Université de Montréal

Metaheuristics for Vehicle Routing Problems : New Methods and Performance Analysis

par

Fernando Obed Guillen Reyes

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique, option recherche opérationnelle

21 février 2024

© Fernando Obed Guillen Reyes, 2024

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Metaheuristics for Vehicle Routing Problems : New Methods and Performance Analysis

présentée par

Fernando Obed Guillen Reyes

a été évaluée par un jury composé des personnes suivantes :

Fabian Bastin
(président-rapporteur)

Jean-Yves Potvin
(directeur de recherche)

Michel Gendreau
(codirecteur)

Thibaut Vidal
(codirecteur)

Margarida Carvalho
(membre du jury)

Jacques Renaud
(examinateur externe)

Jacques Bélair
(représentant du doyen de la FESP)

Résumé

Cette thèse s'intéresse au problème classique de tournées de véhicules avec contraintes de capacité (*CVRP* pour Capacitated Vehicle Routing Problem) ainsi qu'une variante beaucoup plus complexe, soit le problème de tournées de véhicules dépendant du temps avec fenêtres de temps et points de transfert défini sur un réseau routier ($TDVRPTWTP_{RN}$ pour Time-Dependent Vehicle Routing Problem with Time Windows and Transfer Points on a Road Network). Dans le premier article, le $TDVRPTWTP_{RN}$ est résolu en adaptant une métaheuristique qui représente l'état de l'art pour le *CVRP*, appelé Slack Induction for String Removals (*SISR*). Cette métaheuristique fait appel au principe "détruire et reconstruire" en retirant des séquences de clients consécutifs dans les routes de la solution courante et en réinsérant ensuite ces clients de façon à créer une nouvelle solution. Le problème est défini sur un réseau routier où différents chemins alternatifs peuvent être utilisés pour se déplacer d'un client à l'autre. De plus, le temps de parcours sur chacun des arcs du réseau n'est pas fixe, mais dépend du moment où le véhicule quitte le sommet origine. S'inspirant de problèmes rencontrés en logistique urbaine, nous considérons également deux types de véhicules, de petite et grande capacité, où les grands véhicules sont interdits de passage au centre-ville. Ainsi, les clients du centre-ville ne peuvent être servis que suite au transfert de leur demande d'un grand à un petit véhicule à un point de transfert. Comme un point de transfert n'a pas de capacité, une problématique de synchronisation apparaît quand un grand véhicule doit y rencontrer un ou plusieurs petits véhicules pour leur transférer une partie de son contenu. Contrairement aux problèmes stricts de tournées de véhicules à deux échelons, les grands véhicules peuvent aussi servir des clients localisés à l'extérieur du centre-ville. Comme le problème abordé est beaucoup plus complexe que le *CVRP*, des modifications importantes ont dû être apportées à la métaheuristique *SISR* originale. Pour évaluer la performance de notre algorithme, un ensemble d'instances tests a été généré à partir d'instances existantes pour le $TDVRPTW_{RN}$. Les réseaux omt été divisés en trois régions : centre-ville, frontière et extérieur. Le centre-ville et l'extérieur sont respectivemnt les royaumes des petits et grands véhicules, tandis que la frontière (où l'on retrouve les points de transfert) peut être visité par les deux types de véhicules. Les résultats numériques montrent que la métaheuristique proposée exploite les opportunités

d'optimiser une solution en déplaçant autant que possible les clients neutres, soit ceux qui peuvent être servis indifféremment par un petit ou un grand véhicule, des routes des petits véhicules vers les routes des grands véhicules, réduisant ainsi les coûteuses visites aux points de transfert.

Les deuxième et troisième article s'intéressent à des concepts plus fondamentaux et font appel au problème plus simple du *CVRP* pour les évaluer. Dans le second article, un étude expérimentale est conçue afin d'examiner l'impact de données (distances) imprécises sur la performance de différents types d'heuristiques, ainsi qu'une méthode exacte, pour le *CVRP*. À cette fin, différents niveaux d'imprécision ont été introduits dans des instances tests classiques pour le *CVRP* avec 100 à 1 000 clients. Nous avons observé que les meilleures métaheuristiques demeurent les meilleures, même en présence de hauts niveaux d'imprécision, et qu'elles ne sont pas affectées autant par les imprécisions qu'une heuristique simple. Des expériences avec des instances réelles ont mené aux mêmes conclusions.

Le troisième article s'intéresse à l'intégration de l'apprentissage automatique dans la métaheuristique *SISR* qui représente l'état de l'art pour le *CVRP*. Dans ce travail, le principe "détruire et reconstruire" au coeur de *SISR* est hybridé avec une méthode d'apprentissage par renforcement qui s'inspire des systèmes de colonies de fourmis. L'apprentissage automatique a pour but d'identifier les arêtes les plus intéressantes, soit celles qui se retrouvent le plus fréquemment dans les solutions de grande qualité précédemment rencontrées au cours de la recherche. L'inclusion de telles arêtes est alors favorisé lors de la réinsertion des clients ayant été retirés de la solution par le mécanisme de destruction. Les instances utilisées pour tester notre approche hybride sont les mêmes que celles du second article. Nous avons observé que notre algorithme ne peut produire que des solutions légèrement meilleures que la métaheuristique *SISR* originale, celle-ci étant déjà quasi-optimale.

Mots clés : Problème de tournées de véhicules avec contraintes de capacité, temps de parcours dépendants du temps, fenêtres de temps, points de transfert, données inexactes, apprentissage par renforcement, métaheuristique.

Abstract

This thesis is concerned both with the classical Capacitated Vehicle Routing Problem (*CVRP*) and a much more complex variant called the Time-Dependent Vehicle Routing Problem with Time Windows and Transfer Points on a Road Network ($TDVRPTWTP_{RN}$). In the first paper, the $TDVRPTWTP_{RN}$ is solved by adapting a state-of-the-art metaheuristic for the *CVRP*, called Slack Induction for String Removals (*SISR*). This metaheuristic is based on the ruin and recreate principle and removes strings of consecutive customers in the routes of the current solution and then reinserts the removed customers to create a new solution. The problem is formulated in a full road network where different alternative paths can be used to go from one customer to the next. Also, the travel time on each arc of the road network is not fixed, but depends on the departure time from the origin node. Motivated from city logistics applications, we also consider two types of vehicles, large and small, with large vehicles being forbidden from the downtown area. Thus, downtown customers can only be served through a transfer of their goods from large to small vehicles at designated transfer points. Since transfer points have no capacity, synchronization issues arise when a large vehicle must meet one or more small vehicles to transfer goods. As opposed to strict two-echelon *VRPs*, large vehicles can also directly serve customers that are outside of the downtown area. Given that the $TDVRPTWTP_{RN}$ is much more complex than the *CVRP*, important modifications to the original *SISR* metaheuristic were required. To evaluate the performance of our algorithm, we generated a set of test instances by extending existing instances of the $TDVRPTW_{RN}$. The road networks are divided into three regions: downtown, boundary and outside. The downtown and outside areas are the realm of small and large vehicles, respectively, while the boundary area that contains the transfer points can be visited by both small and large vehicles. The results show that the proposed metaheuristic exploits optimization opportunities by moving as much as possible neutral customers (which can be served by either small or large vehicles) from the routes of small vehicles to those of large vehicles, thus avoiding costly visits to transfer points.

The second and third papers examine more fundamental issues, using the classical *CVRP* as a testbed. In the second paper, an experimental study is designed to examine the

impact of inaccurate data (distances) on the performance of different types of heuristics, as well as one exact method, for the *CVRP*. For this purpose, different levels of distance inaccuracies were introduced into well-known benchmark instances for the *CVRP* with 100 to 1,000 customers. We observed that the best state-of-the-art metaheuristics remain the best, even in the presence of high inaccuracy levels, and that they are not as much affected by inaccuracies when compared to a simple heuristic. Some experiments performed on real-world instances led to the same conclusions.

The third paper focuses on the integration of learning into the state-of-the-art *SISR* for the *CVRP*. In this work, the ruin and recreate mechanism at the core of *SISR* is enhanced by a reinforcement learning technique inspired from ant colony systems. The learning component is aimed at identifying promising edges, namely those that are often found in previously encountered high-quality solutions. The inclusion of these promising edges is then favored during the reinsertion of removed customers. The benchmark instances of the second paper were also used here to test the new hybrid algorithm. We observed that the latter can produce only slightly better solutions than the original *SISR*, due to the quasi-optimality of the original solutions.

Keywords: Capacitated vehicle routing problem, time-dependent travel times, time windows, transfer points, inaccurate data, reinforcement learning, metaheuristic.

Contents

Résumé 5

Abstract 7

List of Tables 13

List of Figures 15

List of acronyms and abbreviations 17

Acknowledgments 19

Chapter 1. Introduction 21

1.1. Vehicle routing problems 21

1.2. Ruin and Recreate metaheuristic 23

1.3. Papers of the thesis 23

First Article. A Metaheuristic for a Time-Dependent Vehicle Routing Problem with Time Windows, Two Vehicle Fleets and Synchronization on a Road Network 25

1. Introduction 27

2. Literature review 28

2.1. Time-dependent VRPs 28

2.2. VRPs with intermediate facilities 30

3. Problem Definition 31

4. SISR for the CVRP 34

4.1. Ruin 36

4.2. Recreate 37

5. Time dependency 38

5.1. Time-dependent travel times 38

5.2. Dominant shortest-path structure 38
5.3. Synchronization at a transfer point 40
5.4. Time bounds 41
5.4.1. Green route 41
5.4.2. Black route 43
5.5. Interaction among routes 44

6. SISR for the $TDVRPTWTP_{RN}$ 45
6.1. Initial solution 47
6.2. Ruin 47
6.2.1. Ruining black routes 48
6.2.2. Ruining green routes 48
6.3. Recreate 51
6.3.1. Insertion in black routes 51
6.3.2. Insertion in green routes 52

7. Computational experiments 55
7.1. Test instances 57
7.2. Parameter tuning 60
7.3. Results on test intances 62
7.4. Synchronization 64
7.5. Customer distributions 65
7.6. Time windows 66
7.7. Scenarios 66
7.8. Number of transfer points 66

8. Conclusion 67

Appendix 69

Second Article. Impact of Distance Data Inaccuracies on Vehicle Routing Algorithms: An Experimental Study 71

1. Introduction 73

2. Related works 74

3. Experimental setting 75
3.1. Solution Algorithms 76
3.2. Test Instances 77

3.3. Distance Inaccuracies ... 78
4. Experimental Study ... 79
4.1. Performance of the heuristics on set X instances with inaccurate distances ... 79
4.2. Performance of an exact algorithm on set X with inaccurate distances ... 83
4.3. Performance of heuristics on real-world instances with inaccurate distances ... 84
5. Conclusions ... 86
Appendix ... 87
Third Article. A Ruin&Recreate Ant Colony System Algorithm for the Capacitated Vehicle Routing Problem ... 89
1. Introduction ... 91
2. Literature review ... 92
2.1. Neural Networks ... 92
2.2. Clustering ... 93
2.3. Support Vector Machines and Learning Automata ... 93
2.4. Reinforcement Learning ... 93
2.5. Ant Colony Optimization ... 94
2.6. Others ... 95
3. Ruin&Recreate metaheuristic ... 96
4. Ant Colony System ... 100
5. Hybrid RR-ACS algorithm ... 101
6. Computational results ... 104
6.1. RR Results ... 104
6.2. RR-ACS Parameter Settings ... 105
6.3. RR-ACS Results ... 105
7. Conclusions ... 106
Conclusions ... 111
Bibliography ... 113

List of Tables

1 Speed multipliers for (a) black vehicles and (b) green vehicles under scenario S_I . 60

2 Speed multipliers for (a) black vehicles and (b) green vehicles under scenario S_{II} 60

3 Impact of parameter values on solution quality and computation times (in hours) 62

4 Solution cost and computation time in hours for each subset of instances 63

5 Average best improvements and average improvements for each subset of instances. 64

6 Minimum, maximum and average values of ρ_s for each subset of instances 65

7 Average gaps between average solution costs with 10 transfer points and $k = 8$, 6, 4 transfer points 68

8 Solution cost and computation time in hours for each instance with narrow time windows. 69

9 Solution cost and computation time in hours for each instance with wide time windows. 70

10 Global average gaps and standard deviations for each method and each noise level 82

11 Global average gaps and corresponding standard deviations for each method and each noise level 84

12 p-values of Wilcoxon signed-rank test 85

13 Global average gaps and corresponding standard deviations on real instances for each heuristic and noise level. 85

14 Best-known solutions on Set-X instances 87

15 Best-known solutions on real-world instances 88

16 Average costs and times obtained with RR and SISR, for medium-size Uchoa instances. 107

17 Average costs and times obtained with RR and SISR, for large-size Uchoa instances. 108

18 Average costs and times obtained with RR-ACS and SISR, for medium-size Uchoa instances. 109

19 Average costs and times obtained with RR-ACS and SISR, for large-size Uchoa instances. 110

List of Figures

1 An example of a solution to the $TDVRPTWTP_{RN}$ 33

2 Examples of the two SISR ruin operators : (a) String (b) Split-String..... 38

3 (a) Travel speed function of arc (i, j) (b) Corresponding travel time function assuming that arc (i, j) is of length 4. 39

4 (a) Three different fastest paths between two nodes obtained at different time points using a time-dependent Dijkstra's algorithm (b) Corresponding dominant shortest path structure..... 40

5 Example of forward and backward propagations when customer i is inserted in a route: (a) forward propagation (b) backward propagation..... 45

6 Example of $String^b$ where four customers are removed. First, the transfer points are set apart, then a string of cardinality four is chosen and the corresponding customers are removed. 49

7 Example of $String^g$ where two customers are removed. 50

8 Phase I : a transfer point tp (already present in the black route) followed by customer i are inserted after customer j^g in the green route; (a) and (b) show the black and green routes before and after the insertion, respectively..... 55

9 Phase II: a transfer point tp (already present in the black route) followed by customer i are inserted in a new green route; (a) and (b) show the black and green routes before and after the insertion, respectively. 55

10 Phase III: a new transfer point tp is inserted in the black route; this transfer point followed by customer i are inserted after customer j^g in the green route; (a) and (b) show the black and green routes before and after the insertion, respectively. . 56

11 Phase IV: a new transfer point tp is inserted in the black route; this transfer point followed by customer i are inserted in a new green route; (a) and (b) show the black and green routes before and after the insertion, respectively..... 56

12 A new black and a new green route are created for a single green or neutral customer 56

13 Example of an instance generated with road network RN_3; 59
14 Convergence curves observed for instances with 200 customers generated with road network RN_4 under scenario S_I. 62
15 Average gaps obtained on each original instance for each noise level 80
16 Global average gaps with increasing values of σ for each heuristic. 81
17 Standard deviation of global average gaps (large dots) with increasing values of σ for each heuristic. 82
18 Global average gaps with the optimum for each method with increasing values of σ. On the left, GORT, HGS and SISR are compared with BCP; on the right, using a different scale, only HGS and SISR are compared with BCP. 83
19 Global average gaps obtained on real instances with increasing values of σ for each heuristic. 86

List of acronyms and abbreviations

VRP	Vehicle routing problem
CVRP	Capacitated vehicle routing problem
TDVRP	Time-dependent vehicle routing problem
TDVRPTW	Time-dependent vehicle routing problem with time windows
$TDVRPTWTP_{RN}$	Time-dependent vehicle routing problem with time windows and transfer points on a road network
DSPS	Dominant shortest path structure
CW	Clarke & Wright algorithm
GORT	Google OR-Tools open source software for optimization
HGS	Hybrid genetic search algorithm
SISR	Slack induction by string removals algorithm

BCP	Exact branch-cut-and-price algorithm
ACS	Ant colony system

Acknowledgments

To my father Fernando and my mother Rosa Maria, to my brothers Nahum and Misael, to my sisters Claudia and Paulina (r.i.p.), to my grandparents Rodolfo (r.i.p.), Maria, Teofilo and Guadalupe, without their support, I would not have been able to achieve my academic goals. To my friends and family members who supported me during this process

I also want to thank my advisor Jean-Yves Potvin, and my co-advisors Michel Gendreau and Thibaut Vidal, for the opportunity they gave me to work with them, as well as, all the support and knowledge they gave me during these years.

Chapter 1

Introduction

In the last decades, vehicle routing problems (*VRPs*) have attracted the attention of researchers due to their numerous applications in distribution systems. At the beginning of the century, Toth and Vigo argued in [**80**] that the development of new and efficient methods for transportation problems had produced savings between 5% and 20% in global transportation costs. Recently, with the considerable increase in online purchases and the need to reduce greenhouse gas emissions, there is an even stronger incentive to develop better algorithms and strategies to solve *VRPs*. Although multiple problem-solving methods have been proposed over the years, it is still difficult to optimally solve instances of realistic sizes and the recourse to heuristic approaches, in particular metaheuristics, is very common. The complexity of *VRPs* lies in their combinatorial nature and associated complexity, which increases exponentially with problem size.

In the following, we briefly introduce *VRPs* and the particular problems that we address in this thesis.

1.1. Vehicle routing problems

The *VRP* was originally introduced and mathematically formulated in 1959 by Dantzig and Ramser [**21**]. In this work, the authors extend the classical Traveling Salesman Problem (*TSP*) by adding capacity constraints to the trucks used to deliver gas to gas stations (customers). The objective is then to create routes that start and end at a terminal and serve each customer exactly once, while minimizing the total distance traveled by the vehicles. This problem is now known as the capacitated *VRP* (*CVRP*).

Both exact and heuristic approaches have been proposed to solve *VRPs*. In [**52**], the author classifies the main formulations (e.g., vehicle flow and commodity flow formulations) and exact methods for solving it (e.g., branch-and-bound, dynamic programming). In [**53**],

the authors propose a classification of the multiple heuristics, including metaheuristics, that have been reported through the years. According to [**53**], the first attempts to solve the *VRP* started in the 60's, when some relatively simple heuristics were proposed, like nearest neighbor, insertion and savings heuristics [**16**]. The development of exact algorithms started in the 80's, followed by the advent of metaheuristics in the 90's. Although exact algorithms guarantee optimality, they can hardly solve instances with more than 200 customers. Conversely, heuristics and metaheuristics do not guarantee optimality, but state-of-the-art metaheuristics can now produce high-quality solutions on instances of large size in a reasonable amount of time. This trade off between optimality and computation time is particularly relevant in large-scale real-world problems that involve thousands of nodes. A complete description and classification of exact algorithms, heuristics and metaheuristics for *VRPs* can also be found in [**30, 68, 80**].

The *CVRP* is the simplest and more studied problem, with only a capacity constraint imposed on the vehicles. However, many more complex variants have been proposed over the years by considering additional attributes and constraints aimed at modeling more closely different real-world applications. A detailed description of the best known variants of the *VRP* can be found in [**68, 80**], for example the *VRP* with time windows (*VRPTW*) that introduces scheduling issues by associating a time window for service with each customer.

In this thesis, we address a very complex variant of the *VRP*, which extends the time-dependent vehicle routing problem (*TDVRP*). In the latter, the travel time between each pair of nodes is not fixed but depends on the departure time from the origin node. To the best of our knowledge, the first paper about a *TDVRP* is found in [**5**], where an adapted version of the savings heuristic is presented. A little bit later, the *TDVRP* with time windows was addressed in [**44, 58**]. However, the First-In-First-Out (*FIFO*) property was not satisfied in these previous works. This property establishes that if two vehicles travel along the same arc, the vehicle that departs earlier from the origin node must arrive first at the destination node (which is common sense). Almost ten years later, the authors in [**45**] finally formulated a model satisfying the *FIFO* property by considering time-dependent travel speeds. The *TDVRP* has attracted the interest of researchers in recent years, due to the more realistic time-dependent travel times that can model congestion, but the authors in [**30**] note that the time-dependent attribute is present in only 3.6% of *VRP* papers published between 2009 to 2017.

1.2. Ruin and Recreate metaheuristic

As mentioned before, multiple heuristics and metaheuristics have been developed for solving *VRPs* [**52, 68, 80**]. In [**80**], the authors identify six different types of metaheuristics: simulated annealing, deterministic annealing, tabu search, genetic algorithms, ant systems and neural networks. Since then, many new metaheuristics have appeared, one of them being the *Ruin and Recreate* metaheuristic, originally introduced in [**72**]. In this work, three *Ruin* methods that remove customers from the current solution in different ways are proposed. The removed customers are then reinserted (so as to produce a new solution) by a *Recreate* method. In [**70**], the ruin and recreate principle is exploited to solve a pickup and delivery problem with time windows. The proposed methodology, where several operators are used, is called the Adaptive Large Neighborhood Search (*ALNS*). Here, the choice of the ruin and recreate operators at each iteration is not random, but depends on their previous performance, using a dynamic weighting scheme. Also, a simulated annealing criterion is used to accept or reject any new solution produced through the ruin and recreate process. A very recent variant is proposed in [**15**], known as Slack Induction of String Removals (*SISR*), where strings of consecutive customers are removed from the routes. This metaheuristic has proven to be state-of-the art for the *CVRP*.

1.3. Papers of the thesis

The three next chapters correspond to the three papers produced from my thesis work. In the first paper, a very challenging problem called the time-dependent vehicle routing problem with time windows and transfer points on a road network ($TDVRPTWTP_{RN}$) is addressed by adapting a state-of-the-art metaheuristic for the classical *CVRP*. This metaheuristic, known as Slack Induction for String Removals (*SISR*), is based on the ruin-and-recreate principle where strings of consecutive customers are first removed from routes in the current solution. The removed customers are then reinserted to obtain a different, hopefully better, solution.

The second and third papers use the classical *CVRP* as a testbed. The second paper is an empirical computational study aimed at evaluating the impact of inaccurate data on the performance of different problem-solving methods for the *CVRP*, including *SISR*. The third paper explores the integration of reinforcement learning into *SISR*. Inspired from ant colony systems, the idea is to favor edges frequently found in good solutions when reinserting customers during the recreate phase.

Thus, the intention at the core of this thesis is to explore cutting-edge techniques and problems in the context of vehicle routing: in the first paper, the handling of a complex problem inspired from city logistics, involving the consideration of congestion in downtown areas and synchronization of vehicles at transfer points, leading to the development of the first metaheuristic to address this problem; in the second paper, the handling of inaccurate data by different optimization methods, an intriguing subject that has been mostly ignored in operations research; and, in the third paper, the combination of machine learning and optimization methods, considered through the integration of an ant-based reinforcement learning mechanism into a state-of-the art metaheuristic for the *CVRP*.

First Article.

A Metaheuristic for a Time-Dependent Vehicle Routing Problem with Time Windows, Two Vehicle Fleets and Synchronization on a Road Network

by

Fernando Obed Guillen Reyes[1], Jean-Yves Potvin[2], and Michel Gendreau[3]

([1]) Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

([2]) Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

([3]) Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

This article was submitted to the journal Computers & Operations Research.

The main contributions of Fernando Obed Guillen Reyes for this articles are presented. He designed, with the help of his advisors, a new version of a state-of-the-art metaheuristic to solve the time-dependent vehicle routing problem with time windows and transfer points formulated in a road network. Then, he designed with the help of his advisors, a computational study based on test instances derived from benchmark instances for the time-dependent vehicle routing problem with time windows. He also wrote the first draft of the paper.

Résumé. Dans cet article, nous considérons une extension du problème de tournées de véhicules sur un réseau routier avec fenêtres de temps et temps de parcours qui dépendent du moment de la journée, en introduisant deux types de véhicules de petite et grande dimension pour servir les clients. En s'inspirant des problématiques rencontrées en logistique urbaine, les véhicules de grande dimension sont interdits au centre-ville. Ainsi, les marchandises doivent être transférées des grands aux petits véhicules pour desservir les clients situés au centre-ville. Ceci mène à des considérations de synchronisation aux points de transfert qui sont des points bien identifiés dans le réseau, mais dépourvus de capacité. Par ailleurs, le problème n'est pas un problème pur à deux échelons, étant donné que les clients situés à l'extérieur du centre-ville peuvent être desservis directement par les grands véhicules. Le problème se complique encore davantage lorsque l'on doit tenir compte des temps de parcours sur les arcs du réseau routier qui dépendent du moment de la journée et qui servent à modéliser les périodes de congestion. Pour résoudre ce problème complexe, nous proposons une adaptation de la métaheuristique Slack Induction by String Removals, qui représente l'état de l'art pour le problème de tournées de véhicules avec contraintes de capacité. Des résultats expérimentaux sur des instances tests présentant différentes caractéristiques démontrent empiriquement la capacité d'optimisation de cette nouvelle métaheuristique pour un problème qui est beaucoup plus complexe que le problème canonique de tournées de véhicules avec contraintes de capacité.

Mots clés : Problème de tournées de véhicules, réseau routier, temps de parcours dépendants du temps, fenêtres de temps, points de transfert, synchronisation, metaheuristique, slack induction by string removals

Abstract. In this work, we extend the time-dependent vehicle routing problem with time windows on a road network by considering two types of vehicles, large and small, to serve customers. Motivated from city logistics applications, large vehicles are forbidden from the downtown area. Accordingly, goods must be transferred from large to small vehicles to serve downtown customers. This leads to synchronization issues at transfer points, which are special locations without storage capacity. The problem is not a pure two-echelon vehicle routing problem, since customers outside of the downtown area can be served directly by large vehicles. The problem is further compounded by the presence of time-dependent travel times that are defined on the arcs of the road network and are used to model congestion periods. To solve this complex problem, we propose an adaptation of the Slack Induction by String Removals metaheuristic, which is state-of-the-art for the classical capacitated vehicle routing problem. Computational results on a set of test instances with different characteristics empirically demonstrate the optimization capabilities of this new metaheuristic on a problem which is much more complex than the capacitated vehicle routing problem.
Keywords: Vehicle Routing Problem, Road Network, Time-dependent, Time Windows, Transfer Points, Synchronization, Metaheuristic, Slack Induction by String Removals.

1. Introduction

Although the vehicle routing problem (VRP) has been widely studied for a long time, time-dependent variants have spurred the interest of researchers only recently. Time-dependency is an important issue, since the time to travel from one point to another in a network often depends on the departure time (c.f., rush hours). Furthermore, not only does the time to travel along a path between two customers may change depending on the departure time, but even the best path to use may also change. Thus, recent studies have exploited the additional information available in a road network to account for multiple possible paths between two customers, which is often referred to as the time-dependent vehicle routing problem with time windows on a road network ($TDVRPTW_{RN}$). In this paper, we consider an extension of this problem where both large (black) and small (green) vehicles are involved and where some parts of the road network are forbidden to one type of vehicles or the other. For example, the downtown area is not accessible to large vehicles, whereas areas far from downtown are not accessible to small vehicles (e.g., bicycles). Since the goods to be delivered are initially loaded in large vehicles, a customer located in an area not accessible to them can only be served through a transfer of its demand from a large to a small vehicle. This transfer takes place at special locations with no storage capacity, known as transfer points (TPs). This also leads to synchronization issues between the two types of vehicles at transfer points. In the following, this complex delivery problem will be referred to as the $TDVRPTW_{RN}$ with transfer points or $TDVRPTWTP_{RN}$.

