO} Journal on Transportation and Logistics* 2.1-2, pp. 89–107. ISSN: 2192-4376, 2192-4384. DOI: 10.1007/s13676-013-0018-0.

Waters, C D J (1989). "Vehicle-scheduling problems with uncertainty and omitted customers." In: *Journal of the Operational Research Society*, pp. 1099–1108.

Watson, Jean-Paul, David L Woodruff, and William E Hart (2012). "PySP: modeling and solving stochastic programs in Python." In: *Mathematical Programming Computation* 4.2, pp. 109–149.

Wets, Roger J-B (1974). "Stochastic programs with fixed recourse: The equivalent deterministic program." In: *SIAM review* 16.3, pp. 309–339.

Weyland, Dennis, Roberto Montemanni, and Luca Maria Gambardella (2013). "An improved heuristic for the probabilistic traveling salesman problem with deadlines based on GPGPU." In: *Computer Aided Systems Theory-EUROCAST 2013*. Springer, pp. 332–339.

Wilson, Nigel H M and Neil J Colvin (1977). *Computer control of the Rochester dial-a-ride system*. Massachusetts Institute of Technology, Center for Transportation Studies.

Woensel, T van (2017). *{DATA2MOVE} Initiative*.

Wolsey, Laurence A (1998). *Integer programming*. Wiley.

Xiao, Yiyong et al. (2012). "Development of a fuel consumption optimization model for the capacitated vehicle routing problem." In: *Computers and Operations Research* 39.7, pp. 1419–1431. ISSN: 03050548. DOI: 10.1016/j.cor.2011.08.013.

Yang, Pei-Ying et al. (2013). "Minimizing Carbon Emissions through Vehicle Routing and Scheduling in the Shuttle Service of Picking up and Delivering Customers to the Airport." In: *Acta Automatica Sinica* 39.4, pp. 424–432. ISSN: 18741029. DOI: 10.1016/s1874-1029(13)60042-7.