Our problem needs to be distinguished from problems with intermediate facilities, with or without storage capacity, since there is no facility as such to transfer goods. It must also

be distinguished from two- or multi-echelon VRPs where vehicles are organized into a strict hierarchical structure to deliver goods to customers. In our problem, black vehicles can very well serve customers directly, as long as they do not belong to forbidden areas. Our contribution lies in the adaptation of a state-of-the-art metaheuristic for the capacitated VRP (CVRP) for a much more complex problem that involves two types of vehicles, three types of customers, time-dependent travel times and synchronization issues between the two types of vehicles to transfer loads at transfer points. As far as we know, this problem has never been addressed in the literature.

In the following, Section 2 first reviews problems related to ours, namely time-dependent VRPs and VRPs with intermediate facilities. Section 3 then precisely describes our problem. The original implementation of the metaheuristic Slack Induction by String Removals (SISR) for solving the CVRP is described in Section 4. Then, Section 5 introduces time issues that arise in the $TDVRPTWTP_{RN}$, in particular calculation of time bounds to check insertion feasibility in constant time and synchronization between black and green vehicles at transfer points. Specific modifications to the original SISR implementation that are required to address the much more complex $TDVRPTWTP_{RN}$ are reported in Section 6. Then, computational results obtained on test instances derived from a benchmark for the $TDVRPTW_{RN}$ are reported. Finally, a conclusion follows.

2. Literature review

Two main features of our problem are time-dependent travel times and the presence of intermediate points to transfer loads from one type of vehicles to another. Problems with these characteristics are briefly reviewed in the following.

2.1. Time-dependent VRPs

In the first studies about time-dependent VRPs, a customer-based graph was used, where nodes correspond to customers plus the depot and arcs stand for a particular path between two customers in the underlying road network (e.g., shortest path in distance). Since a fixed path is used to travel between two customers, it is only possible to account for different travel times at different departure times along that path. However, it is not possible to consider different paths between two customers at different departure times. To the best of our knowledge, the first work that addressed time-dependency (although without time windows at customers) on a customer-based graph is found in [**5**]. In this work, the time horizon is divided into periods with a different travel time matrix for each period. This is equivalent to defining a step function to model the travel time at different periods between any given pair of nodes. A similar approach is proposed in [**58**], although

for a problem with time windows. Since the travel time is constant within a period, but may abruptly change from one period to the next, the two previous models do not satisfy the First-In-First-Out (FIFO) property, where it is required that a vehicle traveling earlier on an arc must arrive at the destination node earlier than any other vehicle traveling later on the same arc. A different model is proposed in [**44**], where each node is assigned a speed at a given period of time, which can be interpreted as the average speed around the node. Then, the travel time on a given arc between two nodes is based on the average speed around these two nodes. Once again, since the travel time on a given arc is constant within a period, the FIFO property is not satisfied neither. A model that satisfies the FIFO property was finally proposed in [**45**]. Here, the authors use a step function to model speed (rather than travel time) at different time periods. That is, the speed is constant within a given time period, but may change from one period to the next. The travel time along an arc is then computed by taking into account the new speed when the time boundary between two periods is crossed. Thus, every vehicle that travels along the same arc within the same period has the same speed and the speed of every vehicle changes similarly when the boundary between two given periods is crossed. This way to model time-dependency has been largely adopted in the following years to solve TDVRPTWs using exact methods and metaheuristics [**25, 20, 62, 63**]. In a few cases, continuous functions with special characteristics to satisfy the FIFO property have also been used to model time-dependent travel times [**42, 4**]. For a detailed literature review on time-dependent VRPs using customer-based graphs (up to 2015), the reader is referred to [**35**].

Realistic objective functions are often based on travel times and the fastest path between two customers may well change depending on the departure time. To account for this, multi-graph representations have been proposed [**33, 7, 8**]. In these graphs, parallel arcs between any given pair of customer nodes stand for different least-cost paths in the underlying road network associated with different departure times from the origin customer node (or, more generally, for different non-dominated paths in multi-objective optimization). In [**7, 8**], the authors empirically demonstrate, using a branch-and-price algorithm and an adaptive large neighborhood search (ALNS), that a multigraph representation for a bi-objective VRP with time windows (VRPTW) that accounts for both travel time and travel cost leads to considerably better solutions when compared to a standard customer-based graph with a single arc (path) between two customers. However, the authors note the considerable computation times needed to compute the multigraph. Accordingly, the authors in [**55**], propose to work directly on the road network. In [**9**], a comparison between a branch-and-price algorithm applied to a road network and to a multigraph representation for a bi-objective VRPTW (travel time, travel cost) shows that both approaches are competitive, but with a slight advantage for multigraphs on the more realistic instances.

On the other hand, working directly on a road network is more natural and straightforward. Thus, recent works that addressed TDVRPTWs typically use road networks [**10, 38**].

2.2. VRPs with intermediate facilities

Due to the presence of transfer points in our problem, we provide an overview of the literature on VRPs with intermediate facilities, which are referred to as satellites, hubs, transshipment points or cross-docks. They all represent intermediate points where goods can be transferred while they move from their origin to their destination. In [**74**], the authors provide a survey about intermediate facilities in freight transportation, while a survey dedicated to cross-docking is found in [**83**]. In [**74**], the problems are divided into two classes, that is, two-echelon VRPs (2E-VRPs) and pickup and delivery problems with cross-docks (PDPCDs). With regard to 2E-VRPs, the intermediate facilities are called satellites and have some storage capacity. At the first-level or echelon, vehicles carry goods from a depot to satellites, while at the second-level goods are transported by other vehicles from satellites to customers. Typically, a strict hierarchy is observed, that is, direct deliveries from the depot to customers is forbidden. In this survey, no work requires synchronization between vehicles at satellites. In the case of the surveyed PDPCDs, however, cross-docks have no or little capacity and synchronization is required. Two works are worth mentioning, since there is no real intermediate facility, only transfer points (like in our work). In [**11**], transfers can take place at arbitrary locations and the vehicle that arrives first at the transfer point waits as long as necessary to transfer goods to the other vehicle, although a waiting penalty is incurred. In [**60**], transshipment points are predetermined and synchronization is achieved by setting a time window at these transfer points.

In [**19**], a 2E-VRP is proposed in the context of city logistics, where it is called a two-tier city logistics system. City Distribution Centers (CDCs), located at the outskirts of the city, form the first tier of the system where freight is sorted and consolidated. The second tier of the system is made of satellites located close to or within the city-center area. Different vehicle fleets are used to transport freight from CDCs to satellites and from satellites to customers. In particular, vehicles of the second tier must be adapted for utilization in dense city zones. Since it is assumed that satellites operate according to a cross-dock transshipment operational model, vehicle synchronization is required. That is, vehicles of the first and second tier must meet at satellites at a given time, with very short waiting time permitted. This work proposes only a modeling framework and no algorithmic solution is developed. Different exact and heuristic algorithms were later proposed in [**18, 64, 65**] to solve variants of the initial model (e.g., no synchronization, storage capacity at satellites). A recent work

in [**46**] addresses a two-commodity 2E-VRP with synchronization at satellites. Two types of vehicles are considered at the first level, one for each commodity, as opposed to the second level where only one type of vehicles is considered. Synchronization is only established between the two types of first-level vehicles, which have to meet at satellites to favor efficiency at the second level. The problem is solved with ALNS where, at each iteration, destroy and a repair operators are applied to the second-level tours, and a reconstruction procedure is then applied to the first-level tours in case of infeasibility, followed by an improvement procedure. The 2E-VRP reported in [**2**] is particularly interesting because is shares similarities with our problem. In this work, the first-level vehicles are called vans and the second-level vehicles are called bicycles. Similarly, there are two classes of customers depending on their location: customers located at the city center are called bike-customers, while customers outside of the center are called van-customers. Transfers take place at satellites, with no storage capacity. These satellites are located at the boundary of the city center (a van can cross the city center, but a penalty is incurred). In the proposed heuristic methodology based on GRASP and path-relinking, the second-level tours are constructed before the first level tours. In this way, information about the arrival times of bikes at satellites can be used to construct the first-level tours and account for synchronization. The authors in [**40**] address a similar problem called the two-echelon multi-trip VRP with satellite synchronization. The proposed methodology also constructs second-level tours before first-level tours to produce the initial solution. Then, an ALNS is applied with features aimed at improving synchronization, like a destroy operator that removes trips with the worst synchronization. In [**47**], the authors describe a crowdsourced system for urban parcel deliveries, where truck-carriers visit intermediate facilities called relay points and where parcels are transferred to pedestrians or cyclists that are close to the end customers. However, if no pedestrian or cyclist is available, the truck can perform the deliveries itself. In the proposed system, the delivery tasks, as well as candidate relay points, are broadcast on-line. Then, pedestrians and cyclists bid for these delivery tasks. Thus, a bid selection problem must be solved in addition to the routing problem. This is done in both cases with a tabu search. Another similar crowdsourced system is described In [**71**], where goods can be dropped at transfer points to be picked up later by other vehicles (i.e., transfer points have storage capacity).

3. Problem Definition

This paper addresses the time-dependent vehicle routing problem with time windows and transfer points on a road network or $TDVRPTWTP_{RN}$. As previously mentioned, two types of vehicles with different capacities are considered: black (large) and green (small) vehicles. The two sets of vehicles are denoted K^B and K^G, respectively. There are also three types of customers: black customers that can be served by black vehicles only; green

customers that can be served by green vehicles only; and neutral customers that can be served by both types of vehicles.

A road network in this context is a directed graph $G = (V,A)$, where V is the set of nodes of cardinality n and A the set of arcs or road segments. The set of nodes is then partitioned as follow:

- $D = \{d^b, d^g\}$ is the set of depots with d^b the depot for black vehicles and d^g the depot for green vehicles;
- C^B is the set of black customers of cardinality n_B ;
- C^G is the set of green customers of cardinality n_G ;
- C^E is the set of neutral customers of cardinality n_E;
- TP is the set of transfer points of cardinality n_{TP} ;
- RJ is the set of road junctions (i.e., any node that is not a depot, a customer or a transfer point).

The $TDVRPTWTP_{RN}$ can be characterized as follow:

— Each customer i has a demand d_i and a service (or dwell) time st_i;

— Each customer i has a time window $[\alpha_i,\beta_i]$ to constrain the service start time. If a vehicle arrives at customer i before α_i, then it must wait until α_i to start the service. On the other hand, a vehicle cannot arrive after β_i;

— A green vehicle can serve only one customer at a time, while a black vehicle has a capacity Q^b that allows it to serve many customers;

— The demand of all customers is assumed to be loaded into black vehicles at the start;

— Each black vehicle performs a single route that starts and ends at the black depot; each green vehicle performs a single route that starts and ends at the green depot;

— The black and green depots have a time window $[0,T]$, where T is the end of the time horizon; all vehicles must be back at their depot before or at time T;

— Black and green vehicles have different speeds. Thus, each arc $(i,j) \in A$ is associated with two time-dependent travel speed functions $v^b_{i,j}(t)$ and $v^g_{i,j}(t)$ for black and green vehicles, respectively.

— Transfer points are fixed locations without storage capacity, where a black vehicle can transfer loads to one or more green vehicles. We assume, without loss of generality, that the time Δ_{tp} to transfer a load is null, since a transfer time different to zero would just produce a delay Δ_{tp} in the departure times of the vehicles at transfer points. A black vehicle can visit the same transfer point multiple times along its

route; the same is true of green vehicles. Each visit to transfer point $tp \in TP$ in a route is represented by a copy which is unambiguously denoted tp_j^k, where k is a vehicle and j is the copy (or visit) index. That is, copy tp_j^k corresponds to the j^{th} visit of transfer point tp in the route of vehicle k;

— Each black customer is served directly and exactly once by a black vehicle; each green customer is served exactly once by a green vehicle after its demand has been transferred from a black vehicle at a transfer point; each neutral customer is served exactly once, either directly by a black vehicle or by a green vehicle after its demand has been transferred from a black vehicle at a transfer point;

— The objective is to determine routes of minimal total duration such that all customers are served and all constraints are satisfied.

Figure 1 shows a typical solution, with one black route starting from the black depot (square). This route is identified by arcs (1) to (10). At the first transfer point tp_1 (triangle), there is a connection with the green route with arcs identified with broken lines. This route starts at the green depot (square), gets a load from the black vehicle at tp_1, delivers the load to neutral customer nc_1 (gray node), gets another load from the same black vehicle at the second transfer point tp_2, delivers the load to green customer gc_1 and returns to the green depot. The arcs of the second green route are identified with dotted lines. This small route starts at the green depot, gets a load from the black vehicle at tp_2, delivers the load to green customer gc_2 and returns to the green depot. It should be noted that the black vehicle transfers two loads, one for each green vehicle, at transfer point tp_2.

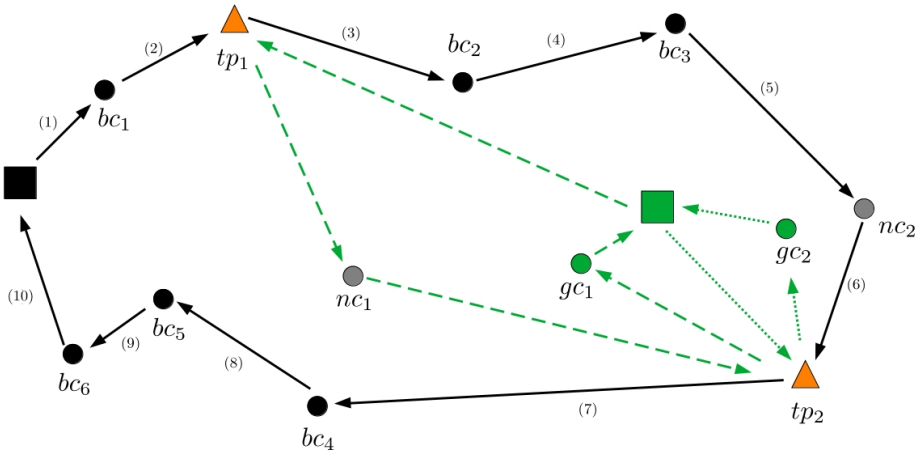

Figure 1. An example of a solution to the $TDVRPTWTP_{RN}$

4. SISR for the CVRP

The proposed methodology for solving our problem is the Slack Induction by String Removals (SISR) metaheuristic [**15**], which is state-of-the-art for the CVRP. This metaheuristic is based on the ruin-and-recreate principle where, at each iteration, a number of nodes are first removed from the routes of the current solution (ruin) and reinserted (recreate) to produce a new solution. A simulated annealing-based criterion is then applied to decide if the new solution should be accepted or not as the current solution. In the following, we precisely describe the *SISR* metaheuristic, as initially proposed for the *CVRP*.

The basic idea of SISR is to remove strings of consecutive customers from a solution, with at most one string removed from any given route. Algorithm 1 shows the pseudo-code of SISR for the CVRP. First, two parameter values are set: L^{max}, which is used to determine the maximum length of a string, and $\bar{c}$, which corresponds to the average number of customers to be removed from a solution. By appropriately setting these values, many strings of small length or only a few strings of large length can be removed. Given that simulated annealing principles guide the search through an exponential cooling schedule, the starting temperature τ_0, final temperature τ_f, $\tau_0 > \tau_f > 0$, and number of iterations f are defined, with the current temperature τ initially set to τ_0, see statements 2 and 3. Then, the cooling factor ρ is defined in statement 4 in such a way that f ruin-and-recreate iterations are performed.

An adjacency list $adj(i)$ is then created for each customer i in statement 5. This list contains all customers ordered from closest to farthest in distance from i, with i as its first element. This adjacency list is used to favor the removal of strings that are relatively close to each other, even if they come from different routes. Before proceeding with the main loop, an initial solution is created in step 6 in a straightforward way, by creating an individual route for each customer. This initial solution becomes the current solution s as well as the best solution known to date s_{best}.

The main loop corresponds to statements 8 to 18. At each iteration, a ruin operator and a recreate operator are applied to a copy $\overline{s}$ of current solution s, see statements 9 and 10. Note that the set A^- is used to store the removed customers. The resulting solution of the ruin-and-recreate process $\overline{s}$ is accepted as the new current solution s if it satisfies the simulated annealing-based criterion in statement 11 (see [**15**]). It also replaces s_{best} if it is the best solution found thus far. The current temperature τ is then updated before the next iteration starts. After f iterations of the main loop, the whole procedure stops and

returns the best solution found.

Algorithm 1 SISR for CVRP

Set L^{max} and $\bar{c}$
Set τ_0, τ_f and f
$\tau \leftarrow \tau_0$
$\rho \leftarrow \left(\frac{\tau_f}{\tau_0}\right)^{1/f}$
Generate adjacency list $adj(i)$ for each customer i
Generate initial solution s (with set of routes R_s)
$s_{best} \leftarrow s$
for f iterations **do**
    $\overline{s}, A^- \leftarrow Ruin(s)$
    $\overline{s} \leftarrow Recreate(\overline{s}, A^-)$
    if $Cost(\overline{s}) < (Cost(s) - \tau \ln(U(0,1)))$ **then**
        $s \leftarrow \overline{s}$;
    end if
    if $Cost(\overline{s}) < Cost(s_{best})$ **then**
        $s_{best} \leftarrow \overline{s}$
    end if
    $\tau \leftarrow \rho\tau$
end for
Return s_{best}

Algorithm 2 Ruin(s)

$l_s^{max} \leftarrow \min\{L^{max}, AvgNodesInRoutes(s)\}$
Calculate n_s^{max} with l_s^{max} and $\bar{c}$
$n_s \leftarrow \lfloor U(1,n_s^{\max}+1) \rfloor$
$R^- \leftarrow \emptyset$
$A^- \leftarrow \emptyset$
$\overline{s} \leftarrow s \quad R_{\overline{s}} \leftarrow R_s$
Select randomly a seed customer i^{seed} in $\overline{s}$
for $i \in adj(i^{seed})$ and $|R^-| < n_s$ **do**
    $r \leftarrow$ route of customer i
    if $i \notin A^-$ and $r \notin R^-$ **then**
        $l_r^{\max} \leftarrow \min\{l_s^{\max}, |r|\}$
        $l_r \leftarrow \lfloor U(1,l_r^{\max}+1) \rfloor$
        $RuinOp \leftarrow Random(String, Split\text{-}String)$
        $A^- \leftarrow A^- \cup RuinOp(\overline{s},r,l_r,i)$
        $R^- \leftarrow R^- \cup \{r\}$
        if r is empty **then**
            $R_{\overline{s}} \leftarrow R_{\overline{s}} \backslash \{r\}$
        end if
    end if
end for
Return $\overline{s}$, A^-

Algorithm 3 Recreate($\overline{s}$, A^-)

1: Sort(A^-) ▷ Recreate
2: **for** $i \in A^-$ **do**
3: $p_{best} \leftarrow NULL$; $CostInsert_{best} \leftarrow \infty$
4: **for** $r \in R_{\overline{s}}$ and r feasible with insertion of i **do**
5: **for** p_r in r **do**
6: **if** $U(0,1) < 1 - \gamma$ **then**
7: **if** $p_{best} = NULL$ or $CostInsert(i,p_r) < CostInsert_{best}$ **then**
8: $p_{best} \leftarrow p_r$
9: $CostInsert_{best} \leftarrow CostInsert(i,p_r)$
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **if** $p_{best} = NULL$ **then**
15: $R_{\overline{s}} \leftarrow R_{\overline{s}} \cup \{$new empty route $r\}$
16: $p_{best} \leftarrow$ first position in r
17: **end if**
18: Insert i in position p_{best}
19: **end for**
20: Return $\overline{s}$

4.1. Ruin

In the ruin procedure described in Algorithm 2, the maximum length of a string to be removed l_s^{max} is first set to the minimum of L^{max} and the average number of nodes in a route of the current solution $AvgRouteNodes(s)$, see statement 1. Then, in statement 2, the maximum number of removed strings n_s^{max} is calculated using l_s^{max} and $\overline{c}$, see the exact formula in [**15**]. The actual number of removed strings n_s is chosen from a continuous uniform distribution defined between 1 and $n_s^{max} + 1$, as indicated in statement 3. The set of ruined routes R^- and the set of removed customers A^- are then initialized with the empty set. After creating a copy $\overline{s}$ of the current solution s, a random seed customer i^{seed} is chosen and its adjacency list is processed (from closest to farthest customers) until all customers have been considered or the number of ruined routes is reached, see the main loop in statements 8-20. Note that the number of ruined routes is the same as the number of removed strings n_s, since each string is removed from a different route. If the current customer i in the adjacency list of i^{seed} has not been previously removed and if the route r that serves i has not been previously ruined (statement 10) then the ruin operator is applied to route r. In statements 11 and 12, the actual length of the removed string l_r is chosen from a continuous uniform distribution defined between 1 and $l_r^{\max} + 1$, where $l_r^{\max}$ is the minimum of $l_s^{\max}$ and cardinality of r (since the length of the removed string cannot exceed the number of nodes in r).

Then, a random choice between two ruin operators takes place in statement 13. These operators are:

— *String*: A random string of length l_r that contains the current customer i in the adjacency list of i^{seed} is removed from route r. This is illustrated in Figure 2(a) for a string of length four with the gray node i_3 as current customer i;

— *Split-String*: A random string of length $l_r + m$ that contains the current customer i in the adjacency list of i^{seed} is chosen in route r (where the procedure to select a value for m is precisely described in [**15**]). Then, a random substring of m consecutive customers within the chosen string is kept in the route, so that only l_r customers are removed. The substring of length m cuts the string of length $l_r + m$ in two parts, unless the substring is at the very beginning or very end of the string of length $l_r + m$. An example is provided in Figure 2(b) for a string of length five with the gray node i_3 as current customer i. In this example, $m = 2$, so that only three customers are removed from the route.

The removed customers are then added to A^- and the ruined route to R^- in statements 14 and 15. If route r becomes empty, then it is deleted from the set of routes in the solution, as indicated in statements 16 and 17. At the end, the ruined solution $\overline{s}$ and the set of removed customers are returned.

4.2. Recreate

A new complete solution is then produced with the recreate operator by reinserting the removed customers. This operator is described in Algorithm 3. In statement 1, the removed customers in A^- are first sorted using a sorting criterion chosen with a particular distribution probability among: random, decreasing demand, increasing distance from the depot, decreasing distance from the depot. Based on the chosen order, the customers in A^- are considered one by one and reinserted in the set of routes $R_{\overline{s}}$ of solution $\overline{s}$, see the main loop in statements 2-19. Each insertion place in each route that can accommodate the demand of customer i is considered and the best encountered insertion place p_{best} is identified. It should be noted, however, that the chosen insertion place is not necessarily the best one among all feasible insertion places, due to blinks that correspond to a small probability γ of skipping a position, see statement 6. In particular, if the best position is skipped then only the second best position can be chosen (as long as this position is not skipped too). Statements 14-17 cover the situation when no feasible insertion place is found for customer i. In this case, a new route is created for that customer. At the end, the recreated solution $\overline{s}$ is returned.

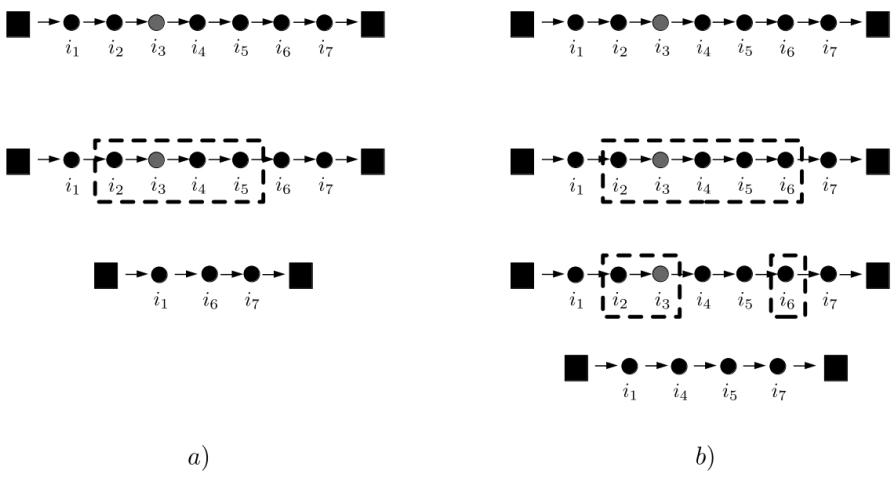

Figure 2. Examples of the two SISR ruin operators : (a) String (b) Split-String.

5. Time dependency

The SISR metaheuristic for the CVRP, as described in the previous section, needs to be considerably modified to address the much more complex $TDVRPTWTP_{RN}$. In particular, the time dimension must now be taken into account; furthermore, routes are not independent anymore since they interact through transfer points. In this section, we introduce the basics of our time-dependent travel time model and explain how time bounds can be derived at each node along the routes of black and green vehicles.

5.1. Time-dependent travel times

The IGP model proposed in [**45**] is used to model time dependency. In this model, the time horizon [0,T] is partitioned into a number l of time periods [0,t_1),[t_1,t_2),...., [t_{l-2},t_{l-1}), [t_{l-1},T], where t_1, t_2,..., t_{l-1} are time boundaries between two periods. For any given arc, a travel speed is associated with each period and a speed change occurs when a vehicle crosses a time boundary. The algorithmic procedure to compute the travel time along an arc for a given departure time based on this model is provided in [**45**]. Although speed is modeled as a step function of time, the corresponding travel time function is a piecewise linear function. Figure 3 shows an example of a travel speed function on a given arc (i, j) and the corresponding travel time function, assuming that the arc is of length 4.

5.2. Dominant shortest-path structure

The dominant shortest-path structure (DSPS), as described in [**38, 13**], is useful to quickly identify the fastest path between any given pair of nodes (either customers, depots

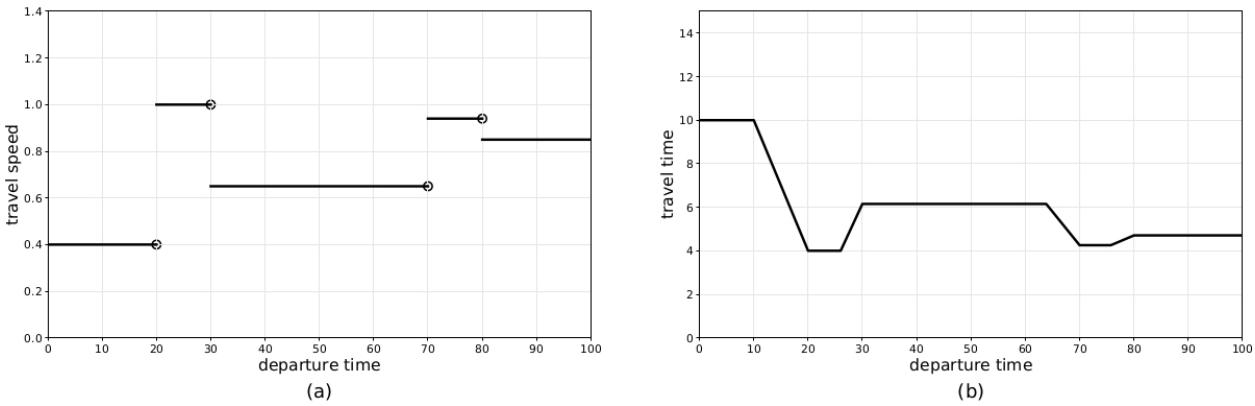

Figure 3. (a) Travel speed function of arc (i, j) (b) Corresponding travel time function assuming that arc (i, j) is of length 4.

or transfer points) in the road network for any given departure time. First, a number of good paths between two given nodes i and j are identified by applying a time-dependent Dijkstra's algorithm [**38, 13**] using different departure times from i, like time boundaries between two periods. The travel time function of each one of those paths is obtained by combining the travel time functions of all arcs along that path (which also produces a piecewise linear function). Figure 4 shows an example of a DSPS based on three different fastest paths between two nodes. In this figure, the arrival time is represented as a function of the departure time, so that the corresponding travel time is simply the difference between arrival and departure times. Since the IGP model satisfies the FIFO property, this piecewise linear function is non decreasing. By overlapping the three paths, it is possible to identify the fastest among the three paths for any given departure time. It is worth noting that the DSPS is exact only if the time-dependent Dijkstra's algorithm is applied with a sufficiently large number of departure times to cover all fastest paths between two nodes, which is rarely the case in practice. But better accuracy is obtained with more departure times. In the experimental section, we generated the dominant shortest-path structures with the implementation described in [**13**], which was kindly provided to us.

Two different travel time functions are associated with each arc, depending if a black or a green vehicle follows that arc, because they do not have the same speed. This leads to two different DSPSs between each pair of nodes made of either customers, depots or transfer points. Accordingly, the following notation will be used:

— $AT^b(i,j,dt)$ is the arrival time at j when a black vehicle departs from i at time dt and follows the fastest path to reach j, as determined by the DSPS of nodes i and j for a black vehicle;

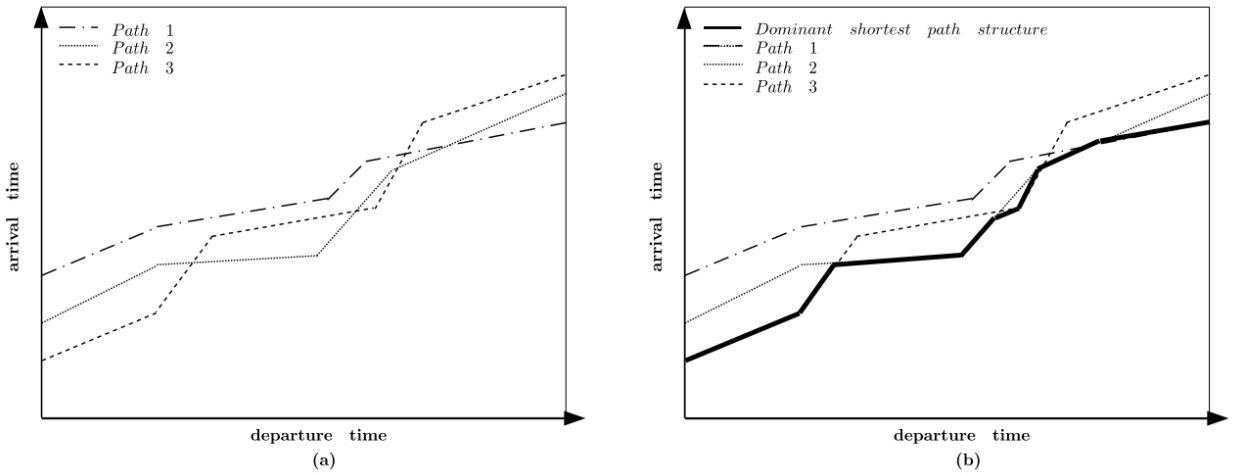

Figure 4. (a) Three different fastest paths between two nodes obtained at different time points using a time-dependent Dijkstra's algorithm (b) Corresponding dominant shortest path structure

— $DT^b(i,j,at)$ is the inverse of $AT^b(i,j,dt)$ and is the departure time at i that allows a black vehicle to arrive at j at time at;

— $AT^g(i,j,dt)$ is the arrival time at j when a green vehicle departs from i at time dt and follows the fastest path to reach j, as determined by the DSPS of the pair of nodes i and j for a green vehicle;

— $DT^g(i,j,at)$ is the inverse of $AT^g(i,j,dt)$ and is the departure time at i that allows a green vehicle to arrive at j at time at.

5.3. Synchronization at a transfer point

Let us consider $tp_{\cdot}^{k^b}$ a copy of transfer point $tp \in TP$ in the route of black vehicle $k^b \in K^B$, where the subscript $\cdot$ corresponds to a particular copy (visit) index of transfer point tp in the route of vehicle k^b. Let us also consider $tp_{\cdot}^{k_1^g}$, $tp_{\cdot}^{k_2^g}$, ..., $tp_{\cdot}^{k_h^g}$, h copies of transfer point tp in the routes of green vehicles k_1^g, k_2^g, ..., $k_h^g \in K^G$. We assume that black vehicle k^b needs to be synchronized with green vehicles k_1^g, k_2^g, ..., k_h^g at these copies of transfer point tp.

We first account for the arrival time of the last green vehicle:

$$at^{max} = \max_{l=1,\ldots,h} \{at_{tp_{\cdot}^{k_l^g}}\} \tag{5.1}$$

Then, the departure time of black vehicle k^b from $tp_{\cdot}^{k^b}$ is :

$$dt_{tp_{\cdot}^{k^b}} = \max\{at_{tp_{\cdot}^{k^b}}, at^{max}\} \tag{5.2}$$

That is, if all green vehicles arrive at the transfer point before black vehicle k^b, then the latter can depart immediately (given that the time to transfer loads from the black vehicle to green vehicles is null) with $dt_{tp_{\cdot}^{k^b}} = at_{tp_{\cdot}^{k^b}}$. Otherwise, vehicle k^b will depart at the arrival time at^{max} of the last green vehicle.

The departure of each green vehicle k_l^g from $tp_{\cdot}^{k_l^g}$, $l = 1, ..., h$, is:

$$dt_{tp_{\cdot}^{k_l^g}} = \max\{at_{tp_{\cdot}^{k^b}}, at_{tp_{\cdot}^{k_l^g}}\} \quad l = 1, ..., h \tag{5.3}$$

That is, if green vehicle k_l^g arrives at the transfer point before black vehicle k^b, it must wait for the arrival of vehicle k^b before it can depart from $tp_{\cdot}^{k_l^g}$. Otherwise, it can depart immediately with $dt_{tp_{\cdot}^{k_l^g}} = at_{tp_{\cdot}^{k_l^g}}$.

5.4. Time bounds

In the following, we define earliest and latest time bounds for the arrival at and departure from each node in the route of a black or green vehicle, where a node can be a customer, a transfer point or a depot. That is, a vehicle must arrive at (depart from) a node before its latest arrival (departure) time to guarantee that the rest of the route satisfies the time constraints. For simplifications purposes, the forward and backward propagation procedures described below focus on a single black or green route and do not account for possible complex interactions among routes (see subsection 5.5) .

5.4.1. Green route

Here, we explain how to propagate the earliest and latest arrival and departure times in a green route. For this purpose, let us consider the route of green vehicle $k^g \in K^G$ which is made of (1) a copy d_0^g of the green depot to start the route, (2) a sequence of copies of one or more transfer points $tp_{l\cdot}^{k^g}$, $l = 1, ..., p$, each followed by a green (or neutral) customer $i_l \in C^G \cup C^E$, $l = 1, ..., p$ and (3) a copy d_{p+1}^g of the green depot to end the route. That is, the route of green vehicle k^g is d_0^g, $tp_{1\cdot}^{k^g}$, i_1, $tp_{2\cdot}^{k^g}$, i_2, ..., $tp_{p\cdot}^{k^g}$, i_p, d_{p+1}^g.

Earliest arrival and departure times

First, the earliest departure time from d_0^g is set equal to 0. Then, we go forward by first computing the earliest arrival time *eat* at transfer point $tp_{1\cdot}^{k^g}$, using function AT^g:

$$eat_{tp_{1\cdot}^{k^g}} = AT^g(d_0^g, tp_{1\cdot}^{k^g}, 0) \tag{5.4}$$

Now, to determine the earliest departure time *edt*, we need to account for the corresponding black vehicle k^b from which green vehicle k^g should receive a load. Accordingly, if $eat_{tp_{1\cdot}^{k^g}} < eat_{tp_{1\cdot}^{k^b}}$, then $edt_{tp_{1\cdot}^{k^g}} = eat_{tp_{1\cdot}^{k^b}}$, since green vehicle k^g cannot depart from the

transfer point before the earliest arrival time of black vehicle k^b. Otherwise, $edt_{tp_{1.}^{k^g}} = eat_{tp_{1.}^{k^g}}$, given that the time to transfer a load is null.

Still going forward, we now consider customer i_i and compute its earliest departure time as :

$$eat_{i_1} = AT^g(tp_{1.}^{k^g}, i_1, edt_{tp_{1.}^{k^g}}) \tag{5.5}$$

Now, if $eat_{i_1} < \alpha_{i_1}$, then the earliest departure time $edt_{i_1} = \alpha_{i_1} + st_{i_1}$, otherwise $edt_{i_1} = eat_{i_1} + st_{i_1}$.

This forward procedure is repeated until the end depot d_{p+1}^g is reached and its earliest arrival time is determined.

Latest arrival and departure times

We start by setting the latest arrival time *lat* at the end depot d_{p+1}^g to be the end of time horizon T, that is $lat_{d_{p+1}^g} = T$. Then, we go backward by first computing the latest departure time *ldt* at customer i_p that allows vehicle k^g to arrive at d_{p+1}^g at time $lat_{d_{p+1}^g}$, using function DT^g:

$$ldt_{i_p} = DT^g(i_p, d_{p+1}^g, lat_{d_{p+1}^g}) \tag{5.6}$$

Now, if $ldt_{i_p} > \beta_{i_p} + st_{i_p}$, then ldt_{i_p} is reset to $\beta_{i_p} + st_{i_p}$, because vehicle k^g cannot depart from i_p later than $\beta_{i_p} + st_{i_p}$ without violating the time window constraint (i.e., the arrival time cannot exceed β_{i_p}). Then, the latest arrival time at customer i_p is simply computed as $lat_{i_p} = ldt_{i_p} - st_{i_p}$.

Still going backward, we now consider the transfer point $tp_{p.}^{k^g}$ and compute its latest departure time as :

$$ldt_{tp_{p.}^{k^g}} = DT^g(tp_{p.}^{k^g}, i_p, lat_{i_p}) \tag{5.7}$$

To determine the latest arrival time of the green vehicle k^g, we must account for the corresponding black vehicle k^b that transfers a load to vehicle k^g. That is, the green vehicle cannot arrive after the latest departure time of black vehicle k^b through the following formula:

$$lat_{tp_{p.}^{k^g}} = \min\{ldt_{tp_{p.}^{k^g}}, ldt_{tp_{p.}^{k^b}}\} \tag{5.8}$$

This backward procedure is applied until the the starting depot d_0^g is reached and its latest departure time is determined. It should be noted that a forward propagation starting

from the latest departure time at d_0^g, until d_{p+1}^g is reached, would produce the latest feasible schedule (i.e., latest possible arrival and departure times at each node along the green route).

5.4.2. Black route

Here, we explain how to propagate the earliest and latest arrival and departure times in a black route. For this purpose, let us consider the route of black vehicle $k^b \in K^B$ which is made of (1) a copy d_0^b of the black depot to start the route, (2) an arbitrary sequence of length p of black (or neutral) customers and copies of one or more transfer points and (3) a copy d_{p+1}^b of the black depot to end the route.

Earliest arrival and departure times

The procedure to compute the earliest arrival and departure times in a black route is similar to the one described for the green route, but two differences are noteworthy: (1) the function AT^b is used to compute the arrival time at a given node from the earliest departure time of the previous node and (2) the earliest departure time at a copy of a transfer point is computed differently, because green vehicles that visit the same transfer point to get a load from the black vehicle must be accounted for.

Considering case (2), let us suppose that black vehicle k^b visits copy $tp_{\cdot}^{k^b}$ of tranfer point $tp \in TP$ and that h green vehicles $k_1^g, k_2^g, ..., k_h^g$ visit copies $tp_{\cdot}^{k_1^g}, tp_{\cdot}^{k_2^g}, ..., tp_{\cdot}^{k_h^g}$ of the same transfer point and that synchronization is required (i.e., black vehicle k^b must transfer a load to each green vehicle). To compute the earliest departure time of vehicle k_b at the transfer point, we first consider the maximum earliest arrival time over all green vehicles, that is:

$$eat^{max} = \max_{l=1,...,h} \{eat_{tp_{\cdot}^{k_l^g}}\} \tag{5.9}$$

Then, the earliest departure time of vehicle k^b at $tp_{\cdot}^{k^b}$ can be computed from its earliest arrival time as follow:

$$edt_{tp_{\cdot}^{k^b}} = \max\{eat^{max}, eat_{tp_{\cdot}^{k^b}}\} \tag{5.10}$$

That is, black vehicle k^b cannot depart earlier than the earliest arrival time of the last green vehicle, otherwise one or more green vehicles will not get their load.

Latest arrival and departure times

The procedure to compute the latest arrival and departure times of a black vehicle is similar to the one described for a green vehicle, although two differences are noteworthy: (1) the function DT^b is used to compute the departure time from a given node to reach the next node at its latest arrival time and (2) the latest arrival time at a copy of a transfer

point is computed differently, because green vehicles that visit the same transfer point to get a load from the black vehicle must be accounted for.

Considering case (2), let us suppose that black vehicle k^b visits copy $tp_{\cdot}^{k^b}$ of tranfer point $tp \in TP$ and that h green vehicles $k_1^g, k_2^g, ..., k_h^g$ visit copies $tp_{\cdot}^{k_1^g}, tp_{\cdot}^{k_2^g}, ..., tp_{\cdot}^{k_h^g}$ of the same transfer point and that synchronization is required (i.e., black vehicle k^b must transfer a load to each green vehicle). To compute the latest arrival time of vehicle k^b at the transfer point, we first consider the minimum latest departure time over all green vehicles, that is:

$$ldt^{min} = \min_{l=1,...,h} \{ldt_{tp_{\cdot}^{k_l^g}}\} \tag{5.11}$$

Then, the latest arrival time of vehicle k^b at $tp_{\cdot}^{k^b}$ can be computed from its latest departure time as follow:

$$lat_{tp_{\cdot}^{k^b}} = \min\{ldt^{min}, ldt_{tp_{\cdot}^{k^b}}\} \tag{5.12}$$

That is, black vehicle k^b cannot arrive at the transfer point later than the minimum latest departure time over all green vehicles that require synchronization, otherwise one or more green vehicles will not get their load.

5.5. Interaction among routes

In the previous section, our description of forward and backward propagation procedures to derive time bounds has focused on a single black or green route. However, complex interactions may occur when multiple black and green routes are involved.

Figure 5 shows an example where customer i is inserted between nodes *prev* and *next* in black route k_2^b (as it occurs during the recreate procedure of SISR). Forward propagation is illustrated in Figure 5(a). First, the ealiest arrival and departure times of the newly inserted customer i are calculated from the earliest departure time at transfer point *prev*. Then, forward propagation is triggered along the black route. However, a green route that connects the three black routes is encountered at the transfer point just after customer *next*. Thus, another forward propagation is triggered at this transfer point along the green route which, in turn, leads to a transfer point that connects the green route to black route k_3^b, thus triggering another forward propagation along that black route. It should be noted that this illustration is a worst case, because forward propagation along a route terminates as soon as the earliest departure time from a node does not change.

Backward propagation is illustrated in Figure 5(b). First, latest arrival and departure times at the newly inserted customer i are calculated from the latest arrival time at customer

next. Then, backward propagation is triggered along black route k_2^b. Since node *prev* is a transfer point, it triggers another backward propagation along the corresponding green route. Still going backward along the black route, another transfer point is met that triggers backward propagation along another green route. Finally, both green routes connect to black route k_1^b at the same transfer point, thus triggering a backward propagation along that black route also. Once again, this illustration is a worst case, because backward propagation along a route stops as soon as the latest arrival time at a node does not change.

To summarize, we had to implement forward and backward propagation procedures that account for the whole solution.

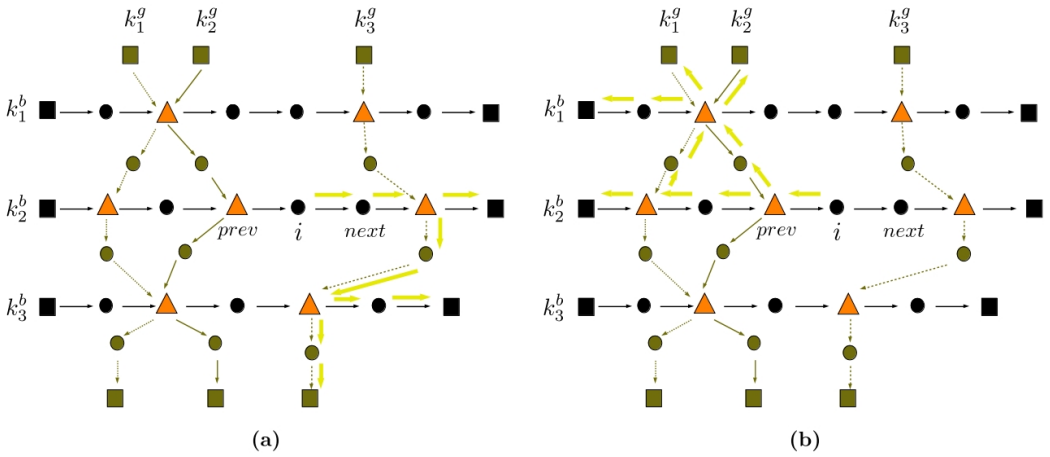

Figure 5. Example of forward and backward propagations when customer i is inserted in a route: (a) forward propagation (b) backward propagation.

6. SISR for the $TDVRPTWTP_{RN}$

The previous section has focused on time issues that arise when the $TDVRPTWTP_{RN}$ is considered. In the following, we revisit the SISR metaheuristic described in Section 4 given that our problem is much more complex than the CVRP.

Although the general algorithmic framework of SISR remains quite the same, there are important differences that can be observed in Algorithm 4. In the initialization phase, the dominant shortest path structures obtained from the underlying road network must first be created to account for the time-dependent travel times, see statements 5 and 6. More precisely, the DSPS of every pair of nodes that involves a black customer, a neutral customer, a transfer point or the black depot is generated, assuming that a black vehicle is used to travel between the two nodes. Then, the DSPS of every pair of nodes that involves a green customer, a neutral customer, a transfer point or the green depot is generated,

assuming that a green vehicle is used to travel between the two nodes.

Algorithm 4 SISR for $TDVRPTWTP_{RN}$

1: Set $L^{\max}$ and $\overline{c}$
2: Set τ_0, τ_f and f
3: $\tau \leftarrow \tau_0$
4: $\rho \leftarrow \left(\frac{\tau_f}{\tau_0}\right)^{1/f}$
5: Generate DSPS^b for every admissible pair of nodes, assuming that a black vehicle is used
6: Generate DSPS^g for every admissible pair of nodes, assuming that a green vehicle is used
7: Generate $adj^b(i)$ of each black customer $i \in C^B$
8: Generate $adj^g(i)$ of each green customer $i \in C^G$
9: Generate $adj^b(i)$ and $adj^g(i)$ of each neutral customer $i \in C^E$
10: Generate $adj^{tp}(i)$ of each green and neutral customer $i \in C^G \cup C^E$
11: $s \leftarrow Initial\_Solution(k)$
12: **for** f iterations **do**
13: Calculate $\overline{c}_s^b$ and $\overline{c}_s^g$
14: $\overline{s}, A^b \leftarrow RuinBlack(s)$
15: $\overline{s}, A^g \leftarrow RuinGreen(\overline{s})$
16: $A^- \leftarrow A^b \cup A^g$
17: $\overline{s} \leftarrow Recreate(\overline{s}, A^-)$
18: **if** $Cost(\overline{s}) < (Cost(s) - \tau \ln(U(0,1)))$ **then**
19: $s \leftarrow \overline{s}$;
20: **end if**
21: **if** $Cost(\overline{s}) < Cost(s_{best})$ **then**
22: $s_{best} \leftarrow \overline{s}$
23: **end if**
24: $\tau \leftarrow \rho\tau$
25: **end for**
26: Return s_{best}

In statements 7-10, the adjacency list of each black, green and neutral customer i is generated. In the adjacency list $adj^b(i)$ of a black customer, only black and neutral customers are considered. In the adjacency list adj^g of a green customer, only green and neutral customers are considered. In the case of a neutral customer, since they can be visited by both types of vehicles, two adjacency lists $adj^b(i)$ and $adj^g(i)$ are generated: one that contains only black and neutral customers (in case the neutral customer is in a black route) and the other that contains only green and neutral customers (in case the neutral customer is in a green route). Also, for each green and neutral customer, an adjacency list $adj^{tp}(i)$ made of all feasible transfer points, from closest to farthest from i, is created. A transfer point is feasible for a green or neutral customer, if it is possible for a green vehicle to receive the corresponding load from a black vehicle and serve the customer, while satisfying all constraints.

6.1. Initial solution

Another difference with the original SISR implementation is how the initial solution is generated in statement 11. A solution is constructed with a greedy insertion heuristic where, at each iteration, a customer is randomly selected and then inserted at its best place in the current partial solution. This is repeated until all customers are served. This insertion procedure is the same as the one used in the recreate operator, except that all feasible insertion places are considered (i.e., there is no blink so that no insertion place is skipped). Since this is a randomized heuristic, different runs typically produce different solutions. Accordingly, in the computational results, the greedy insertion heuristic was run 100 times on each instance and the best solution obtained was chosen as the intial solution (preliminary experiments have shown that no significant improvement is observed beyond 100 runs).

At each iteration of the main loop in statements 12-25, *RuinBlack* and *RuinGreen* are applied in sequence to ruin black and green routes, respectively. In the original algorithm, parameter $\overline{c}$ determines the average number of customers that are removed from the current solution by the ruin operator. Since we have two types of routes, we define $\overline{c}_s^b$ and $\overline{c}_s^g$ in statement 13 to control the average number of customers that are removed from the black and green routes, respectively, of solution s with $\overline{c}_s^b + \overline{c}_s^g = \overline{c}$. The values of $\overline{c}_s^b$ and $\overline{c}_s^g$ are dynamically set at each iteration depending on the number of customers in black routes n_s^{BR} and number of customers in green routes n_s^{GR} in the current solution, using the formula :

$$\overline{c}_s^b = \left\lceil \left(\frac{n_s^{BR}}{n_s^{BR} + n_s^{GR}} \right) \cdot \overline{c} \right\rceil \tag{6.1}$$

with $\overline{c}_s^g = \overline{c} - \overline{c}_s^b$. This dynamic setting is required because the number of customers in the routes of black and green vehicles cannot be known a priori, due to the presence of neutral customers that can be served by both types of vehicles. After collecting in set A^- the two sets of removed customers A^b and A^g from the black and green routes, respectively (see statements 14 and 15), the *Recreate* operator is called in statement 17 with the ruined solution and the removed customers. When the new recreated solution $\overline{s}$ is returned, this new solution is compared with the current solution s in the same way as in the original algorithm, see statements 18-23.

In the following, we will now focus on the *Ruin* and *Recreate* procedures.

6.2. Ruin

Given that black and green routes do not have the same structure, a different *Ruin* operator has been designed for each type of route. It should first be noted that removing

customers has no impact on solution feasibility (i.e., a feasible solution will remain feasible). Thus, the earliest and latest arrival and departure times at each node can be recalculated only once after the two ruin operators for black and green routes have been applied. The same applies to the solution value, which is of no interest while customers are removed from the solution.

6.2.1. Ruining black routes

The pseudo-code of *RuinBlack* is shown in Algorithm 5. The *RuinBlack* procedure is very similar to the *Ruin* procedure, except that the focus is only on black routes. Thus, the maximum length of a string l_s^{max} is taken as the minimum between L^{max} and the average number of customers in black routes *AvgCustInBlackRoutes(s)*, see statement 1, while the maximum number of strings n_s is calculated using l_s^{max} and $\overline{c}_s^b$ in statement 2. The seed customer must also be chosen in a black route, see statement 7. As in the original implementation, the ruin operator is randomly chosen between $String^b$ and $Split-String^b$, which are adaptations of *String* and *Split-String*, see statement 14. In these two new operators, the transfer points in a black route are always preserved and only customers are removed. This is because a transfer point in a black route connects to one or more green routes, so that the green or neutral customer that follows the removed transfer point in each green route would have to be removed too.

Figure 6 illustrates operator $String^b$, assuming that four customers must be removed. When two or more consecutive copies of the same transfer point are visited by the black vehicle after the string removal, then these copies are merged into a single copy. This is illustrated in the figure where two consecutive copies of the same transfer point tp_3 are merged together. Furthermore, the green routes that were connected to the original copies are collected and are all connected to the new single copy. The operator $Split-String^b$, works similarly, except that m customers in the string are preserved.

6.2.2. Ruining green routes

Ruining a green route also shares similarity with the original *Ruin* operator, except that the focus is on green routes. One particularity of *RuinGreen* is that the seed customer is the closest from the seed customer chosen in *RuinBlack*, over all green and neutral customers in green routes. The idea is to ruin green routes that are close to the previously ruined black routes. Another particularity is that each time a green or neutral customer is removed from a green route, the copy of the transfer point where the load is transferred from a black route to the green route must also be removed. That is, a pair $(tp_\cdot^{kg}, i)$ is removed, where $tp_\cdot^{kg}$ is the copy of transfer point $tp \in TP$ in the green route and i is the immediate successor customer, that is, the one who receives the load. The pseudo-code of *RuinGreen* is shown

Algorithm 5 RuinBlack(s)

$l_s^{max} \leftarrow \min\{L^{max}, AvgCustInBlackRoutes(s)\}$
Calculate n_s^{max} with l_s^{max} and $\overline{c}_s^b$
$n_s \leftarrow \lfloor U(1,n_s^{max}+1) \rfloor$
$R^b \leftarrow \emptyset$
$A^b \leftarrow \emptyset$
$\overline{s} \leftarrow s \quad R_{\overline{s}} \leftarrow R_s$
Select randomly a seed customer i_{seed}^b over all black and neutral customers in black routes
for $i \in adj^b(i_{seed}^b)$ and $|R^b| < n_s$ **do**
    if $(i \in C^B)$ or $(i \in C^E$ and i is in a black route) **then**
        $r \leftarrow$ route of customer i
        if $i \notin A^b$ and $r \notin R^b$ **then**
            $l_r^{max} \leftarrow \min\{l_s^{max}, |r|\}$
            $l_r \leftarrow \lfloor U(1,l_r^{max}+1) \rfloor$
            $RuinOp \leftarrow Random(String^b, Split-String^b)$
            $A^b \leftarrow A^b \cup RuinOp(\overline{s},r,l_r,i)$
            $R^b \leftarrow R^b \cup \{r\}$
            if r is empty **then**
                $R_{\overline{s}} \leftarrow R_{\overline{s}} \backslash \{r\}$
            end if
        end if
    end if
end for
Return $\overline{s}$, A^b

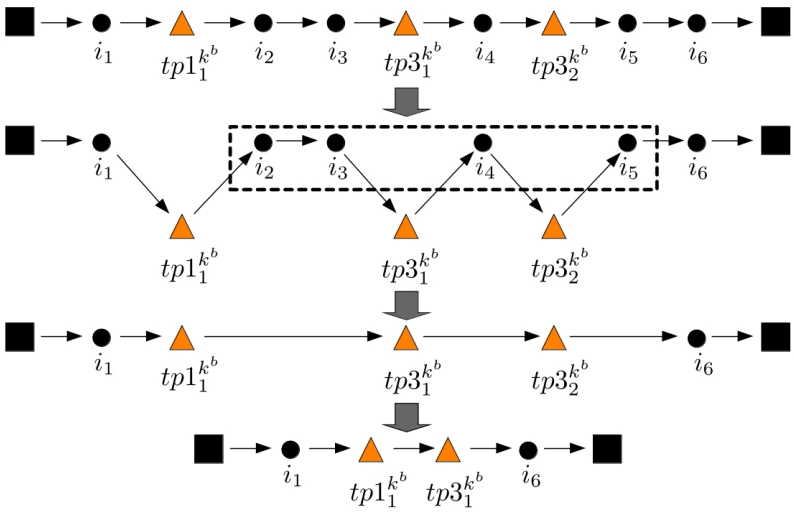

Figure 6. Example of $String^b$ where four customers are removed. First, the transfer points are set apart, then a string of cardinality four is chosen and the corresponding customers are removed.

in Algorithm 6.

Algorithm 6 RuinGreen(s)

$l_s^{max} \leftarrow \min\{L^{max}, AvgCustInGreenRoutes(s)\}$
Calculate n_s^{max} with l_s^{max} and $\overline{c}_s^g$
$n_s \leftarrow \lfloor U(1,n_s^{\max}+1) \rfloor$
$R^g \leftarrow \emptyset$
$A^g \leftarrow \emptyset$
Select i_{seed}^g the closest customer from i_{seed}^b over all green and neutral customers in green routes
for $i \in adj^g(i_{seed}^g)$ and $|R^g| < n_s$ **do**
    if $(i \in C^G)$ or $(i \in C^E$ and i is in a green route) **then**
        $r \leftarrow$ route of customer i
        if $i \notin A^g$ and $r \notin R^g$ **then**
            $l_r^{max} \leftarrow \min\{l_s^{max}, |r|\}$
            $l_r \leftarrow \lfloor U(1,l_r^{max}+1) \rfloor$
            $RuinOp \leftarrow Random(String^g,\ Split-String^g)$
            $A^g \leftarrow A^g \cup RuinOp(\overline{s},r,l_r,i)$
            $R^g \leftarrow R^g \cup \{r\}$
            if r is empty **then**
                $R_{\overline{s}} \leftarrow R_{\overline{s}} \backslash \{r\}$
            end if
        end if
    end if
end for
Return $\overline{s}$, A^g

Figure 7 shows an example of $String^g$ where two customers are removed. It should be noted that removing a copy of a transfer point in a green route may have an impact on the corresponding black route if the green route is the only one that connects to the black route there. In this case, the corresponding copy of the transfer point in the black route must also be removed. Also, if this removal leads to the visit of two or more consecutive copies of another transfer point in the black route, then these copies are merged into a simple copy, as previously explained in subsection 6.2.1.

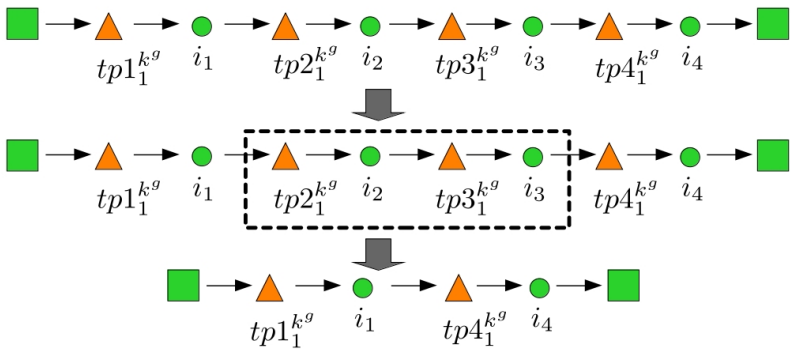

Figure 7. Example of $String^g$ where two customers are removed.

6.3. Recreate

Given that we have two different types of routes and three different types of customers, reinserting the removed customers is more complex than in the original implementation of SISR for the CVRP. As shown in Algorithm 7, the removed customers in set A^- are first sorted randomly or by decreasing demand, with equal probability (the two other sorting criteria proposed in [**15**], increasing and decreasing distance from the depot are not considered due to the ambiguity for neutral customers who can appear in black and green routes). Then, the customers are considered one by one in the sorted list. If the current customer is black then the method *InsertionBlack* for insertion in a black route is called. If the current customer is green then the method *InsertionGreen* for insertion in a green route is called. Finally, if the current customer is neutral then both methods are called and the best of the two proposed insertions is chosen. The procedure for inserting a customer in black routes is similar to the one presented in Algorithm 3 for the CVRP, except that the focus is on black routes only. However, the procedure for inserting a customer in a green route is more complex, as it is explained below.

It should also be noted that the *Recreate* method assumes that the vehicles visit nodes as soon as possible, that is, they follow a schedule based on the earliest arrival and departure times at each node. This earliest schedule induces slack time (waiting time) in the routes that can be exploited to feasibly insert new customers. But since we aim at minimizing the total duration of routes, the latest schedule is then used to get exact solution costs.

Algorithm 7 Recreate($\overline{s}$, A^-)

```
1: Sort(A⁻)
2: for i ∈ A⁻ do
3:     if i is a black customer then
4:         s̄ ← InsertionBlack(i, s̄)
5:     end if
6:     if i is a green customer then
7:         s̄ ← InsertionGreen(i, s̄)
8:     end if
9:     if i is a neutral customer then
10:        s̄ ← Best of InsertionBlack(i, s̄) and InsertionGreen(i, s̄)
11:    end if
12: end for
13: Return s̄
```

6.3.1. Insertion in black routes

As in the original implementation for the CVRP, *InsertionBlack* considers all possible insertions of the current customer i between two consecutive nodes in black routes with enough residual capacity to accommodate the demand of i, except for blinks (i.e., a position

may be skipped with a small probability γ).

It is worth noting that the feasibility and (approximate) evaluation of inserting customer i between two consecutive nodes j and l in a black route are done in constant time. First, the arrival and departure times at i are calculated from the departure time at j to check if the time window at i is satisfied. If i is feasible, then the new arrival time at l is calculated from the departure time at i. If the new arrival time at l does not exceed its latest arrival time lat_l, then the insertion is feasible. Assuming feasibility, the additional cost induced by this insertion is then evaluated. To maintain a constant time evaluation, an approximation is used. That is, the approximate or local additional cost corresponds to the arrival time delay at node l due to the insertion of customer i (even if l is the end depot). This delay corresponds to the difference between the arrival time at l after the insertion of i minus the arrival time at l before the insertion of i. To get an exact evaluation of the additional cost, the delay at l would need to be propagated along the route, until either it vanishes (due to waiting times) or the end depot is reached. It may also lead to forward propagation along other connecting routes (see Section 5.5). To alleviate the impact of using only approximate additional costs, the n_{pos} best insertion places of customer i, based on the approximation, are kept. Each one of these n_{pos} alternative insertion places are then evaluated exactly through propagation. The best insertion place, based on the exact evaluation of the additional cost, is finally chosen. It is worth noting that, after the insertion of customer i, the latest arrival and departure times need to be recomputed through backward propagation from i to the starting depot. This may also lead to backpropagation along other connecting routes (see Section 5.5). If no feasible insertion place is found for customer i then a new route is created for this customer.

6.3.2. Insertion in green routes

When a customer is inserted in a green route, a copy of a transfer point needs to be coupled with it. Accordingly, the search for possible insertion places is divided into four different phases, as it is explained below. It should be noted that the insertion procedure of each phase accounts for blinks. Furthermore, as for black routes, the feasibility and (approximate) evaluation of all possible insertion places of the current customer in green routes are done in constant time.

Phase I. Existing copy of a transfer point in a black route; existing green route.

If i is a green or neutral customer, we consider every black vehicle k^b with enough residual capacity to accommodate the demand of i. Then, we consider the route of every green vehicle k^g and every copy $tp_{\cdot}^{k^b}$ of a transfer point in the route of black vehicle k^b

that satisfy the two following conditions: (1) the route of black vehicle k^b does not already connect with the route of green vehicle k^g through $tp_{\cdot}^{k^b}$ and (2) the transfer point is in set $adj^{tp}(i)$. Then, we create a corresponding copy $tp_{\cdot}^{k^g}$ for the green vehicle k^g and we try to insert the pair $(tp_{\cdot}^{k^g}, i)$ after every customer in the green route. An example is provided in Figure 8 where the two corresponding copies of a transfer point in the black and green routes are represented as a single node tp. A further refinement allows a reduction in the number of insertion places to be considered. Let us denote $\widetilde{tp}_L$ the last transfer point before tp that connects vehicles k^b and k^g (if any) and $\widetilde{tp}_R$ the first transfer point after tp that connects vehicles k^b and k^g (if any), see Figure 8. Then, there is no need to consider insertion places in the green route after customers that are visited before $\widetilde{tp}_L$. In such a case, $\widetilde{tp}_L$ would be visited before tp in the black route while $\widetilde{tp}_L$ would be visited after tp in the green route, thus no synchronization of the black and green vehicles is possible. Similarly, insertion places in the green route after customers that are visited after $\widetilde{tp}_R$ are discarded, since in such case $\widetilde{tp}_R$ would be visited after tp in the black route, but tp would be visited before $\widetilde{tp}_R$ in the green route.

It is important to note that the insertion of customer i impacts both the black and green routes. With regard to feasibility of the black route, the departure time of vehicle k^b at tp may be delayed since a load must now be transferred to green vehicle k^g (vehicle k^b may have to wait for vehicle k^g). However, it the new departure time does not exceed the latest departure time of k^b at tp, then the insertion is feasible in the black route. With regard to feasibility of the green route, we start with customer j^g after which the transfer point and customer i are inserted. That is, time is forward propagated from the latest arrival time at j^g to the transfer point, customer i and the following transfer point (or end depot), denoted l^g in the figure. If the time constraints are satisfied at the transfer point and at customer i, and if the new arrival time of the green vehicle at l^g does not exceed its latest arrival time then the insertion is feasible in the green route. The approximate or local cost of this insertion corresponds to the arrival time delay of black vehicle k^b at l^b plus the arrival time delay of green vehicle k^b at l^g.

Phase II. Existing copy of a transfer point in a black route; new green route.

This is similar to Phase I, except that a new green route for the current green or neutral customer i is created. This is illustrated in Figure 9. Feasibility can be checked similarly to Phase I. The approximate or local cost corresponds here to the duration of the new green route plus the arrival time delay of black vehicle k^b at l^b, given the new green route.

Phase III. New copy of a transfer point in a black route; existing green route.

Due to the additional computational complexity of this phase, we only consider the most interesting insertion places through a subset of $adj^{tp}(i)$ that contains the n_{ftp} nearest feasible transfer points of current customer i. For each such transfer point, we insert a copy of the transfer point at each possible insertion place in the route of each black vehicle with enough residual capacity to accommodate the demand of i. Then, for each such insertion place of this copy in a black route, we insert a corresponding copy of the transfer point followed by customer i after each green customer in each green route. This is illustrated in Figure 10 where the copies of the transfer point in the black and green routes are represented as a single node denoted *tp*. The transfer points denoted as $\widetilde{tp}_L$ and $\widetilde{tp}_R$ have the same meaning than in Phase I and are also used to reduce the number of insertion places that must be considered. In addition, there is no need to consider the insertion of the new copy before or after a copy of the same transfer point in the black route. These two consecutive copies of the same transfer point could then be merged, which would lead to a case already considered in Phase I (c.f., existing copy of a transfer point in a black route). Feasibility is checked as in Phase I and the approximate cost is obtained by summing the arrival time delay of the black vehicle at customer l^b and arrival time delay of the green vehicle at l^g.

Phase IV. New copy of a transfer point in a black route; new green route.

This is similar to Phase III, except that a new green route for the current green or neutral customer i is created, as illustrated in Figure 11. The approximate cost is obtained here by summing the duration of the new green route, plus the arrival time delay of the black vehicle at l^b, given the new green route.

The n_{pos} best insertion places according to the approximate cost, as identified during Phases I to IV, are then evaluated exactly through propagation. For evaluation purposes, the latest schedule is used to reduce as much as possible the waiting time. The best insertion place, based on the exact evaluation of the additional cost, is finally chosen. After the insertion of customer i at the chosen place, the latest arrival and departure times must be recomputed through backward propagation from customer i.

If no feasible insertion place is found for customer i, then two new routes are created to serve this customer, one for a black vehicle and one for a green vehicle. This is shown in Figure 12, where the nearest feasible transfer point from i is used to connect the two routes.

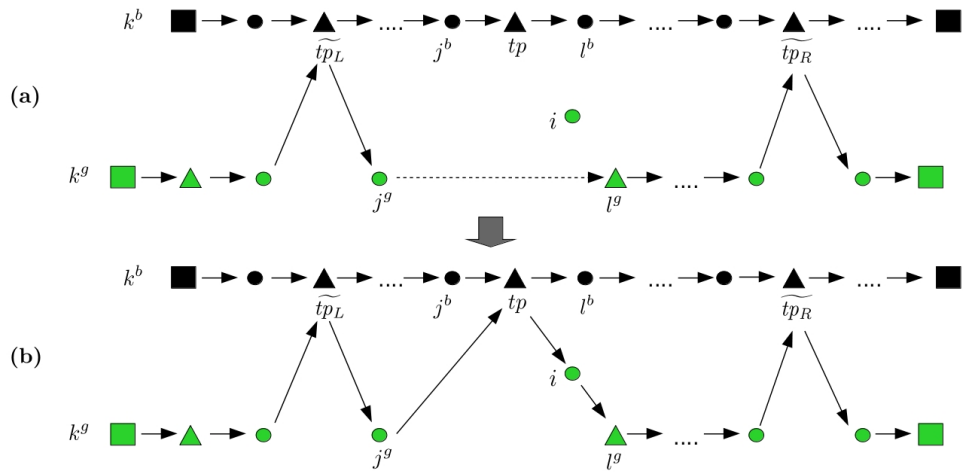

Figure 8. Phase I : a transfer point tp (already present in the black route) followed by customer i are inserted after customer j^g in the green route; (a) and (b) show the black and green routes before and after the insertion, respectively.

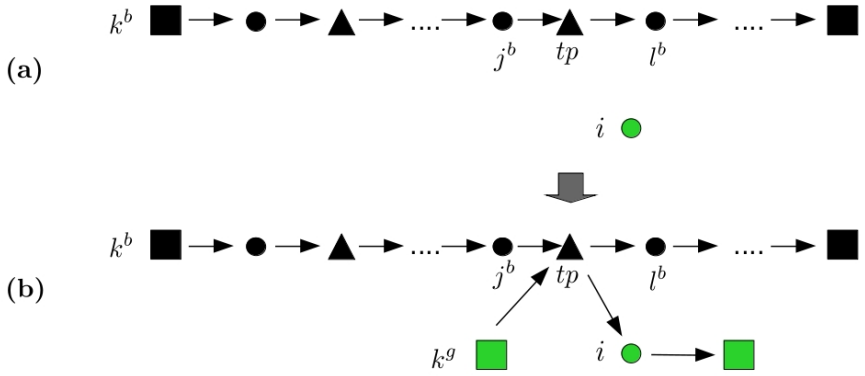

Figure 9. Phase II: a transfer point tp (already present in the black route) followed by customer i are inserted in a new green route; (a) and (b) show the black and green routes before and after the insertion, respectively.

7. Computational experiments

In the following sections, we first describe how the test instances were designed. Then, we explain the parameter tuning process of our SISR. This is followed by the results obtained on our test instances. Finally, we analyze the synchronization efficiency at transfer points and study the impact of customer distributions, scenarios, time windows and number of transfer points on the solutions obtained. We note that the results reported in this section were obtained with a processor Intel Gold 6148 Skylake 2.4 GHz, with 6GB of RAM.

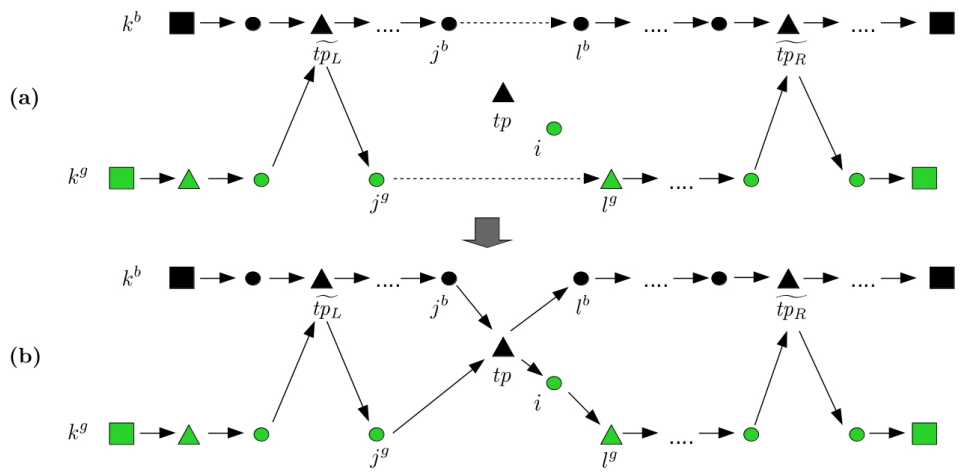

Figure 10. Phase III: a new transfer point tp is inserted in the black route; this transfer point followed by customer i are inserted after customer j^g in the green route; (a) and (b) show the black and green routes before and after the insertion, respectively.

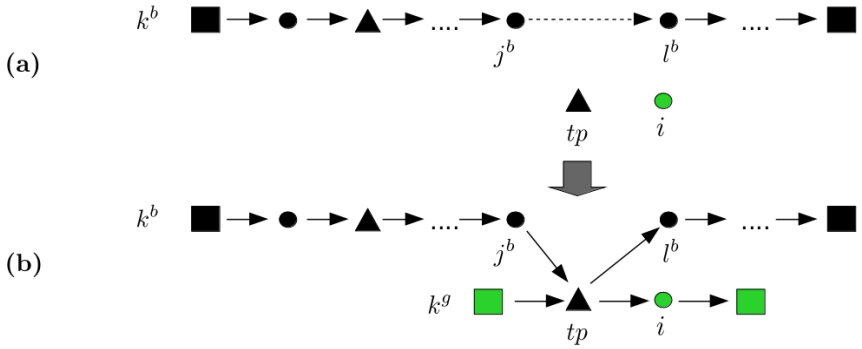

Figure 11. Phase IV: a new transfer point tp is inserted in the black route; this transfer point followed by customer i are inserted in a new green route; (a) and (b) show the black and green routes before and after the insertion, respectively.

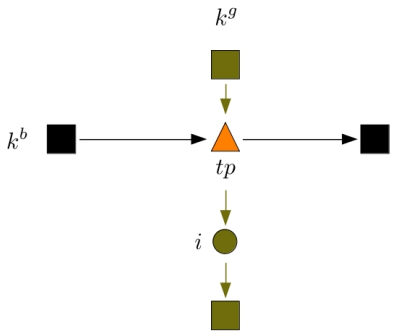

Figure 12. A new black and a new green route are created for a single green or neutral customer

7.1. Test instances

Our benchmark was produced by extending the well-known *NEWLET* instances for the $TDVRPTW_{RN}$ [**6**]. These Euclidean instances are defined on road networks generated by a procedure reported in [**55**]. Node coordinates in these networks are randomly generated in the interval $[-10\sqrt{n},10\sqrt{n}]$, where n is the number of nodes. The arc set E is obtained by considering all possible arcs and by adding an arc to set E when two conditions are satisfied: (1) the new arc does not cross any arc already included in E and (2) if the new arc has in common one endpoint with an arc already included in E, then the angle between these two arcs must be greater than or equal to 60 degrees.

In the $NEWLET$ instances, there are different networks of different sizes. For each road network of each size, three different ways to set the fixed or nominal travel time on every arc, depending on the correlation level with its length, are proposed through the formula $t_{ij} = \nu \cdot d_{ij} + \mu \cdot \gamma_{ij} \cdot \bar{d}$, where t_{ij} and d_{ij} are the travel time and length of arc (i,j), respectively. In the formula, $\bar{d}$ denotes the maximum arc length in the road network, ν and μ are correlation parameters selected in interval [0,1] and γ_{ij} is a randomly generated number in interval [0,1] for each arc (i,j). Thus, for each road network of each size, three different instances are obtained, which are denoted NC (i.e., no correlation, with $\nu = 0$ and $\mu = 1$), WC (i.e., weak correlation, with $\nu = 0.5$ and $\mu = 0.5$) and SC (i.e., strong correlation, with $\nu = 0.9$ and $\mu = 0.1$). The nominal travel times are then used to derive nominal travel speeds. Then, each one of these instances is duplicated by considering instances with narrow time windows (NTW) and wide time windows (WTW), where the length of narrow time windows is randomly selected from $\{3, 4\}$, and the length of wide time windows is randomly selected from $\{10, ..., 15\}$. To account for time-dependency, the time horizon [0,100] is partitioned into five periods $\tau_1 = [0, 20)$, $\tau_2 = [20, 30)$, $\tau_3 = [30, 70)$, $\tau_4 = [70, 80)$ and $\tau_5 = [80, 100]$. Based on this partition, one of three different time-dependent speed profiles is randomly associated with each arc, where a profile corresponds to a set of five speed multipliers, one for each time period. Finally, a service time is randomly selected from $\{1, 2\}$ for each customer.

For the computational tests, we used four SC road networks with 500 nodes, which is the largest network size. They are called RN_1, RN_2, RN_3 and RN_4 in the following. Since there is only one type of vehicles and one type of customers in the $NEWLET$ instances, we had to extend these instances to fit our purposes. The new characteristics of our test instances with regard to the original $NEWLET$ instances are the following:

— A number of $n_c = 50$, 100 and 200 customers are randomly selected from the 500 nodes of a given network.

— For a given road network and n_c value, four customer distributions are considered:
  — D_1: 60% black customers, 20% green customers and 20% neutral customers.
  — D_2: 20% black customers, 60% green customers and 20% neutral customers.
  — D_3: 20% black customers, 20% green customers and 60% neutral customers.
  — D_4: 40% black customers, 40% green customers and 20% neutral customers.
— Each road network is divided into three regions: downtown, boundary (or frontier) and outside, as illustrated in Figure 13. In this figure, the light gray area represents downtown, while the dark gray area represents the boundary region. Orange triangles are transfer points, while the black and green squares are the black and green depot, respectively. Also, the black, green and gray nodes stand for black, green and neutral customers, respectively. The downtown area is defined by expanding a central and rectangular area until it can contain 80% of the maximum possible number of green customers (see distribution D_2), the green depot and additional road junctions. Then, the surrounding boundary or frontier region is expanded until it can contain the maximum possible number of neutral customers (see distribution D_3), 20% of the maximum possible number of black and green customers (see distributions D_1 and D_2), 10 transfer points and additional road junctions. The outside region contains the rest of the nodes.
— For all instances derived from a given road network, the three regions are the same as well as the location of the depots and transfer points.
— Black vehicles can perform deliveries to customers in the outside and boundary regions only. That is, downtown is forbidden to them. Conversely, the smaller green vehicles can perform deliveries to customers in the downtown and boundary regions only. For this reason, black customers are located in the outside and boundary regions, while green customers are located in the downtown and boundary regions. Neutral customers and transfer points are only found in the boundary region, since they can be visited by both black and green vehicles.
— The black depot is randomly located in the outside region, while the green depot is randomly located in the downtown region. There are also 10 transfer points that are randomly located in the boundary region.
— Arcs are characterized by the regions where they are found. Let (i,j) be an arc in the network. If both i and j are in the boundary (or frontier) region, then the arc is of type F (and is accessible to both black and green vehicles). If both i and j are in the downtown region, or one is in the downtown region and the other in the boundary region, then the arc is of type D (and is accessible only to green vehicles). If both i and j are in the outside region, or if one is in the outside region and the

other is in the boundary region, then the arc is of type O (and is accessible only to black vehicles). It should be noted that no arcs connect the downtown and outside regions. Given that the regions are the same for a given network, then the arc type also stays the same for all instances generated from a given road network.

— The test instances are duplicated by considering two different sets of travel speed multipliers (scenarios), where a speed multiplier depends on the time period, vehicle (black or green) and arc type (D, F, O), see Tables 1 and 2. In the two scenarios S_I and S_{II}, the second and fourth periods correspond to rush hours. The speed multipliers of green vehicles are fixed at 1 everywhere, which means that they are not affected by congestion since they are small (e.g., bicycles). Black vehicles are faster than green vehicles when there is no congestion. However, they are slower than green vehicles in the boundary region during rush hours in scenario S_I while they have the same speed than green vehicles in scenario S_{II}. The second scenario is aimed at evaluating the impact of increasing the speed of black vehicles when compared to green vehicles.

— The capacity of black vehicles is set to 40.

— The demand of each customer is randomly selected from $\{1, ..., 5\}$.

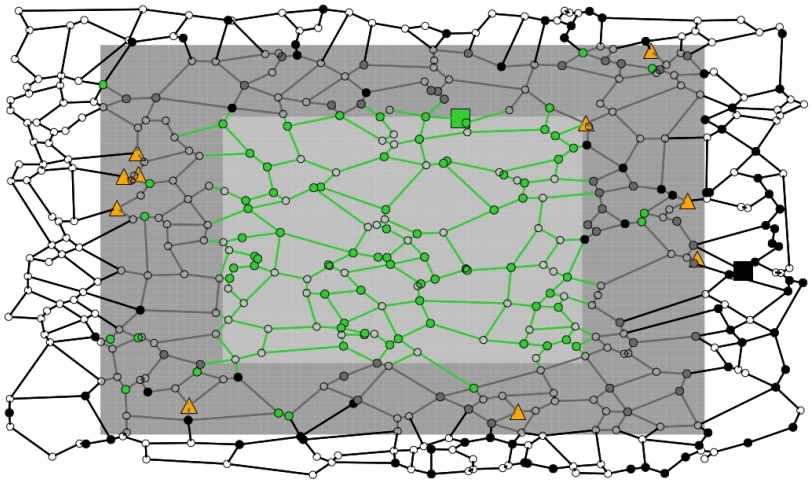

Figure 13. Example of an instance generated with road network RN_3; .

Overall, there are 4 road networks × 3 numbers of customers (n_c) × 4 customer distributions × 2 types of time windows × 2 scenarios for a total of 192 instances, that is, 96 instances for each scenario.

	Time period				
Arc type	$\tau_1 = [0,20)$	$\tau_2 = [20,30)$	$\tau_3 = [30,70)$	$\tau_4 = [70,80)$	$\tau_5 = [80,100)$
Type F	1.2	0.8	1.2	0.8	1.2
Type O	1.5	1.0	1.5	1.0	1.5

(a) Black vehicles

	Time period				
Arc type	$\tau_1 = [0,20)$	$\tau_2 = [20,30)$	$\tau_3 = [30,70)$	$\tau_4 = [70,80)$	$\tau_5 = [80,100)$
Type D	1.0	1.0	1.0	1.0	1.0
Type F	1.0	1.0	1.0	1.0	1.0

(b) Green vehicles

Table 1. Speed multipliers for (a) black vehicles and (b) green vehicles under scenario S_I

	Time period				
Arc type	$\tau_1 = [0,20)$	$\tau_2 = [20,30)$	$\tau_3 = [30,70)$	$\tau_4 = [70,80)$	$\tau_5 = [80,100)$
Type F	1.5	1.0	1.5	1.0	1.5
Type O	2.0	1.5	2.0	1.5	2.0

(a) Black vehicles

	Time period				
Arc type	$\tau_1 = [0,20)$	$\tau_2 = [20,30)$	$\tau_3 = [30,70)$	$\tau_4 = [70,80)$	$\tau_5 = [80,100)$
Type D	1.0	1.0	1.0	1.0	1.0
Type F	1.0	1.0	1.0	1.0	1.0

(b) Green vehicles

Table 2. Speed multipliers for (a) black vehicles and (b) green vehicles under scenario S_{II}

7.2. Parameter tuning

Four parameters have a significant impact on the performance of our SISR, namely, $\bar{c}$ (average number of removed customers), n_{pos} (number of best insertion positions, based on the approximation), n_{ftp} (number of nearest feasible transfer points) and L^{max} (maximum length of removed strings). To adjust their values, we selected a subset of 16 tuning instances with 100 customers by randomly selecting only one of the four networks, for each possible configuration of customer distribution (D_1, D_2, D_3, D_4), time window (NTW, WTW) and scenario (S_1, S_2). Given that solution quality tends to improve with increasing values of n_{pos} and n_{ftp}, at the expense of computation time, these two parameters were first set to high values, that is, $n_{pos} = 7$ and $n_{ftp} = 10$ (the latter value cannot be larger, since

there are only 10 transfer points in each instance). In other words, we did not care at this point about computation time. Then, we focused on parameters $\overline{c}$ and L^{max} and tuned them with the IRACE software [**57**], using $\overline{c} = \{5, ..., 17\}$ and $L^{max} = \{3, ..., 13\}$. The best values returned by IRACE were $\overline{c} = 15$ and $L^{max} = 8$. Based on the default configuration $\overline{c} = 15$, $n_{pos} = 7$, $n_{ftp} = 10$ and $L^{max} = 8$, we then modified the value of one parameter at a time, keeping the other parameters at their default value. The values considered for each parameter were: $\overline{c} = \{9, 11, 13, 15, 17, 19\}$, $n_{pos} = \{1, ..., 10\}$, $n_{ftp} = \{1, ..., 10\}$ and $L^{max} = \{1, ..., 10\}$. Since our algorithm is non deterministic, we show the average results (solution quality, computation time in hours) obtained over 10 runs on each tuning instance in Table 3.

As expected, increasing the values of parameters n_{pos} and n_{ftp} leads to an increase in computation time, although the impact is more significant in the case of n_{ftp}, since it increases the number of possible insertions of green and neutral customers in *Phase III* and *Phase IV* of the *Recreate* method (these insertions are quite complex). The computation times increase even more with increasing values of parameter $\overline{c}$ because more removed customers simply mean more customers to be reinserted. On the other hand, parameter L^{max} has no impact on computation time. With regard to solution quality, we observe an improvement in solution quality for the first values of each parameter, but then some kind of stagnation is observed. Accordingly, the parameter setting $\overline{c}$ =15, $n_{pos} = 3$, $n_{ftp} = 4$ and $L^{max} = 4$ was chosen for the experiments reported in the following sections. We also checked that this particular combination of parameter values led to good solutions on the tuning instances, which turned to be true with an average solution cost of 2522.2 and average computation time of 1.47 hours.

Some experiments were also performed with regard to the number of iterations. We observed that convergence is obtained, even on the largest instances with 200 customers, after a maximum of 300,000 iterations. That is, a plateau is reached and no further significant improvement in solution quality is observed. Figure 14 shows an example of convergence curves for the best solutions found on instances with 200 customers generated with road network RN_4, using the four customer distributions, and both narrow and wide time windows, under scenario S_I. In this figure, black, green, gray and blue curves are associated with customer distributions D_1, D_2, D_3 and D_4, respectively, while full lines and broken lines are associated with instances with narrow and wide time windows, respectively. Based on the results obtained, the number of iterations was set to 300,000 for all instances (which is admittedly too much for instances with 50 and 100 customers).

		Parameter values									
	$\bar{c}$ =	**9**	**11**	**13**	**15**	**17**	**19**				
Avg. cost		2552.1	2532.7	2523.3	2520.3	2520.1	2520.0				
Avg. time		1.29	1.54	1.81	2.10	2.36	2.56				
	n_{pos} =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Avg. cost		2537.9	2524.6	2521.4	2521.9	2520.3	2521.0	2520.3	2520.2	2519.6	2520.6
Avg. time		1.76	1.92	1.93	1.99	2.00	2.05	2.1	2.16	2.16	2.21
	n_{ftp} =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Avg. cost		2531.2	2525.5	2522.0	2520.2	2520.6	2518.9	2520.1	2520.3	2520.4	2520.3
Avg. time		0.83	1.15	1.41	1.61	1.75	1.88	1.97	2.02	2.06	2.10
	L^{max} =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Avg. cost		2542.9	2522.1	2520.2	2519.7	2520.4	2520.0	2520.1	2520.3	2519.8	2520.5
Avg. time		2.02	2.14	2.09	2.09	2.08	2.10	2.11	2.10	2.08	2.08

Table 3. Impact of parameter values on solution quality and computation times (in hours)

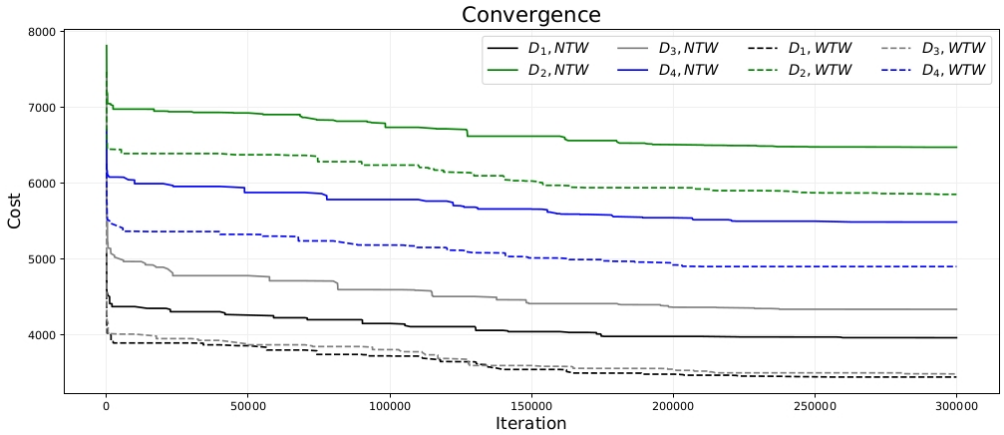

Figure 14. Convergence curves observed for instances with 200 customers generated with road network RN_4 under scenario S_I.

7.3. Results on test intances

We report in this section the results produced by our algorithm on the whole set of test instances, based on 10 different runs on each instance. Tables 8 and 9 in the Appendix reports the best and average costs, as well as the average computation times in hours for each instance. Each line of this table corresponds to a particular type of instance using the notation RNx_ny_Dz, where x is the road network index, y is the number of customers and z is the customer distribution index. For each type, we show the results obtained on

instances associated with narrow time windows (NTW) and wide time windows (WTW) under scenarios S_I and S_{II}. Table 4 in this section is a reduced version, where averages are taken over the four road networks. That is, ny_Dz in Table 4 encompasses $RN1\_ny\_Dz$, $RN2\_ny\_Dz$, $RN3\_ny\_Dz$ and $RN4\_ny\_Dz$, so that the numbers in Table 4 correspond to the bold Avg. lines in the full table in the Appendix.

	NTW						**WTW**					
Instances	**Avg. Best Cost**		**Avg. Cost**		**Avg. Time**		**Avg. Best Cost**		**Avg. Cost**		**Avg. Time**	
	S_I	**S_{II}**	**S_I**	**S_{II}**	**S_I**	**S_{II}**	**S_I**	**S_{II}**	**S_I**	**S_{II}**	**S_I**	**S_{II}**
n50_D1	1397.8	1125.3	1398.9	1126.6	0.27	0.28	1219.0	929.5	1220.5	930.0	0.26	0.26
n50_D2	2002.5	1841.0	2004.5	1844.5	0.77	0.87	1801.4	1608.4	1805.2	1615.5	0.76	0.81
n50_D3	1368.4	1112.9	1374.8	1118.0	0.66	0.69	1117.0	899.2	1117.0	901.8	0.59	0.54
n50_D4	1748.9	1517.1	1752.0	1520.4	0.53	0.56	1542.4	1279.1	1543.8	1280.3	0.51	0.51
n100_D1	2498.1	2030.9	2505.7	2034.1	0.64	0.70	2128.9	1702.6	2140.4	1710.0	0.64	0.65
n100_D2	3684.7	3298.5	3696.0	3308.6	2.18	2.37	3314.9	2909.0	3326.5	2922.0	2.06	2.13
n100_D3	2368.0	1976.9	2372.9	1979.9	1.86	1.92	1916.3	1564.2	1921.6	1572.9	1.42	1.39
n100_D4	3138.3	2734.4	3148.2	2756.3	1.34	1.55	2743.1	2303.3	2754.8	2316.5	1.35	1.34
n200_D1	4186.9	3461.1	4221.5	3489.7	1.71	1.88	3582.6	2822.7	3610.5	2855.8	1.42	1.57
n200_D2	6980.9	6356.7	7028.4	6408.2	5.76	6.47	6321.5	5691.5	6380.1	5758.7	5.47	5.69
n200_D3	4399.9	3638.8	4439.4	3672.7	4.54	5.03	3636.4	2918.6	3663.4	2941.5	3.44	3.52
n200_D4	5613.9	4894.5	5642.2	4930.7	3.70	4.21	4927.8	4255.0	4967.9	4294.4	3.41	3.57
Overall Avg.	**3282.4**	**2832.3**	**3298.7**	**2849.1**	**2.00**	**2.21**	**2854.3**	**2407.0**	**2871.0**	**2425.0**	**1.78**	**1.83**

Table 4. Solution cost and computation time in hours for each subset of instances

Since there are no similar instances in the literature that we could refer to for comparison purposes, Table 5 reports both the average best improvements and average improvements provided by our algorithm over the initial solutions, to measure its optimization power. Denoting s_i and s_f the initial and final solutions produced by our algorithm on a given instance, with $cost(s_i)$ and $cost(s_f)$ their respective cost, the percentage of improvement of the final solution over the initial one is calculated as follow:

$$Impr = 100 \left(\frac{cost(s_i) - cost(s_f)}{cost(s_i)} \right) \tag{7.1}$$

Tables 4 and 5 will be referred to in the following sections when we analyze in more detail the behavior of our algorithm for different characteristics of the test instances. For now, we can observe the obvious increase in solution cost and computation time with instance size in Table 4. We also note the overall average improvements over the initial solutions in Table 5 that stand between 20% and 30%, which is substantial. It is clear that the greedy insertion heuristic is limited with regard to solution quality on such a complex problem, but still, these percentages of improvement show that our algorithm can take advantage of optimization opportunities.

Instances	NTW				WTW			
	Avg. Best Impr.		Avg. Impr.		Avg. Best Impr.		Avg. Impr.	
	S_I	S_{II}	S_I	S_{II}	S_I	S_{II}	S_I	S_{II}
n50_D1	13.02	14.14	10.48	11.19	16.03	18.38	13.48	15.03
n50_D2	13.28	15.33	11.21	13.15	16.75	19.36	14.28	16.51
n50_D3	33.53	41.27	30.95	38.51	35.42	43.14	32.02	39.05
n50_D4	11.20	14.87	9.49	11.64	15.06	18.21	12.62	15.52
n100_D1	17.59	20.28	15.96	18.46	22.32	24.56	20.33	22.70
n100_D2	17.55	19.68	15.34	17.67	19.48	22.70	17.63	20.68
n100_D3	41.76	46.49	39.68	44.93	44.31	48.55	42.14	45.87
n100_D4	14.67	17.62	13.27	15.28	20.16	22.18	17.98	20.56
n200_D1	25.57	27.73	24.00	25.46	29.57	31.24	27.73	29.54
n200_D2	18.76	22.39	17.32	21.05	21.01	23.97	19.50	22.20
n200_D3	46.53	52.75	45.27	51.05	48.17	54.24	46.51	52.07
n200_D4	19.75	22.61	18.46	20.89	23.95	25.71	21.95	23.81
Overall Avg.	**22.77**	**26.26**	**20.95**	**24.11**	**26.02**	**29.35**	**23.85**	**26.96**

Table 5. Average best improvements and average improvements for each subset of instances.

7.4. Synchronization

In this section, we examine if synchronization between black and green vehicles at transfer points is efficient. For this purpose, we define the percentage of solution cost (duration) that corresponds to the total time that black and green vehicles spend at transfer points. For a given solution s, this percentage is denoted as ρ_s. In Equation (7.2), this percentage is defined using ΔTP_s^b and ΔTP_s^g, which are the total time spent at transfer points by black and green vehicles, respectively.

$$\rho_s = 100 \left(\frac{\Delta TP_s^b + \Delta TP_s^g}{cost(s)} \right) \tag{7.2}$$

Table 6 reports the minimum, maximum and average values of ρ_s on different subsets of instances. The format of this table is reduced when compared to Tables 4 and 5 by also averaging over the two types of time windows and the two scenarios. We observe that the ρ_s values are smaller for customer distributions D_1 and D_3. In fact, if we compute the average ρ_s values for D_1, D_2, D_3 and D_4, we obtain 0.97%, 1.97%, 0.94% and 1.57%, respectively. The fact that D_1 and D_3 lead to better synchronization than D_2 and D_4 can be explained ty their small percentage of green customers (20%), since they are the only ones for which synchronization at a transfer point is mandatory. In any case, the differences observed are small in absolute terms. The fact is that only 1.36% (overall average) of

solution cost is due to synchronization, which indicates that synchronization is well achieved.

Instances	Min	Max	Avg.
n50_D1	0.12	2.18	0.77
n50_D2	1.05	3.71	2.08
n50_D3	0.03	2.55	1.02
n50_D4	0.18	3.23	1.60
Avg.	**0.34**	**2.92**	**1.37**
n100_D1	0.01	3.03	0.90
n100_D2	0.82	3.11	1.91
n100_D3	0.19	1.97	0.95
n100_D4	0.45	3.13	1.48
Avg.	**0.37**	**2.81**	**1.31**
n200_D1	0.45	2.45	1.25
n200_D2	0.84	3.37	1.91
n200_D3	0.25	2.20	0.86
n200_D4	0.53	2.85	1.63
Avg.	**0.52**	**2.71**	**1.41**
Overall Avg.	**0.01**	**3.71**	**1.36**

Table 6. Minimum, maximum and average values of ρ_s for each subset of instances

7.5. Customer distributions

Table 4 shows that customer distributions D_1 and D_3 are the best for solution cost, while D_1 outperforms the three other distributions for computation time. Both D_1 and D_3 have only 20% of green customers, which is beneficial for solution cost because these customers require a detour at a transfer point for both a black and a green vehicle. Furthermore, D_1 has a large percentage of 60% of black customers (versus only 20% of neutral customers), which is helpful for computation time because only simple insertions in black routes need to be considered for black customers. Although D_3 is competitive with D_1 for solution cost, this is not the case for computation time. Distribution D_3 has 60% of neutral customers (versus only 20% of black customers) and their potential insertion in both black and green routes need to be considered. As opposed to black routes, insertion in green routes is quite complex and computationally expensive. With the largest percentage of 60% of green customers, distribution D_2 is consequently the worst for solution cost and computation time.

When considering improvements over initial solutions in Table 5, the largest improvements are associated with distribution D_3. This distribution has the largest percentage of neutral customers (60%) and these customers offer more flexibility for optimization because

they can be inserted either in black or green routes. In this case, the average percentage of neutral customers belonging to black routes in the initial solutions ranges between 51.1% and 54.4%. However, in the final solutions, these percentages drastically increase between 94.6% and 98.6%. That is, the optimization algorithm finds ways to move a large proportion of neutral customers in black routes, which decreases solution cost. Accordingly, more neutral customers means more opportunities for improvement.

7.6. Time windows

Tables 4 and 5 show that better solution costs and larger improvements over initial solutions are associated with instances with wide time windows. Clearly, these time windows offer more feasible insertion places for customers and, consequently, greater flexibility for the optimization procedure to move them around. For example, the percentage of neutral customers in black routes for the instances with wide time windows is 98.3%, as compared with 92.8% for instances with narrow time windows. We also observed a reduced number of routes in the solutions obtained on instances with wide time windows, when compared with narrow time windows, namely 25.2% less black routes and 15.1% less green routes. Finally, black vehicles visit 2% more transfer points on average in the presence of wide time windows.

7.7. Scenarios

Better solution costs are observed in Table 4 under scenario S_{II}, when compared with scenario S_I. Since black vehicles now travel faster, the time windows at black and neutral customers become easier to satisfy. We also observed that, on average, a larger percentage of neutral customers is served by black vehicles in the final solutions under scenario S_{II} (97.4%) when compared to scenario S_I (93.7%). It is generally less costly to serve neutral customers with black vehicles and the latter can get a larger share of neutral customers when their speed increases. In other words, they win more often the "battle" for neutral customers against green vehicles in the boundary region. We also observed a reduction of 14.7% and 6.2% in the number of black and green routes, respectively, under scenario S_{II}, with corresponding increases of 20.5% and 3.5% in the average number of customers in black and green routes, respectively. Finaly, black vehicles visit 4.4% more transfer points on average, when compared to scenario S_I.

7.8. Number of transfer points

We propose here an experiment where we gradually reduce the number of transfer points to see the impact on the solutions obtained. To this end, some transfer points are randomly selected and transformed into simple nodes (road junctions). In some cases, infeasible instances may be created (e.g., it may not be possible for any green vehicle to

visit a transfer point and serve a green customer before the upper bound of its time window).

Here, we focus on the test instances with 100 customers. First, two transfer points are randomly chosen and removed from the 10 original ones to obtain instances with 8 transfer points. From these 8 transfer points, the procedure is repeated to get instances with 6 transfer points. Finally, two additional transfer points are removed to get instances with 4 transfer points. Ten runs were performed on each feasible instance with a reduced number of transfer points. Table 7 shows the average gaps between the average cost of solutions obtained with 10 transfer points and with $k = 8, 6, 4$ transfer points, based on Equation (7.3). Each line in this table is the average over four instances, considering that there are two types of time windows and two scenarios.

$$Gap = 100 \left(\frac{ave_{cost}^{|TP|=k} - ave_{cost}^{|TP|=10}}{ave_{cost}^{|TP|=10}} \right) \tag{7.3}$$

We observed no infeasible instance with $k = 8$ transfer points, one infeasible instance with $k = 6$ transfer points and 19 infeasible instances (out of 64 instances) with $k = 4$. Thus, some averages are computed with less than four gaps. When there is no value, the four instances are infeasible. A few small negative values appear in the table, which means that slightly better solutions are obtained with fewer transfer points. This situation can occur, due to the randomized nature of our algorithm, when the removed transfer points are not or seldom used in the original instances, thus producing no or little impact on solution quality.

Obviously, removing transfer points generally lead to worse solutions and this trend is more pronounced when more transfer points are removed. Customer distribution D_2 is more affected than the other distributions due to its large percentage of green customers (which must use transfer points). We also see that instances associated with road network RN_1 are greatly affected when 4 or 6 transfer points are removed. In the solutions obtained with 10 transfer points, we observed that some transfer points are much more exploited than others. Clearly, such critical transfer points are more likely to disappear when more transfer points are removed, which in turn greatly impact solution quality.

8. Conclusion

In this work, we have proposed a new SISR metaheuristic for the $TDVRPTWTP_{RN}$. To the best of our knowledge, this challenging problem where customers can be served either directly by black vehicles or indirectly by green vehicles through transfer points has not been previously addressed in the literature. Computational results on test instances with

Instances	Avg. Gap		
	$k = 8$	$k = 6$	$k = 4$
RN1_n100_D1	-0.04	6.02	– – –
RN2_n100_D1	-0.10	-0.11	0.47
RN3_n100_D1	-0.02	-0.04	0.51
RN4_n100_D1	0.51	1.74	1.76
RN1_n100_D2	2.06	10.95	27.45
RN2_n100_D2	1.56	1.58	5.86
RN3_n100_D2	0.62	1.56	9.08
RN4_n100_D2	1.81	7.09	7.30
RN1_n100_D3	0.48	7.32	16.08
RN2_n100_D3	-0.05	-0.09	0.48
RN3_n100_D3	1.12	2.34	6.55
RN4_n100_D3	1.24	1.79	1.82
RN1_n100_D4	1.00	9.72	– – –
RN2_n100_D4	-0.86	-0.75	-0.02
RN3_n100_D4	0.19	1.21	4.45
RN4_n100_D4	0.86	1.47	1.68

Table 7. Average gaps between average solution costs with 10 transfer points and $k = 8$, 6, 4 transfer points

different characteristics show that our algorithm performs well, in particular by finding ways to transfer more neutral customers into black routes, which lead to solutions of better quality. Furthermore, we observed that the time spent by vehicles at transfer points is very low, thus indicating that good synchronization is achieved.

For the future, new ruin operators could be developed to enhance the performance of our algorithm and solve other complex variants of vehicle routing problems. It would also be interesting to consider the integration of learning into our algorithm, for example to identify the most promising transfer points, based on the topology of the network and distribution of customers.

Appendix

Instances	Best Cost (10 runs)		Avg. Cost (10 runs)		Avg. Time (hours)	
	S_I	S_{II}	S_I	S_{II}	S_I	S_{II}
RN1_n50_D1	1563.47	1217.90	1563.91	1217.90	0.30	0.29
RN2_n50_D1	1365.48	1096.91	1365.48	1096.91	0.25	0.23
RN3_n50_D1	1430.09	1159.05	1432.17	1159.65	0.25	0.27
RN4_n50_D1	1232.26	1027.51	1233.85	1031.95	0.28	0.34
Avg.	**1397.83**	**1125.34**	**1398.85**	**1126.60**	**0.27**	**0.28**
RN1_n50_D2	2295.75	2067.62	2295.83	2069.44	0.81	0.90
RN2_n50_D2	1929.28	1752.89	1932.68	1753.13	0.74	0.81
RN3_n50_D2	1996.08	1864.24	1996.14	1875.20	0.69	0.71
RN4_n50_D2	1788.88	1679.38	1793.19	1680.13	0.83	1.06
Avg.	**2002.50**	**1841.03**	**2004.46**	**1844.48**	**0.77**	**0.87**
RN1_n50_D3	1440.17	1183.37	1449.07	1184.69	0.65	0.68
RN2_n50_D3	1365.98	1087.79	1378.58	1105.24	0.72	0.66
RN3_n50_D3	1353.17	1120.44	1356.42	1121.83	0.57	0.61
RN4_n50_D3	1314.31	1059.94	1314.97	1060.22	0.70	0.78
Avg.	**1368.41**	**1112.88**	**1374.76**	**1118.00**	**0.66**	**0.69**
RN1_n50_D4	1786.66	1587.00	1786.66	1587.00	0.49	0.53
RN2_n50_D4	1944.47	1629.15	1947.70	1630.98	0.54	0.51
RN3_n50_D4	1535.58	1376.85	1537.07	1386.52	0.48	0.47
RN4_n50_D4	1728.69	1475.37	1736.48	1477.26	0.61	0.72
Avg.	**1748.85**	**1517.09**	**1751.98**	**1520.44**	**0.53**	**0.56**
RN1_n100_D1	2526.91	2085.89	2528.87	2088.53	0.64	0.70
RN2_n100_D1	2559.89	2067.83	2570.32	2071.26	0.53	0.65
RN3_n100_D1	2540.37	2113.93	2540.48	2115.17	0.64	0.65
RN4_n100_D1	2365.21	1856.08	2383.25	1861.46	0.73	0.81
Avg.	**2498.09**	**2030.93**	**2505.73**	**2034.11**	**0.64**	**0.70**
RN1_n100_D2	3804.11	3452.62	3812.33	3461.66	2.07	2.22
RN2_n100_D2	3537.95	3237.98	3543.50	3246.02	1.84	2.37
RN3_n100_D2	3919.34	3520.71	3936.62	3534.99	2.12	2.18
RN4_n100_D2	3477.32	2982.62	3491.33	2991.65	2.69	2.72
Avg.	**3684.68**	**3298.48**	**3695.95**	**3308.58**	**2.18**	**2.37**
RN1_n100_D3	2480.34	1996.41	2488.51	1997.98	1.83	1.70
RN2_n100_D3	2315.24	1990.81	2319.78	1996.14	1.70	1.71
RN3_n100_D3	2435.62	2051.60	2440.28	2054.78	2.09	2.03
RN4_n100_D3	2240.82	1868.86	2243.17	1870.65	1.82	2.23
Avg.	**2368.00**	**1976.92**	**2372.94**	**1979.89**	**1.86**	**1.92**
RN1_n100_D4	3534.04	2929.31	3538.32	2940.50	1.56	1.65
RN2_n100_D4	2807.13	2505.56	2827.05	2573.06	1.09	1.27
RN3_n100_D4	3161.54	2876.90	3168.41	2882.75	1.28	1.64
RN4_n100_D4	3050.61	2625.61	3058.81	2628.78	1.43	1.65
Avg.	**3138.33**	**2734.35**	**3148.15**	**2756.27**	**1.34**	**1.55**
RN1_n200_D1	4586.99	3673.59	4607.01	3698.37	1.97	1.84
RN2_n200_D1	4006.30	3338.27	4037.61	3380.21	1.41	1.76
RN3_n200_D1	4235.06	3605.87	4287.61	3632.18	1.83	1.97
RN4_n200_D1	3919.20	3226.55	3953.88	3247.98	1.62	1.95
Avg.	**4186.89**	**3461.07**	**4221.53**	**3489.68**	**1.71**	**1.88**
RN1_n200_D2	7428.48	6656.70	7468.67	6667.52	5.64	6.09
RN2_n200_D2	6883.99	6320.66	6938.59	6428.07	5.29	6.62
RN3_n200_D2	7143.03	6595.11	7201.80	6636.35	6.35	6.69
RN4_n200_D2	6468.03	5854.25	6504.63	5900.71	5.76	6.48
Avg.	**6980.88**	**6356.68**	**7028.42**	**6408.16**	**5.76**	**6.47**
RN1_n200_D3	4333.66	3631.80	4385.00	3671.93	4.12	4.57
RN2_n200_D3	4336.39	3638.83	4385.59	3681.60	4.46	5.27
RN3_n200_D3	4600.69	3735.66	4628.11	3767.51	4.49	4.84
RN4_n200_D3	4328.94	3548.88	4358.82	3569.77	5.06	5.45
Avg.	**4399.92**	**3638.79**	**4439.38**	**3672.70**	**4.54**	**5.03**
RN1_n200_D4	5863.31	5023.25	5872.71	5048.40	4.19	4.35
RN2_n200_D4	5302.13	4717.90	5314.75	4770.30	2.82	3.42
RN3_n200_D4	5842.90	5220.16	5902.48	5258.46	4.38	4.93
RN4_n200_D4	5447.38	4616.55	5478.77	4645.50	3.42	4.17
Avg.	**5613.93**	**4894.47**	**5642.18**	**4930.67**	**3.70**	**4.21**
Overall avg.	**3282.36**	**2832.34**	**3298.69**	**2849.13**	**2.00**	**2.21**

Table 8. Solution cost and computation time in hours for each instance with narrow time windows.

Instances	Best Cost (10 runs)		Avg. Cost (10 runs)		Avg. Time (hours)	
	S_I	S_{II}	S_I	S_{II}	S_I	S_{II}
RN1_n50_D1	1331.90	1030.23	1332.03	1030.23	0.26	0.29
RN2_n50_D1	1159.71	898.09	1165.04	898.09	0.26	0.24
RN3_n50_D1	1292.61	950.84	1292.81	950.85	0.25	0.24
RN4_n50_D1	1091.71	839.01	1092.13	840.98	0.27	0.28
Avg.	**1218.98**	**929.54**	**1220.50**	**930.04**	**0.26**	**0.26**
RN1_n50_D2	2018.20	1737.63	2018.72	1742.08	0.82	0.89
RN2_n50_D2	1664.44	1476.99	1664.84	1478.37	0.71	0.77
RN3_n50_D2	1843.62	1696.06	1855.69	1713.47	0.63	0.71
RN4_n50_D2	1679.18	1523.08	1681.66	1527.97	0.88	0.88
Avg.	**1801.36**	**1608.44**	**1805.23**	**1615.47**	**0.76**	**0.81**
RN1_n50_D3	1234.84	983.12	1234.86	985.14	0.61	0.53
RN2_n50_D3	1059.19	845.14	1059.19	853.41	0.57	0.53
RN3_n50_D3	1094.61	931.62	1094.61	931.62	0.59	0.53
RN4_n50_D3	1079.52	836.87	1079.52	837.14	0.60	0.58
Avg.	**1117.04**	**899.19**	**1117.05**	**901.83**	**0.59**	**0.54**
RN1_n50_D4	1679.56	1365.57	1679.56	1365.71	0.56	0.47
RN2_n50_D4	1622.76	1328.29	1622.76	1330.34	0.57	0.61
RN3_n50_D4	1355.06	1195.54	1355.06	1197.90	0.41	0.45
RN4_n50_D4	1512.20	1227.06	1517.89	1227.06	0.49	0.51
Avg.	**1542.39**	**1279.12**	**1543.82**	**1280.25**	**0.51**	**0.51**
RN1_n100_D1	2189.86	1662.22	2198.75	1675.03	0.65	0.56
RN2_n100_D1	2161.96	1726.10	2177.80	1727.87	0.56	0.59
RN3_n100_D1	2200.53	1889.07	2202.66	1891.56	0.70	0.78
RN4_n100_D1	1963.41	1533.17	1982.42	1545.58	0.63	0.69
Avg.	**2128.94**	**1702.64**	**2140.41**	**1710.01**	**0.64**	**0.65**
RN1_n100_D2	3426.28	2988.77	3434.23	3006.73	2.03	1.99
RN2_n100_D2	3256.69	2866.68	3265.31	2873.14	2.05	2.03
RN3_n100_D2	3538.40	3158.21	3551.07	3168.52	1.79	1.98
RN4_n100_D2	3038.18	2622.38	3055.51	2639.41	2.36	2.53
Avg.	**3314.89**	**2909.01**	**3326.53**	**2921.95**	**2.06**	**2.13**
RN1_n100_D3	2124.82	1635.38	2127.03	1645.98	1.58	1.29
RN2_n100_D3	1923.52	1612.65	1930.14	1618.16	1.37	1.44
RN3_n100_D3	1851.77	1569.09	1856.39	1573.41	1.32	1.36
RN4_n100_D3	1764.91	1439.53	1772.65	1454.09	1.40	1.47
Avg.	**1916.26**	**1564.16**	**1921.55**	**1572.91**	**1.42**	**1.39**
RN1_n100_D4	3044.64	2398.28	3048.74	2415.50	1.43	1.31
RN2_n100_D4	2444.41	2141.93	2456.91	2151.48	1.07	1.16
RN3_n100_D4	2742.29	2428.80	2765.12	2448.75	1.48	1.50
RN4_n100_D4	2741.18	2244.28	2748.61	2250.31	1.41	1.38
Avg.	**2743.13**	**2303.32**	**2754.84**	**2316.51**	**1.35**	**1.34**
RN1_n200_D1	3890.26	2964.06	3911.65	3002.76	1.59	1.63
RN2_n200_D1	3420.59	2774.14	3444.88	2805.14	1.32	1.62
RN3_n200_D1	3632.22	3011.05	3649.54	3032.49	1.27	1.43
RN4_n200_D1	3387.47	2541.72	3436.05	2582.96	1.52	1.59
Avg.	**3582.63**	**2822.74**	**3610.53**	**2855.84**	**1.42**	**1.57**
RN1_n200_D2	6650.45	5893.97	6692.08	5936.45	5.23	5.60
RN2_n200_D2	6220.31	5650.36	6267.01	5712.54	5.40	5.85
RN3_n200_D2	6615.83	6020.32	6681.27	6079.96	5.38	5.24
RN4_n200_D2	5799.34	5201.53	5879.89	5306.00	5.86	6.07
Avg.	**6321.48**	**5691.54**	**6380.06**	**5758.74**	**5.47**	**5.69**
RN1_n200_D3	3647.47	2941.04	3684.66	2960.16	2.97	2.99
RN2_n200_D3	3622.43	2916.56	3644.45	2943.40	3.56	4.12
RN3_n200_D3	3818.68	3096.20	3835.78	3112.43	3.53	3.35
RN4_n200_D3	3457.12	2720.75	3488.66	2749.87	3.69	3.63
Avg.	**3636.43**	**2918.64**	**3663.39**	**2941.46**	**3.44**	**3.52**
RN1_n200_D4	5056.61	4284.61	5081.22	4323.29	3.37	3.26
RN2_n200_D4	4661.26	4074.33	4685.65	4125.05	2.93	3.15
RN3_n200_D4	5141.24	4588.17	5213.82	4629.61	3.86	3.95
RN4_n200_D4	4851.89	4073.05	4890.97	4099.56	3.50	3.93
Avg.	**4927.75**	**4255.04**	**4967.92**	**4294.38**	**3.41**	**3.57**
Overall avg.	**2854.27**	**2406.95**	**2870.99**	**2424.95**	**1.78**	**1.83**

Table 9. Solution cost and computation time in hours for each instance with wide time windows.

Second Article.

Impact of Distance Data Inaccuracies on Vehicle Routing Algorithms: An Experimental Study

by

Fernando Obed Guillen Reyes[1], Jean-Yves Potvin[2], Michel Gendreau[3], and Thibaut Vidal[4]

([1]) Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

([2]) Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

([3]) Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

([4]) Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

SCALE-AI Chair in Data-Driven Supply Chains.

This article was submitted to the Journal of the Operational Research Society.

The main contributions of Fernando Obed Guillen Reyes for this articles are presented. He designed the empirical study with the help of his advisors. He generated different levels of inaccuracies in the distances of classical benchmark instances. Then, he had to adapt previous implementations of the selected problem-solving methods to make them operational within the experimental setting. He ran the experiments and analyzed the results with the help of his advisors. He also wrote the first draft of the paper.

RÉSUMÉ. Cet article s'intéresse à l'impact d'inexactitudes dans les distances sur la performance de différentes méthodes pour résoudre le problème de tournées de véhicules avec contraintes de capacité. Il s'agit d'une étude expérimentale qui inclut une heuristique simple, des métaheuristiques qui représentent l'état de l'art et un algorithme exact. Nous voulons répondre aux questions suivantes: i) vaut-il la peine de faire appel à des algorithmes sophistiqués quand les inexactitudes dans les données sont plus grandes ou similaires aux écarts d'optimalité typiquement produits par de tels algorithmes; ii) les meilleurs algorithmes seront-ils attirés davantage par des distances inexactes (e.g., erronément trop courtes) et seront-ils dès lors disproportionnellement affectés par ces distances inexactes? À cet effet, nous menons une étude expérimentale pour le problème de tournées de véhicules avec contraintes de capacité en faisant appel à des instances tests contenant de 100 à 1 000 clients en introduisant différents niveaux d'inexactitude dans les distances. Suite à cette étude, nous pouvons répondre affirmativement à la première question et négativement à la seconde: les erreurs dans les données et les algorithmes s'influencent mutuellement, si bien que les meilleurs algorithmes demeurent les meilleurs dans tous les cas. De plus, l'impact des distances inexactes sur la performance est assez semblable d'un algorithme à l'autre, sauf pour l'heuristique de construction simple de Clarke et Wright qui semble disproportionnellement affectée par ces inexactitudes.

Mots clés : Étude expérimentale, distances inexactes, heuristiques, métaheuristiques, algorithme exact, problème de tournées de véhicules avec contraintes de capacité

Abstract.

This paper studies the impact of distance data inaccuracies on different methods for solving the capacitated vehicle routing problem. It is an experimental study examining how simple heuristics, state-of-the-art metaheuristics and an exact algorithm behave under such inaccuracies. We investigate the two following questions: i) is it worth using sophisticated state-of-the-art algorithms when data inaccuracies are greater or similar in their scale to algorithmic error gaps; ii) would the best algorithms tend to select arcs with inaccurate (e.g., erroneously shorter) distances and be disproportionally affected by inaccurate data? We conduct extensive experiments on the capacitated vehicle routing problem, considering instances ranging from 100 to 1,000 customers with different distance-estimation inaccuracies. Interestingly, we respond to the first question in the affirmative and the second one in the negative: errors from data and algorithms tend to compound, such that state-of-the-art algorithms remain the better choice in all cases. Moreover, the impact of inaccurate data is fairly uniform over the algorithms, except for a simpler construction heuristic (Clarke and Wright algorithm) which seems disproportionally impacted by inaccurate data.
Keywords: Experimental study, inaccurate distances, heuristics, metaheuristics, exact algorithm, capacitated vehicle routing problem

1. Introduction

The Capacitated Vehicle Routing Problem (CVRP) seeks to find efficient routes to transport goods from a depot to several customers using a fleet of vehicles limited by their capacity. This problem is of significant interest to researchers because any improvement in the methods developed for it can result in significant savings for transportation companies. Although many complex variants of the CVRP have been studied in the last decades, the canonical CVRP remains challenging for large-scale instances. In this case, one must often resort to heuristics and metaheuristics to produce solutions of good quality in a reasonable amount of computation time.

Formally, the CVRP can be stated as follows. We consider a complete graph $G = (V, A)$, where V is the set of nodes and A is the set of arcs. The set V contains the customers plus the depot. Each customer $i \in V$ has a demand q_i that must be served exactly once by a vehicle. Each arc $(i,j) \in A$ between node i and node j is associated with a distance d_{ij}. A fleet of homogeneous vehicles of capacity Q is located at the depot. Each vehicle performs one closed route that starts at the depot, delivers the demand to a subset of customers, and returns to the depot at the end. The goal is to assign all customers to vehicles and sequence the customers assigned to each vehicle to minimize the total distance (or travel time) while not exceeding the capacity of any vehicle.

In most practical cases, data (e.g., travel times) can be inaccurate due to the devices and methods used to collect them, as well as possible human errors. In view of this, though

the development of more and more accurate heuristics is interesting from a methodological standpoint, many practitioners would argue that sophisticated heuristics capable of approximating the exact solutions within a fraction of percent errors do not pay off, given that the magnitude of the errors in the data is likely larger than that of nowadays heuristics. Going even further, in a similar fashion as learning algorithms, some could suspect that state-of-the-art optimization algorithms applied to inaccurate data may "overfit" and deliberately use arcs whose distances are most underestimated.

In this paper, we conduct extensive experimental analyses to examine these common beliefs. To this end, we design a controlled experiment considering four CVRP heuristics and an exact method tested on instances ranging from 100 to 1,000 customers with different distance-estimation inaccuracies. We show that the two aforementioned beliefs are incorrect and that state-of-the-art methods consistently produce solutions of better quality even when considering the largest level of inaccuracy in the data. Moreover, the impact of inaccurate data is fairly consistent over the algorithms, except for the Clarke-and-Wright construction algorithm, which seems disproportionally impacted by inaccurate data.

The rest of this paper is organized as follows. Section 2 reviews related works, mostly from the machine learning literature, in which the impact of inaccurate data on algorithmic performance is studied. Section 3 presents our experimental setting, the heuristics considered, and the generation of test instances with inaccurate distances from the datasets presented in [**82**] and the 2021 DIMACS implementation challenge on vehicle routing problems [**24**]. Section 4 discusses our computational results and Section 5 concludes.

2. Related works

Extensive research has been conducted on the CVRP, but no paper has, to our knowledge, analyzed the impact of data inaccuracies on the results of classical heuristics, metaheuristics, and exact methods. The closest related analyses come from the machine learning and data mining literature where the impact of noise has been studied. Given that noise is unavoidable in real-world problems, several methods in machine learning have been aimed at identifying different types of noise and ways to handle them. For example, the work reported in [**43**] reviews uncertainty and big data analytics and discusses the impact of different sources of errors in the data and their consequences. Methods to deal with noise and to measure it are also reported. The authors in [**41**] additionally survey studies published between 1993 and 2018 about noise identification and mitigation.

Outside of the optimization and vehicle routing domain, many papers in machine learning study the impact of inaccurate data on solutions produced by different algorithms. The following papers share a common approach by comparing the sensitivity of different algorithms to different types and levels of noise in the data. In [**3, 61, 75, 77**], simple learning algorithms like decision trees, naive Bayesian classifiers, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and logistic regression are considered. In [**77**], a new metric called Equalized Loss of Accuracy (ELA) is proposed to evaluate different algorithms based on their overall performance and their robustness to noise. Moreover, the work in [**17**] reports an extensive comparison of different Bayesian network structure learning algorithms under noise.

Multiple classifier systems (MCSs) that combine different learning algorithms have also attracted the attention of researchers. MCSs are obtained by combining different learning algorithms. Although MCSs are totally different from the algorithms used to solve the CVRP, it is also quite common to combine different heuristics in problem-solving methods for the CVRP. In [**76**], the authors study the behavior of MCSs under noise by combining the decision tree algorithm C4.5, SVM and k-NN. Similar studies are reported in [**23, 59, 50**] using MCSs based on AdaBoost, boosting, randomization, and bagging techniques.

In the data mining field, statistical analyses on the impact of noise (and unbalanced datasets) on the performance of 11 learning algorithms have been reported in [**84**]. In this work, the authors added noise to seven datasets while controlling the overall amount of corrupted data. Through an analysis of variance (ANOVA) they determine which factors, like the level of noise or the percentage of noisy data in a given class, impact the most the considered learning algorithms.

Overall, those studies indicate that noise is almost always present in real-world applications and has a considerable impact on the performance of learning algorithms. Also, it is observed that different types of noise might impact a particular algorithm differently. Thus, accuracy is not the only desirable feature of a learning algorithm, but also its robustness to noise.

3. Experimental setting

In this section, we describe the experimental setting designed to evaluate the impact of noise on different problem-solving methods for the CVRP. First, the tested heuristics and metaheuristics are introduced. Then, the benchmark instances are described. Finally, we

explain how perturbations to these instances were generated to include different levels of noise in the distances.

3.1. Solution Algorithms

A large variety of heuristics have been proposed to solve the CVRP, from simple heuristics to sophisticated metaheuristics. To analyze the impact of noise on different types of methods, we considered the following approaches:

(a) The *Clarke & Wright savings heuristic* (CW) [**16**] starts with an initial solution in which an individual route serves each customer. Then, it evaluates for each existing route the "distance savings" due to merging each route pair. These savings are then sorted from largest to smallest. Starting at the top of the list of savings, the two routes associated with the current saving are merged to form a single route. This process is then repeated until no pair of routes can be merged without violating the capacity constraints.

(b) The open source software *Google OR-Tools* (GORT) [**66**] is a generic tool designed to solve various operations research problems, including VRPs. Here, the user must define the data (instance) and write a high-level algorithm that calls the required predefined functions to solve a problem. For example, the user defines the method to compute the initial solution, the local search method, the stopping criterion, etc. In our case, we chose PATH_CHEAPEST_ARC to generate the initial solution and GUIDED_LOCAL_SEARCH to perform the local search, as recommended in the user manual for VRPs.

(c) The *Slack Induction by String Removal* (SISR) [**15**] is a recent metaheuristic based on the ruin-and-recreate paradigm where, at each iteration, a part of the current solution is partially destroyed by removing several customers from the routes and by creating a new solution through a reinsertion procedure. A simulated annealing criterion is used to determine whether the new solution should be accepted as the current solution. In the computational results, the parameter settings suggested in [**15**] were used.

(d) The *Hybrid Genetic Search* (HGS) [**85**] is the current state-of-the-art metaheuristic for the CVRP. It is a hybrid genetic algorithm that combines crossover operations for solution generation with a neighborhood search for solution improvement. Additionally, the population is maintained to ensure a diversified search, and infeasible solutions are allowed with adaptive penalties. For this algorithm, we rely

on parameter settings suggested in [**85**].

Regarding the stopping criterion, it should be noted that CW naturally stops when it is impossible to merge two routes without violating the capacity constraints. For GORT, SISR, and HGS, the stopping criterion is set to a time limit that depends linearly on the instance size, as in [**85**]. More precisely, we use:

$$T(n) = 2.4 \cdot n \text{ seconds} \tag{3.1}$$

per instance, where n is the number of customers. With this, the time limit ranges from four minutes on an instance with 100 customers to 40 minutes for 1000 customers.

As previously mentioned, we also include experiments with an exact branch-cut-and-price algorithm [**67**]. However, since this algorithm is limited in the size of problems that it can solve, our experiments with this approach are applied only to a subset of smaller instances. This is covered in a dedicated section (Section 4.2).

3.2. Test Instances

We conducted our experiments on the 100 classical synthetic CVRP instances from Uchoa et al. [**82**] with Euclidean distances, as well as 12 real-world instances provided by the companies Loggi and ORTEC for the 2021 DIMACS implementation challenge.

The six instances from Loggi contain 400 to 1,000 customers and come from urban delivery services in some of the largest cities in Brazil. The distances provided in these instances were calculated from the urban networks. The six instances from ORTEC contain between 241 and 700 customers and were derived from the activities of a US-based grocery delivery service. In this latter case, the distances correspond to real driving times.

The Euclidean instances from Uchoa et al. [**82**] (usually called "Set X") have a size ranging from 100 to 1,000 customers. These instances were generated to reflect a variety of possible factors and situations. Their generation has been driven by six main factors:

- n: number of customers (i.e., instance size)
- *Dep*: location of the depot, either C (center of the grid), E (corner of the grid), or R (random);
- *Cust*: location of the customers, either R (random), C (clustered), or RC(k) (mixed) with $k \in \{3, \ldots, 8\}$ clusters
- *Dem*: customer demand, which can be:

  U: all demands equal to 1

1-10: demand from UD[1-10] (where UD[a,b] denotes a uniform discrete distribution over interval [a, b])
5-10: demand from UD[5-10]
1-100: demand from UD[1-100]
50-100: demand from UD[50-100]
Q: demand from UD[1-50] or UD[51-100] depending if the customer is located in an even or odd quadrant of the grid, respectively
SL: many Small values, few Large values, that is, 70% to 95% of demands are from UD[1-10], while the remaining demands are from UD[50-100]

— r: approximation of the average number of customers per route (see [**82**]).

3.3. Distance Inaccuracies

It is common to observe data inaccuracies due, for example, to the use of geolocalization devices to collect data. To account for these inaccuracies, we introduce perturbations to the distances (or travel times) between each pair of nodes. For a given distance d_{ij}, a value ϵ_{ij} is generated by truncating a normal distribution of mean 1 and standard deviation σ. Thus, the distribution is defined within the interval $[1-\sigma,1+\sigma]$. With this, a "noisy" distance d^*_{ij} is obtained by multiplying d_{ij} by ϵ_{ij}, that is:

$$d^*_{ij} = \epsilon_{ij} \cdot d_{ij}. \tag{3.2}$$

Larger standard deviations σ lead to larger perturbations to the original distances. To analyze the impact of noise, six levels were considered in our computational study: $\sigma = 0.00$ (no noise), and then $\sigma = 0.01, 0.02, 0.05, 0.10$, and 0.15.

Let us denote the 100 instances of Set X as P_1 to P_{100}, where P_1 is the smallest instance and P_{100} is the largest one. For each original instance P_i with $i \in \{1,\ldots,100\}$ and noise level $\sigma \in \{0.01,0.02,0.05,0.10,0.15\}$, we generated ten noisy instances $j \in \{1,\ldots,10\}$. In the following, each noisy instance is denoted $P^\sigma_{i,j}$. This leads to $100 \times 5 \times 10 = 5000$ noisy instances. Finally, in the case of $\sigma = 0.00$ (no noise), ten runs of SISR and HGS were performed on each instance with different random seeds to obtain averages, as in the other cases. For GORT, instead of random seeds, different orders of customers were used because this algorithm is sensitive to the order of the data.

Let s^σ_{ij} be the solution obtained with a given method on a noisy instance $P^\sigma_{i,j}$. We can evaluate the quality of this solution using the true distances (ground truth) in the original instance P_i to obtain the true total distance (cost) traveled by the vehicles, denoted as $c(s^\sigma_{ij})$. Then, considering s_i the best-known solution of the original instance P_i and the

corresponding cost $c(s_i)$ (see the Appendix), the percentage gap between the true cost of solution s_{ij}^{σ} and the cost of the best-known solution s_i can be computed as:

$$Gap_{ij}^{\sigma} = 100 \left(\frac{c(s_{ij}^{\sigma}) - c(s_i)}{c(s_i)} \right). \tag{3.3}$$

For an original instance P_i and a given noise level σ, the average percentage gap over the ten corresponding noisy instances is calculated as:

$$Avg_i^{\sigma} = \frac{1}{10} \left(\sum_{j=1}^{10} Gap_{ij}^{\sigma} \right). \tag{3.4}$$

Finally, for a given noise level σ, the global average percentage gap over the 100 instances is calculated as:

$$Avg^{\sigma} = \frac{1}{1000} \left(\sum_{i=1}^{100} \sum_{j=1}^{10} Gap_{ij}^{\sigma} \right). \tag{3.5}$$

The noisy instances based on the 12 real-world instances of Loggi and ORTEC were generated in the same way, leading to $12 \times 5 \times 10 = 600$ noisy instances overall.

4. Experimental Study

Our experiments are divided into three parts. First, Section 4.1 examines the impact of noise on the average gap of the four heuristics presented before, using the set of 5,000 noisy test instances derived from Set X. Next, Section 4.2 conducts the same experiment for the exact method, using a subset of small instances from Set X, and compares the results obtained with our heuristic methods. Finally, Section 4.3 examines the impact of noise on real-world instances using a set of 600 noisy instances derived from Loggi's and ORTEC's applications. All algorithms have been run on an Intel Gold 6148 Skylake 2.4 GHz processor with 40 GB of RAM under CentOS 7.8.2003 (one thread).

4.1. Performance of the heuristics on set X instances with inaccurate distances

We start our analysis by examining the average gaps Avg_i^{σ} of each heuristic on the ten noisy instances associated with each original instance P_i, $i = 1, ..., 100$, and each noise level σ, as defined in Equation (3.4). These results are reported in Figure 15.

Clearly, the simplest constructive heuristic (CW) has the worst performance. The solution quality achieved by GORT is better than that of CW but still far below the solution quality of the two other metaheuristics SISR and HGS. The performance of GORT seems to degrade as the instance size increases, as indicated by the upward trend from left to right in the figure. SISR and HGS are the best heuristics and produce similar results. Also, the

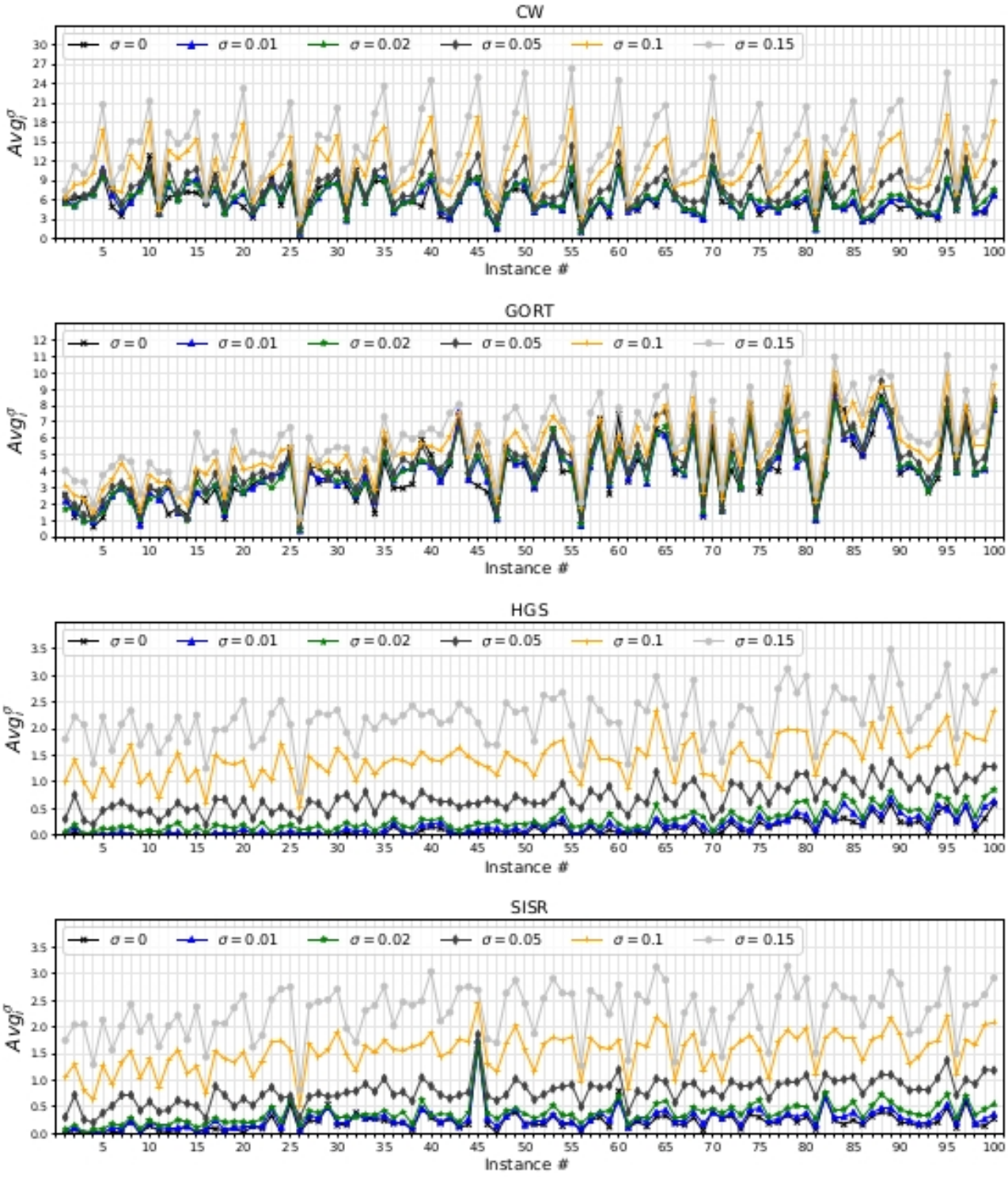

Figure 15. Average gaps obtained on each original instance for each noise level

average gaps increase with the level of noise σ, which was expected, given that the noisy

instances differ more significantly from the original instances when there is more noise.

The figure also shows regular patterns that are a consequence of the design of the Set-X instances in [**82**]. More precisely, the benchmark is organized into groups of five consecutive instances, such that the average number of customers per route in a solution, as represented by attribute r, increases from the first instance in a group to the fifth instance. Figure 15 thus indicates that the average gap with the best-known solution increases with the average number of customers per route in a solution, with a peak for instances P_5, P_{10}, P_{15}, etc. This behavior is more apparent for CW. We made an exhautive examination to determine possible relations between the gaps shown in Figure 15 and the other Set-X instances attributes (n, Dep, $Cust$ and Dem), however, we did not find a strong relation.

An insightful view on the impact of noise is obtained when we consider Avg^{σ}, as defined in Equation (3.5), which is the global average gap with the best-known solution over the 1,000 noisy instances, generated from the 100 original instances, for any given noise level. This is shown in Figure 16, where each dot in the figure corresponds to the global average gap Avg^{σ} for a given heuristic and noise level σ. In this figure, we can observe that, on average, the noise level has a greater impact on CW, compared with the three other heuristics. To distinguish between HGS and SISR, a different scale is used for the graph on the right. This graph indicates that HGS is slightly more robust to noise than SISR.

The poor performance of CW is not much of a surprise. This is a pure construction heuristic whose behavior is totally dependent on the ordering of the savings (from largest to smallest) to merge routes. With inaccurate distances, the ordering of the savings is very likely to change, thus leading to bad decisions. And, as opposed to the three competing metaheuristics, this construction heuristic has no way to recover later when bad decisions are taken.

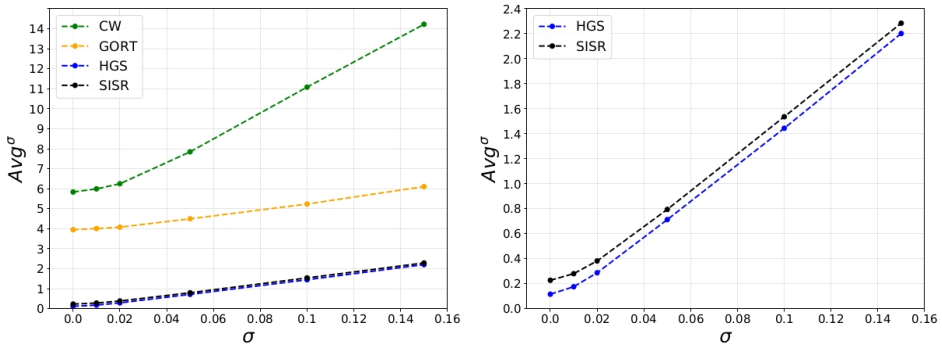

Figure 16. Global average gaps with increasing values of σ for each heuristic.

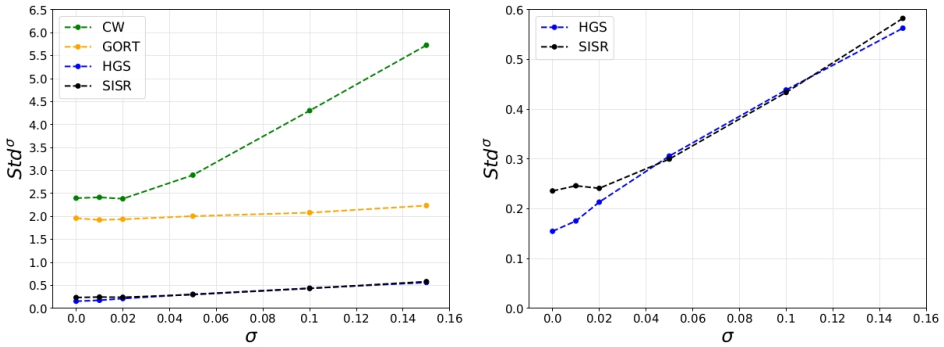

Figure 17. Standard deviation of global average gaps (large dots) with increasing values of σ for each heuristic.

While Figure 16 shows the global average gaps Avg^σ for each heuristic and noise level σ, Figure 17 shows the standard deviations Std^σ of the corresponding global gaps which, obviously, increase with the noise level σ. As in Figure 16, another scale is used in the graph shown on the right to distinguish between SISR and HGS. We can see that the standard deviations of SISR are substantially larger than HGS on the original instances (no noise). It means that the solutions produced via multiple executions of SISR (with different seeds) on the same instance vary more widely from one execution to the next. This difference gradually decreases as the level of noise increases and both SISR and HGS meet at $\sigma = 0.05$, where the noise due to distance inaccuracies start to dominate.

Table 10 shows the explicit global average gap values Avg^σ for each heuristic and noise level, which correspond to the dots in Figure 16, as well as the corresponding standard deviation values Std^σ, which correspond to the dots in Figure 17.

	CW		**GORT**		**HGS**		**SISR**	
	Avg^σ	Std^σ	Avg^σ	Std^σ	Avg^σ	Std^σ	Avg^σ	Std^σ
$\sigma = 0.00$	5.83	2.39	3.94	1.96	0.11	0.15	0.22	0.23
$\sigma = 0.01$	5.99	2.41	4.00	1.92	0.17	0.17	0.27	0.24
$\sigma = 0.02$	6.24	2.38	4.07	1.93	0.28	0.21	0.38	0.24
$\sigma = 0.05$	7.84	2.90	4.49	2.00	0.71	0.30	0.79	0.29
$\sigma = 0.10$	11.08	4.30	5.23	2.08	1.44	0.43	1.53	0.43
$\sigma = 0.15$	14.22	5.72	6.10	2.23	2.20	0.56	2.28	0.58

Table 10. Global average gaps and standard deviations for each method and each noise level

It is worth noting that a Wilcoxon signed-rank test, applied to every pair of methods, showed a statistically significant difference in solution quality in every case with a level of confidence of 95%.

4.2. Performance of an exact algorithm on set X with inaccurate distances

In this section, we additionally consider the results of an exact branch-cut-and-price (BCP) algorithm introduced in [**67**]. We analyze its sensitivity to noise compared to that of the heuristics (except CW which was largely outperformed by the other methods). For this analysis, we focus on the 13 instances for which it was possible to obtain optimal solutions within 12 hours of computation time for all seeds and noise levels (instances with index i = 1, 2, 3, 4, 6, 7, 12, 13, 16, 17, 18, 21, and 22). This subset of instances is denoted $\hat{P}$ in the following. We also consider two additional noise levels (σ = 0.20, 0.25) to have a better sense of the behavior of SISR and HGS in the presence of very large noise.

Figure 18 shows the global average gaps with the optimum over the 13 × 10 = 130 instances for each level of noise σ and each method, including BCP. The methods SISR, HGS and BCP are so close on these relatively small instances, with at most 200 customers, that they are difficult to distinguish even by looking at the graph on the right, which uses a different scale. Accordingly, Table 11 reports explicit global average gap values, which correspond to the dots in Figure 18. In addition, the standard deviations of the corresponding global gaps are shown. Obviously, the gap of BCP is null for σ = 0.00 (no noise) since the original instances with true distances are solved optimally.

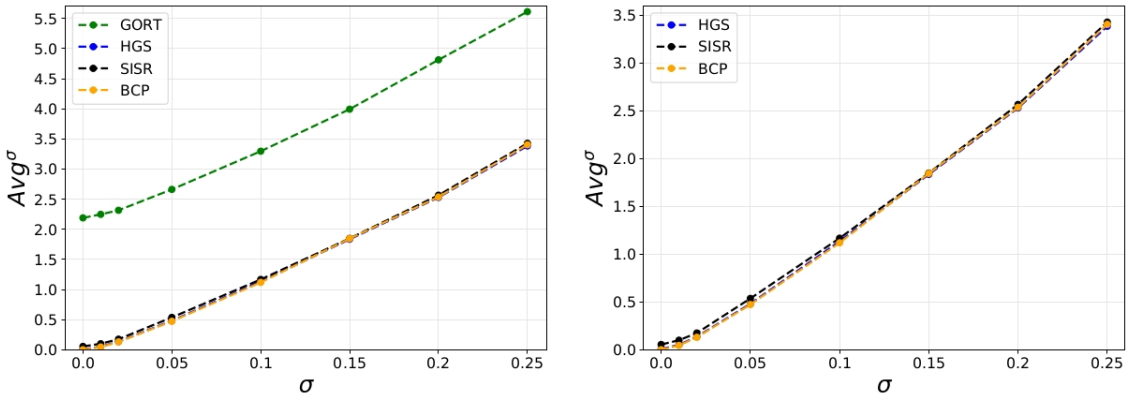

Figure 18. Global average gaps with the optimum for each method with increasing values of σ. On the left, GORT, HGS and SISR are compared with BCP; on the right, using a different scale, only HGS and SISR are compared with BCP.

We observe that the difference between SISR and HGS stays approximately the same with increasing values of σ. Considering Table 11, the difference between the global average gap values Avg^σ of SISR and HGS remains in the range of 0.03% to 0.05%, except for σ = 0.15 where it is equal to 1.84% - 1.83% = 0.01%. While the difference between the global average gaps of SISR and HGS stays about the same, both methods get closer to BCP with

	BCP		GORT		HGS		SISR	
	Avg^σ	Std^σ	Avg^σ	Std^σ	Avg^σ	Std^σ	Avg^σ	Std^σ
$\sigma = 0.00$	0.00	0.00	2.18	0.87	0.01	0.01	0.05	0.06
$\sigma = 0.01$	0.04	0.03	2.24	1.00	0.05	0.04	0.09	0.11
$\sigma = 0.02$	0.13	0.08	2.31	1.04	0.13	0.09	0.17	0.11
$\sigma = 0.05$	0.47	0.22	2.66	1.08	0.48	0.23	0.53	0.26
$\sigma = 0.10$	1.11	0.38	3.29	1.15	1.13	0.40	1.16	0.41
$\sigma = 0.15$	1.84	0.48	3.99	1.24	1.83	0.47	1.84	0.46
$\sigma = 0.20$	2.53	0.67	4.81	1.21	2.52	0.70	2.56	0.68
$\sigma = 0.25$	3.40	0.88	5.61	1.29	3.38	0.90	3.42	0.91

Table 11. Global average gaps and corresponding standard deviations for each method and each noise level

increasing values of σ, to the point where HGS becomes in fact better than BCP on average for $\sigma = 0.15, 0.20, 0.25$.

To evaluate the statistical significance of these results, we performed multiple pairwise Wilcoxon signed-rank tests, where the null hypothesis states that solution quality is the same for the two methods involved in the test. The p-values obtained are shown in Table 12. Basically, when a p-value is less than or equal to 0.05, the null hypothesis is rejected and a statistically significant difference is observed with a 95% confidence level. Although SISR, HGS, and BCP are very close, statistically significant differences are observed with a low level of noise (see the gray cells, where HGS outperforms SISR and BCP outperforms both SISR and HGS). This is not the case for a high level of noise. Even if the difference between the global average gaps Avg^σ of HGS and SISR stays about the same over the different σ values, no statistically significant difference is observed between the two methods for σ values greater than or equal to 0.10, according to the Wilcoxon test. This is the same when SISR and HGS are compared with BCP. Thus, when two methods are very close with regard to solution quality, the superiority of one method over the other gets blurred when a sufficiently high level of noise is present (i.e., $\sigma = 0.10$ or more in our experiments).

4.3. Performance of heuristics on real-world instances with inaccurate distances

Here, we examine the performance of our four heuristics over real data. For this purpose, we tested these heuristics on the real-world instances of Loggi and ORTEC. The best-known solutions for these instances can be found in the Appendix. Noise was introduced into the distance data in the same way as in the instances of Set X. Thus, the analysis is similar to that of Section 4.1.

	SISR vs HGS	SISR vs BCP	HGS vs BCP
$\sigma = 0.00$	6.03×10^{-16}	7.76×10^{-16}	7.41×10^{-5}
$\sigma = 0.01$	2.48×10^{-8}	4.21×10^{-11}	0.03
$\sigma = 0.02$	1.60×10^{-5}	2.76×10^{-7}	0.02
$\sigma = 0.05$	1.90×10^{-5}	3.03×10^{-6}	0.21
$\sigma = 0.10$	0.05	0.01	0.24
$\sigma = 0.15$	0.60	0.96	0.23
$\sigma = 0.20$	0.33	0.37	0.69
$\sigma = 0.25$	0.13	0.27	0.35

Table 12. p-values of Wilcoxon signed-rank test

Figure 19 is similar to Figure 16 and shows the global average gaps Avg^σ for each heuristic and noise level σ over the 120 noisy instances, generated from the 12 real-world instances. As in the previous experiments, CW is much more affected by noise than the other heuristics. Also, GORT remains far from SISR and HGS, whatever the level of noise. SISR and HGS are again quite close and by changing the scale (see the graph on the right), we observe that SISR is more affected by noise than HGS. Table 13 explicitly reports the global average gaps Avg^σ, in addition to the corresponding standard deviations Std^σ, for each heuristic and each noise level. We observe that the difference between the global average gaps of the two methods increases from 0.29% with $\sigma = 0.01$ to 0.56% with $\sigma = 0.15$ (no such increase was observed on Set-X instances, see Table 10). In fact, HGS outperforms SISR by a wider margin on the original (no noise) real-world instances when compared to the original Set-X instances. The differences between the global average gaps of HGS and SISR are equal to 0.28% and 0.11%, respectively.

As in Section 4.1, a Wilcoxon signed-rank test, applied to every pair of methods, showed a statistically significant difference in solution quality in every case with a level of confidence of 95%.

	CW		GORT		HGS		SISR	
	Avg^σ	Std^σ	Avg^σ	Std^σ	Avg^σ	Std^σ	Avg^σ	Std^σ
$\sigma = 0.00$	6.10	1.65	4.25	1.71	0.14	0.17	0.43	0.34
$\sigma = 0.01$	6.31	1.89	4.86	1.92	0.19	0.18	0.47	0.31
$\sigma = 0.02$	6.93	1.77	4.91	1.93	0.30	0.22	0.59	0.37
$\sigma = 0.05$	8.93	1.94	5.50	2.17	0.70	0.27	1.04	0.39
$\sigma = 0.10$	12.75	3.04	6.55	2.30	1.51	0.36	1.95	0.59
$\sigma = 0.15$	16.72	3.71	7.65	2.16	2.43	0.47	2.99	0.65

Table 13. Global average gaps and corresponding standard deviations on real instances for each heuristic and noise level

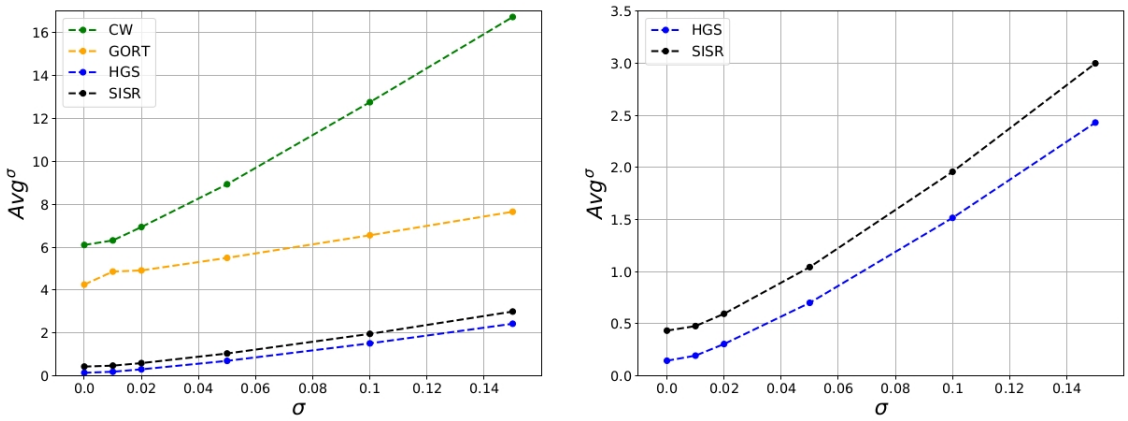

Figure 19. Global average gaps obtained on real instances with increasing values of σ for each heuristic.

5. Conclusions

This work has studied the performance of four heuristics and one exact method for the CVRP over two sets of instances using different levels of noise. The study first shows that not all methods are similarly impacted by noisy distances. In particular, the CW heuristic is much more sensitive to noise than the three other metaheuristics. Second, our results indicate that better methods generally lead to better solutions, even in the presence of noise, which is true for both the synthetic and real-world instances in our test set. Accordingly, sophisticated state-of-the-art methods are still preferable when data are inaccurate. We also observed that when two methods exhibit a sufficiently small, although statistically significant, difference in performance on true data, this statistically significant difference vanishes with high levels of noise. This situation occurred when we compared the two state-of-the-art metaheuristics SISR and HGS with an exact method on a subset of small instances from Set X with at most 200 customers. For future work, it would be interesting to conduct similar studies for other, more complex, variants of the CVRP, for example the VRP with time windows for which many standard benchmark instances are available in the literature.

Appendix

Table 14 shows the number of customers n and best known solution values $c(s_i)$, i = 1, ..., 100, for the Set-X instances from [**82**]. Table 15 shows the number of customers n and best-known solution values $c(s_i)$, $i = 1, ..., 12$, for the real-world instances provided by Loggi and ORTEC for the 2021 DIMACS implementation challenge.

Instance	i	n	$c(s_i)$	**Instance**	i	n	$c(s_i)$	**Instance**	i	n	$c(s_i)$
X-n101-k25	1	101	27591	X-n261-k13	35	261	26558	X-n502-k39	69	502	69226
X-n106-k14	2	106	26362	X-n266-k58	36	266	75478	X-n513-k21	70	513	24201
X-n110-k13	3	110	14971	X-n270-k35	37	270	35291	X-n524-k137	71	524	154593
X-n115-k10	4	115	12747	X-n275-k28	38	275	21245	X-n536-k96	72	536	94868
X-n120-k6	5	120	13332	X-n280-k17	39	280	33503	X-n548-k50	73	548	86700
X-n125-k30	6	125	55539	X-n284-k15	40	284	20215	X-n561-k42	74	561	42717
X-n129-k18	7	129	28940	X-n289-k60	41	289	95151	X-n573-k30	75	573	50673
X-n134-k13	8	134	10916	X-n294-k50	42	294	47161	X-n586-k159	76	586	190316
X-n139-k10	9	139	13590	X-n298-k31	43	298	34231	X-n599-k92	77	599	108451
X-n143-k7	10	143	15700	X-n303-k21	44	303	21736	X-n613-k62	78	613	59535
X-n148-k46	11	148	43448	X-n308-k13	45	308	25859	X-n627-k43	79	627	62164
X-n153-k22	12	153	21220	X-n313-k71	46	313	94043	X-n641-k35	80	641	63694
X-n157-k13	13	157	16876	X-n317-k53	47	317	78355	X-n655-k131	81	655	106780
X-n162-k11	14	162	14138	X-n322-k28	48	322	29834	X-n670-k126	82	670	146332
X-n167-k10	15	167	20557	X-n327-k20	49	327	27532	X-n685-k75	83	685	68205
X-n172-k51	16	172	45607	X-n331-k15	50	331	31102	X-n701-k44	84	701	81923
X-n176-k26	17	176	47812	X-n336-k84	51	336	139111	X-n716-k35	85	716	43387
X-n181-k23	18	181	25569	X-n344-k43	52	344	42050	X-n733-k159	86	733	136190
X-n186-k15	19	186	24145	X-n351-k40	53	351	25896	X-n749-k98	87	749	77314
X-n190-k8	20	190	16980	X-n359-k29	54	359	51505	X-n766-k71	88	766	114454
X-n195-k51	21	195	44225	X-n367-k17	55	367	22814	X-n783-k48	89	783	72394
X-n200-k36	22	200	58578	X-n376-k94	56	376	147713	X-n801-k40	90	801	73305
X-n204-k19	23	204	19565	X-n384-k52	57	384	65940	X-n819-k171	91	819	158121
X-n209-k16	24	209	30656	X-n393-k38	58	393	38260	X-n837-k142	92	837	193737
X-n214-k11	25	214	10856	X-n401-k29	59	401	66163	X-n856-k95	93	856	88965
X-n219-k73	26	219	117595	X-n411-k19	60	411	19712	X-n876-k59	94	876	99299
X-n223-k34	27	223	40437	X-n420-k130	61	420	107798	X-n895-k37	95	895	53860
X-n228-k23	28	228	25742	X-n429-k61	62	429	65449	X-n916-k207	96	916	329179
X-n233-k16	29	233	19230	X-n439-k37	63	439	36391	X-n936-k151	97	936	132725
X-n237-k14	30	237	27042	X-n449-k29	64	449	55233	X-n957-k87	98	957	85465
X-n242-k48	31	242	82751	X-n459-k26	65	459	24139	X-n979-k58	99	979	118987
X-n247-k47	32	247	37274	X-n469-k138	66	469	221824	X-n1001-k43	100	1001	72359
X-n251-k28	33	251	38684	X-n480-k70	67	480	89449				
X-n256-k16	34	256	18839	X-n491-k59	68	491	66487				

Table 14. Best-known solutions on Set-X instances

Loggi				ORTEC			
Instance	i	n	$c(s_i)$	**Instance**	i	n	$c(s_i)$
Loggi-n401-k23	1	400	336903	ORTEC-n242-k12	7	241	123750
Loggi-n501-k24	2	500	177176	ORTEC-n323-k21	8	322	214071
Loggi-n601-k19	3	600	113155	ORTEC-n405-k18	9	404	200986
Loggi-n601-k42	4	600	347059	ORTEC-n455-k41	10	454	292485
Loggi-n901-k42	5	900	246301	ORTEC-n510-k23	11	509	184529
Loggi-n1001-k31	6	1000	284356	ORTEC-n701-k64	12	700	445543

Table 15. Best-known solutions on real-world instances

Third Article.

A Ruin&Recreate Ant Colony System Algorithm for the Capacitated Vehicle Routing Problem

by

Fernando Obed Guillen Reyes[1], Jean-Yves Potvin[2], Michel Gendreau[3], and Thibaut Vidal[4]

([1]) Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

([2]) Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

([3]) Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

([4]) Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, Canada.

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT).

SCALE-AI Chair in Data-Driven Supply Chains.

The jounal to submit this article will be determined.

The main contributions of Fernando Obed Guillen Reyes for this articles are presented. He designed, with the help of his advisors, the hybrid algorithm based on the integration of a reinforcement learning mechanism into a state-of-the-art ruin-and-recreate metaheuristic for the capacitated vehicle routing problem. Then, he implemented the algorithm and performed the computational experiments. He also wrote the first draft of the paper.

RÉSUMÉ. Dans cet article, nous présentons un algorithme hybride qui combine une approche "détruire et reconstruire" avec une méthode d'apprentissage par renforcement afin de résoudre le problème classique de tournées de véhicules avec contraintes de capacité. Dans un tel contexte, "détruire et reconstruire" consiste à retirer des clients dans les routes de la solution courante ("détruire") et de réinsérer ces clients d'une façon différente ("reconstruire") dans l'espoir d'obtenir une solution de meilleure qualité. À cette fin, nous exploitons la métaheuristique appelée Slack Induction for String Removals (*SISR*), proposée par Christiaens and Van den Berghe, qui s'est montrée compétitive avec l'état de l'art sur des instances tests classiques pour le problème de tournées de véhicules avec contraintes de capacité. Nous enrichissons cet algorithme à l'aide d'un apprentissage par renforcement qui s'inspire des systèmes de colonies de fourmis, où l'apprentissage provient de l'addition de traces de phéromone. Plus précisément, une quantité de phéromone est ajoutée sur les arêtes prometteuses, soit celles que l'on retrouve fréquemment dans les bonnes solutions rencontrées au cours de la recherche, ceci afin de guider l'algorithme vers des solutions de meilleure qualité. L'addition d'un mécanisme d'apprentissage à l'algorithme SISR original montre que de meilleures solutions peuvent être obtenues sur les instances tests classiques mentionnées précédemment, même si les solutions originales étaient déjà de très grande qualité.
Mots clés : Détruire et reconstruire, apprentissage par renforcement, systèmes de colonies de fourmis, problème de tournées de véhicules avec contraintes de capacité

ABSTRACT. In this paper, we present a hybrid algorithm that combines a ruin-and-recreate heuristic search strategy with a reinforcement learning component to solve the classical capacitated vehicle routing problem. In a vehicle routing context, ruin-and-recreate consists in removing a number of customers from the current vehicle routes (ruin) and reinserting those customers into the routes (recreate) in another way to obtain a different, hopefully better, solution. For this purpose, we exploit the recent SISR metaheuristic proposed by Christiaens and Van den Berghe that proved to be highly competitive on standard benchmark instances for the capacitated vehicle routing problem. We enrich this algorithm with reinforcement learning ideas taken from ant colony systems, where learning takes place through the addition of pheromone traces. Basically, pheromone is added to promising edges (i.e., those that are frequently found in good solutions encountered during the search) to guide the algorithm towards even better solutions. Through the addition of this learning component to the original SISR algorithm, we show that better solutions can be obtained, even if the original solutions were already of very high quality.
Keywords: Ruin and recreate, reinforcement learning, ant colony systems, capacitated vehicle routing problem

1. Introduction

Vehicle routing problems (VRPs) have been a topic of interest for researchers for many years due to their combinatorial complexity and their multiple applications. Although there are many variants of these problems, exact algorithms can only solve relatively small instances of the simplest variants, like the capacitated VRP (CVRP) or the VRP with time windows (VRPTW). Accordingly, researchers have developed sophisticated heuristic search methods, called metaheuristics, to generate near-optimal solutions for difficult problems in relatively small amount of computation times. Along with metaheuristics, machine learning has also become very popular among researchers by providing significant contributions in many different areas. Thus, our goal here is to integrate a learning component into a state-of-the-art metaheuristic for the CVRP to produce an even better algorithm.

The CVRP is NP-Hard and is defined on a graph $G = (V,E)$, where $V = \{0, 1, 2,...., n_c\}$ is the set of vertices and $E = \{(i,j) : i, j \in V\}$ is the set of edges. Vertex 0 stands for the depot where a homogeneous fleet of vehicles is located and where the route of each vehicle starts and ends. The other vertices are customers, where each customer $i \in V \setminus \{0\}$ has a demand d_i that must be served by a single vehicle. It is important to note that the total customer demand served by a vehicle cannot exceed its capacity Q. The goal is to construct a set of minimum cost routes, one for each vehicle, that serve all customers while satisfying the capacity constraints. The cost to be minimized corresponds to the total distance traveled by all vehicles.

The basic metaheuristic that we use for solving the CVRP is called SISR (for Slack Induction by String Removal) and is reported by Christiaens and Vanden Berghe in [**14**]. This metaheuristic employs a particular ruin-and-recreate strategy to produce highly competitive results on well-known CVRP benchmark instances. We implemented this algorithm based on its description in the original paper and we were able to produce solutions that are, statistically, of the same quality. Then, we integrated into this algorithm some concepts borrowed from a variant of Ant Colony Optimization (ACO), called Ant Colony System (ACS), to improve the recreate phase of SISR by allowing the algorithm to better recognize edges that belong to good routes.

The rest of the paper is organized as follows. We first provide in Section 2 a brief literature review of different classes of machine learning methods, including ACO, that have been applied to vehicle routing problems. Then, Section 3 describes the SISR metaheuristic. Some fundamental concepts of ACS are then introduced in Section 4, followed by a complete description of our hybrid algorithm in Section 5. Computational results on well known benchmark instances for the CVRP are reported in Section 6. Finally, we conclude with some remarks and future research avenues in Section 7.

2. Literature review

In this section, we briefly review different classes of learning methods that have been exploited to address vehicle routing problems.

2.1. Neural Networks

Neural networks were among the first machine learning methods applied to VRPs, mostly in the 1980's and 1990's. For example, self-organizing maps and elastic neural networks were used to find solutions of VRPs with a geometric structure [**36, 37**]. Basically, chains of neurons are progressively stretched until they form the contours of routes. Other types of neural networks were also used to identify customers that should be served by the same vehicle to facilitate the construction of good routes by another algorithm. Multi-layer neural networks were also exploited in other ways, for example, as a prediction tool to decide about the best algorithm (among a set of possible algorithms) to solve a given VRP instance, based on its characteristics [**49, 81**]. Solutions produced by different metaheuristics on different types of instances are first collected in this case to form a training sample for the neural network. Overall, attempts at addressing VRPs with neural network approaches did not prove to be very successful and were limited to relatively small instances.

2.2. Clustering

Clustering can be seen as the grouping of elements in sets based on some similarity measure. Clustering is the most widely used learning-based approach for VRPs, since it allows to implement an intuitive divide-and-conquer approach. The most popular clustering algorithms for this purpose are k-means and density-based clustering (one advantage of the latter is that the number of clusters does not need to be a priori determined). Most works in this category have in common that the clustering algorithm is used as a first step to transform a VRP variant into a simpler variant, for example a multiple-depot VRP into a number of single-depot VRPs [**34**], or to partition the set of customers into subsets [**51**], thus producing a form of aggregation that reduces the problem size. In a second step, an optimization algorithm is then applied to each subproblem produced by the clustering method [**22, 39**].

2.3. Support Vector Machines and Learning Automata

The literature is scarce with regard to support vector machines (SVM) [**87**] and learning automata (LA) in the context of vehicle routing, although they have both been used to guide metaheuristics. For example, a LA is reported in [**79**] to determine which neighborhood of the current solution (among a set of possible neighborhoods) is the most promising for a local search-based metaheuristic. A probability of success, associated with each neighborhood, is updated by the learning component as the algorithm unfolds. A distributed LA, where a network of LAs cooperate on a given problem, has also been reported to address the CVRP [**1**]. Here, each customer is a learning automaton that determines the next customer that should be visited to produce a good route.

2.4. Reinforcement Learning

One difficulty when applying reinforcement learning (RL) to a combinatorial optimization problem like the VRP is to model the environment that provides feedback signals, because the number of states can quickly become prohibitive. However, a particular form of RL, called Q-learning, does not require such an explicit model, which may explain its wide use. In [**56**], the authors employ Q-learning to design an operator that creates a solution by iteratively choosing the next customer to be visited, based on the current customer and the current Q-values. If the new solution obtained at the end is better than the current one, then it becomes the new current solution. In [**73**], no VRP variant is specifically addressed, but the authors describe an algorithm where Q-learning is employed to model an agent capable of determining the optimal itinerary between two customer locations. The agent has access to different information from the environment, like traffic density. However, this framework seems promising for only small problem instances. Another interesting approach

is found in [**61**], where a RL-based component, integrated within a recurrent neural network framework, is proposed for the CVRP. In this case, the final solution is viewed as the product of a sequence of decisions and the RL component increases the probability that a desirable sequence of decisions is produced by the neural network.

2.5. Ant Colony Optimization

ACO is a metaheuristic that artificially reproduces the communication scheme of ants, which is based on pheromone traces laid on the ground. Ants are attracted by pheromone and, over time, it tends to accumulate on the shortest paths between food sources and the nest. Accordingly, ACO has been used to solve problems where shortest paths or shortest routes are looked for. The fundamentals of ACO, including the particular ACS variant, are described in [**26, 27, 28, 29**]. Basically, each ant constructs a route in an incremental fashion by deciding at each iteration about the next customer to be added to the current route. The choice of this next customer is based on a probabilistic rule where the probability of selecting a given customer depends on the distance between that customer and the last customer in the current route, as well as the amount of pheromone on the corresponding edge. While constructing a solution, each ant lays pheromone on the visited edges (local pheromone update rule). Also, when all ants have constructed their solution, an additional amount of pheromone is laid on the edges of the best solution found (global pheromone update rule). This procedure is then repeated until a stopping criterion is met.

While ant systems have initially been applied to the Traveling Salesman Problem (TSP) where a single (unconstrained) tour of minimum distance is constructed over all vertices in a graph, a number of extensions dealing with different types of vehicle routing problems have since been reported in the literature. Quite often, these ant systems are combined with local search-based improvement procedures to produce more competitive results. One of the first application for the CVRP is found in [**12**], where each ant constructs as many routes as necessary to visit all customers. That is, an ant returns to the depot to construct another route when no customer can be added to the current route without exceeding vehicle capacity. In [**69**], the authors depart from the traditional strategy, where a route is extended by adding a new customer at its end, and rather propose to use a savings-based heuristic where the pheromone on each edge is taken into account in the computation of the savings. In [**48**], pheromone is added to the edges of the best solutions found by a variable neighborhood search (VNS). This pheromone is then exploited by a perturbation operator within the VNS that constructs a new solution with an ant system. The authors applied this problem-solving approach to a VRP with simultaneous pickup and delivery.

An extension of ACS where multiple ant colonies are involved (MACS) is presented in [**32**]. Under this scheme, each colony optimizes its own objective. For example, in the case of the VRPTW, one colony minimizes the number of vehicles, while the second colony takes the solutions produced by the first colony and minimizes the total travel time. Both colony interacts through the pheromone update process. Similar approaches for the time-dependent VRP are reported in [**25, 32**]. In [**31**], a MACS is applied to the vehicle routing problem with backhauls. In this case, one colony assigns customers to vehicles, while the other colony constructs routes for customers assigned to the same vehicle.

2.6. Others

Some works reported in the literature do not necessarily apply classical machine learning methods but still, accumulate knowledge about structures found in good solutions and exploit this knowledge to guide the search process. In [**78**], an algorithm called BoneRoute is applied to the CVRP. Here, new solutions are constructed while using good components of previously constructed solutions. These components (bones) consist of sequences of consecutive customers in a route. The rationale is that bones that appear in a large number of good solutions should be combined to form even better solutions. A pool of solutions, made of the best visited solutions, is exploited in the process. That is, at each iteration, a finite number of bones are extracted from the pool based on two parameters: number of customers in a bone (bone-length) and number of routes in the pool that contain the bone (bone-frequency). Then, these bones are used by a variant of the savings heuristic to create a new solution. Tabu search is applied to the constructed solution for a final improvement before the solution is considered for inclusion in the pool (in which case, the routes associated with the solution of lowest quality in the pool are replaced by those of the new solution).

Another example of knowledge extraction is found in [**54**] for the VRPTW. Here, several independent processes (threads) cooperate to define and update a pool of solutions. These threads can perform different tasks, like constructing new solutions, applying an improvement metaheuristic or performing population-based metaheuristics. The knowledge extraction consists in identifying useful patterns in the pool of solutions. To this end, solutions in the pool are partitioned into three subsets: elite, average and worst, depending on their quality. The patterns extracted are subsets of edges along with their frequency. Two different kinds of patterns are looked for: promising and unpromising ones. If the pattern's frequency decreases from the elite subset to the worst subset, then the pattern is promising, but if the pattern's frequency increases, then it is considered unpromising. Components from unpromising patterns are prohibited during the search, while components from promising patterns are fixed in the solution, to intensify the search in good regions

of the solution space. Another interesting pattern identification procedure is described in [**86**], where a genetic algorithm is used to solve large-scale instances of the VRPTW. In particular, an education phase is applied to the offspring using two local search-based improvement procedures. One of these is called pattern improvement, where the best re-insertion combinations are identified.

3. Ruin&Recreate metaheuristic

The ruin-and-recreate (RR) problem-solving strategy was originally proposed in [**72**]. In this work, three ruin functions were proposed: the first one randomly selects the customers to be removed, the second one selects a random customer and then removes it along with a number of closest customers and the last one removes a string of customers (a sequence of consecutive customers in a route). Then, the recreate function repeatedly selects a removed customer and inserts it at the best feasible insertion place in the current partial solution until all removed customers are back into the solution. The algorithm proposed in [**70**] applies this idea within the Adaptive Large Neighborhood Search (ALNS) framework, where both removal and reinsertion heuristics are available. Each heuristic has a weight and, at each iteration, a removal heuristic and then a reinsertion heuristic are probabilistically selected based on their weight. There are two distinct probability distributions for the removal and reinsertion heuristics that are updated at each iteration depending on the performance of the selected heuristics. This is done so that heuristics with the best performance in each category are more likely to be selected. It is worth noting that a simulated annealing (SA) criterion is employed during the search to decide if the new solution obtained through the removal/reinsertion process should be kept or not. Basically, if the new solution is better than the current one, it is accepted and becomes the new current solution. Otherwise, there is still some probability to accept the new solution, but this probability decreases with the increase to the total distance of the current solution and a parameter, called the temperature.

A recent approach based on ruin-and-recreate is the Slack Induction by String Removal (SISR) metaheuristic reported in [**14**]. This is a state-of-the-art algorithm for solving the CVRP and is the basic algorithm used in our work. It is described in the following, based on the pseudo-code shown in Algorithm 8.

Initialization.

First, the number of iterations n is set to a multiple m of the number of customers n_c, with m equal to 300 000 in [**14**]. SISR starts with an initial solution s_0 made of individual routes that each contain a single customer. Thus, there are as many routes as

there are customers at the beginning. The current solution s and best known solution thus far s_{best} are also set to s_0. Then, the value of some parameters related to the simulated annealing acceptance criterion are defined, namely the initial temperature, final temperature and cooling factor. That is, the temperature is progressively reduced from its initial value to its final value according to an exponential cooling scheme where the temperature at iteration $iter + 1$ is obtained by multiplying the temperature at iteration $iter$ by a cooling factor k close to 1. This factor is such that it allows the temperature to decrease from its initial value to its final value within the desired number of iterations.

Main loop. At each iteration of the main loop, the current solution s is ruined and recreated to generate a new solution s_{new}, which can be accepted or not, depending on the simulated annealing criterion. That is, better solutions are always accepted and worse solutions are less and less likely to be accepted as the algorithm unfolds, due to the decreasing temperature value, to allow the algorithm to converge to a (hopefully good) local minimum. If solution s_{new} is accepted, then it replaces the current solution s. Furthermore, if solution s_{new} is better than s_{best}, then s_{best} is also set to s_{new}. Then, the temperature is decreased before the next iteration starts. The algorithm returns the best solution found at the end.

The originality of SISR comes from the two proposed ruin methods called *String* and *Split-String*, where strings of consecutive customers are removed from the routes. Although a full description of these ruin methods and their rationale can be found in [**14**], we will briefly describe the ruin algorithm of SISR based on the pseudo-code of Algorithm 9. It should first be noted that the number of strings to be removed n_r (which is equal to the number of routes that are ruined, since at most one string can be removed from a route), as well as the length of those strings, are carefully determined through appropriate parameter settings. At the beginning, the set of removed customers L and the set of ruined routes R^- are empty. These two sets are augmented each time a ruin method is applied. A first seed customer c_{seed} is randomly chosen among the whole set of customers, then at each iteration a customer i is selected from the adjacency list $adj(c_{seed})$, which contains all customers sorted in increasing order, with respect to the Euclidean distance to c_{seed}, and where the first customer is c_{seed}. Let i the current seed customer at a given iteration, and let r the tour that belongs to, if customer i has not been removed yet and its route r has not been ruined yet, then a ruin operator *RuinOp* is selected randomly and taking into account the characteristics of tour r, in order to ruin tour r. The two ruin operators are the following:

— *String.* , then a string of consecutive customers of cardinality l_r, that must include i, is randomly selected and removed from route r.

— *Split-String.* This ruin method is slightly different from *String.* More precisely, when a string is randomly selected in a route, a certain number of consecutive customers in this string are kept in the route. Consequently, these customers split the string into two substrings (one substring on each side of the customers that are kept in the route). Then, the two substrings are removed from the route.

A single recreate method is available in SISR to reinsert the removed customers into the solution. These removed customers are stored in L and are first sorted using a four possible strategies. A probability is associated with each strategy to select the one to be used at a given iteration. These sorting strategies are :

— Sort randomly (probability 4/11).

— Sort based on the customers' demands, with customers with larger demands first (probability 4/11).

— Sort based on the distance between customers and the depot, with the farthest customers first (probability 2/11).

— Sort based on the distance between customers and the depot, with the closest customers first (probability 1/11).

Once the order of reinsertion of the removed customers has been determined, a greedy insertion heuristic with blinks is applied, where all feasible insertion places of the current customer c in the partial solution are considered. The goal is to identify the best possible insertion place, that is, the one that leads to the smallest additional distance. Denoting $d(i,j)$ the distance or length of edge (i,j), the additional distance when customer c is inserted between consecutive customers i and j in a route corresponds to the detour $d(i,c) + d(c,j)$ - $d(i,j)$, since edges (i,c) and (c,j) are added to the solution, while edge (i,j) is removed. To introduce a form of diversification, a parameter β_0 defines a small probability for an insertion place to be overlooked, which is called a blink. Thus, a customer is inserted in its best feasible insertion place with probability $1-\beta_0$, its second best place with probability $(1-\beta_0)\beta_0$, etc. In general, a customer is inserted at its i-best place with probability $(1-\beta_0)\beta_0^{i-1}$. The value $\beta_0 = 0.01$ is suggested in [**14**], so that the probability associated with the best insertion place is 0.99, the second best insertion place 0.0099, etc. Once the current customer is inserted, the next customer in L is considered and the procedure is repeated until all customers in L are done. It should be noted that if there is no feasible insertion place for a customer, then a new route is created for that customer.

Algorithm 8 SISR metaheuristic

Initialization:
$n \leftarrow m \cdot n_c$
$s_0 \leftarrow$ *Create_initial_solution* (one individual route for each customer)
$s \leftarrow s_0$
$s_{best} \leftarrow s_0$
$t_0 \leftarrow$ initial temperature
$t_f \leftarrow$ final temperature
$k \leftarrow \left(\frac{t_f}{t_0}\right)^{1/n}$
$t \leftarrow t_0$
for $i = 1$ to n **do**
    $w \leftarrow$ random number in (0,1)
    $s_{new} \leftarrow Recreate(Ruin(s))$
    if $w < e^{-(dist(s_{new})-dist(s))/t}$ **then**
        $s \leftarrow s_{new}$
        if $dist(s_{new}) < dist(s_{best})$ **then**
            $s_{best} \leftarrow s_{new}$
        end if
    end if
    $t \leftarrow k \cdot t$
end for
Return s_{best}

Algorithm 9 SISR Ruin

Let R the set of routes of solution s

Initialization:
$n_r \leftarrow$ number of routes to be ruined
$L \leftarrow \emptyset$
$R^- \leftarrow \emptyset$
Select randomly a seed customer c_{seed}
Let $adj(c_{seed})$ the adjacency list associated to c_{seed}

for each $i \in adj(c_{seed})$ and $|R^-| < n_r$ **do**
    Let $r \in R$ be the route that contains customer i
    $l_r \leftarrow$ length of string to be removed
    if $r \notin R^-$ and $i \notin L$ **then**
        $RuinOp \leftarrow Random(String, Split-String)$
        $A \leftarrow Remove(i, r, l_r, RuinOp)$
        $L \leftarrow L \cup A$
        $R^- \leftarrow R^- \cup \{r\}$
    end if
end for
return s, L

4. Ant Colony System

In this section, we present some fundamentals about ACS, which is the variant of ACO used in this work. In general, these ant systems can be viewed as a form of reinforcement learning algorithm, where the pheromone added on the edges of a graph plays the role of reinforcement (or reward). At the beginning of ACS, an initial amount of pheromone is put on each edge. This value is typically the same for each edge and is denoted τ_0. These initial pheromone values are then progressively modified, through pheromone addition and evaporation. In the standard ACS framework, a fixed number of ants is used, where each ant constructs a solution by sequentially adding a new customer at the end of the current route. Selection of the next customer is based on the pheromone value as well as the length of the edges leading from the current customer (i.e., the last customer in the current route) to customers not yet included in the solution. The following transition rule is typically applied to select the next customer j, assuming that ant k is currently at customer i:

$$j = \begin{cases} argmax_{j' \in J_k}\{\tau(i,j')^{\alpha} h(i,j')^{\beta}\} & \text{if } w \leq q_0 \\ y & \text{otherwise} \end{cases} \tag{4.1}$$

where:

— J_k is the set of customers not yet visited by ant k.

— $\tau(i,j)$ is the pheromone value on edge (i,j)

— $h(i,j)$ is the heuristic component, often defined as the inverse of the length of edge (i,j) (i.e., inverse of the distance between customers i and j).

— α and β are the pheromone and heuristic exponents, respectively.

— w is a random number in [0,1] and $q_0 \in [0,1]$ is a threshold parameter.

— y is a random variable with the following probability distribution

$$p_j^k = \begin{cases} \frac{\tau(i,j)^{\alpha} h(i,j)^{\beta}}{\sum_{j' \in J_k} \tau(i,j')^{\alpha} h(i,j')^{\beta}} & \text{if } j \in J_k \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Thus, the selection of the next customer j is deterministic when $w < q_0$, otherwise j is selected according to the probability distribution defined in Equation (4.2), where p_j^k should be understood as the probability that ant k selects customer j as the next customer in the current route. Once all ants have constructed their solution, the pheromone values on the edges are updated. Thus, there is no local pheromone update rule, only a global one, which is shown in Equation (4.3).

$$\tau(i,j) = (1-\rho)\tau(i,j) + \rho\Delta \tag{4.3}$$

In this equation, $\rho \in (0,1)$ is the pheromone evaporation rate. If edge (i,j) is in the best solution s^*_{iter} produced by the ants at the current iteration, parameter Δ is set to the inverse of the total distance of this solution, otherwise Δ is set to 0. Thus, pheromone on edges that are not in s^*_{iter} decreases. In the next section, we explain how these ACS ideas were integrated into the SISR algorithm.

5. Hybrid RR-ACS algorithm

In our hybrid algorithm, we expect that the addition of pheromone on the edges will allow SISR to better identify the most promising edges during the recreate process. It should be noted that, in this case, a complete solution is recreated from a partial solution, not from scratch, as it is usually the case in ACS. A pseudo-code of our hybrid algorithm is shown in Algorithm 10. As we can see, it is strongly based on SISR, except that pheromone is exploited when a complete solution must be recreated after being ruined. The new features of our hybrid algorithm are described in the following.

Initialization. There is only one additional step in the initialization phase, which is to put an initial amount of pheromone τ_0 on every edge. In our implementation, τ_0 is set to 0.

Recreate. The main modification to the original SISR algorithm is the procedure to recreate a new complete solution after ruining the current solution, which is called *Recreate_ACS*. This procedure is described in Algorithm 11. The first step consists in sorting the removed customers stored in L as SISR does, that is, with the same four possible sorting strategies and the same probabilities associated with each strategy. Then, the customers are reinserted one by one into the solution based on this ordering. Let c be the customer to be inserted in the current partial solution. As in SISR, we consider all feasible insertion places for c, but we keep at the end the z best insertion places, that is, those leading to the smallest detours (after some experimentation, parameter z was set to 3). This is the role of function $Keep\_z\_Best\_Positions()$ in the pseudo-code. It should be noted that each one of those insertion places is identified by the corresponding edge (i, j) in the solution, which is added to set P. To define the transition rule where both the detour and the pheromone on the edges are taken into account, we need to calculate the heuristic and pheromone components of each insertion place in P. First, the heuristic component $h(i,c,j)$ shown in Equation (5.1) corresponds to the inverse of the detour, or additional distance, induced by inserting customer c between i and j. It should be noted that a value of 1 is added to the detour, in case the detour is 0. Then, the pheromone component $\tau(i,c,j)$ in Equation (5.2) corresponds to the maximum amount of pheromone taken over the two edges (i,c) and (c,j) that are added to the solution when c is inserted between i and j. Again, a value of 1 is added to

Algorithm 10 RR-ACS algorithm

Initialization:
$n \leftarrow m \cdot n_c$ (number of iterations)
$s_0 \leftarrow$ *Create_initial_solution* (one individual route for each customer)
$s \leftarrow s_0$
$s_{best} \leftarrow s_0$
$t_0 \leftarrow$ *initial temperature*
$t_f \leftarrow$ *final temperature*
$k \leftarrow \left(\frac{t_f}{t_0}\right)^{1/n}$ (cooling factor)
$t \leftarrow t_0$
for $(i,j) \in E$ **do**
    $\tau_{ij} \leftarrow \tau_0$
end for
for $i = 1$ to n **do**
    $w \leftarrow$ random number in interval (0,1)
    $s_{new} \leftarrow Recreate\_ACS(Ruin(s))$
    if $w < e^{-(dist(s_{new})-dist(s))/t}$ **then**
        $s \leftarrow s_{new}$
        if $dist(s_{new}) < dist(s_{best})$ **then**
            $s_{best} \leftarrow s_{new}$
        end if
        $\tau \leftarrow Update\_Accepted(\tau, s, \Delta, e_p)$
    else
        $\tau \leftarrow Update\_Rejected(\tau, e_p)$
    end if
    $t \leftarrow k \cdot t$
end for
Return s_{best}

guarantee a non null contribution of the pheromone component. To avoid recreating the same solution, the pheromone on the edges that were removed from the solution in the previous ruin process are temporarily reset to 0 (this set of edges is identified by S in the equation). In this way, the introduction of new edges in the solution is favored during the recreate process. This can be seen as a form of diversification.

$$h(i,c,j) = \left(\frac{1}{d(i,c) + d(c,j) - d(i,j) + 1}\right) \tag{5.1}$$

$$\tau(i,c,j) = (1 + max\{\tau'(i,c), \tau'(j,c)\}) \tag{5.2}$$

$$\tau'(i,j) = \begin{cases} \tau(i,j) & \text{if } (i,j) \notin S \\ 0 & \text{otherwise} \end{cases}$$

The final transition rule, as shown in Equation (5.3), is similar to the one used in ACS, where w is a random number in [0,1] and $q_0 \in [0,1]$ is the threshold parameter. This rule is aimed at selecting the final insertion place (i_{best}, j_{best}) for customer c among those stored in P. If w is less than or equal to the threshold, a deterministic choice is made, otherwise y is a random variable that returns an insertion place according to the probability distribution in Equation (5.4). When P is empty, that is, when there is no feasible insertion place for customer c, a new route is created for this customer.

$$(i_{best}, j_{best}) = \begin{cases} argmax_{(i,j)\in P}\{\tau(i,c,j)^{\alpha} h(i,c,j)^{\beta}\} & \text{if } w \leq q_0 \\ y & \text{otherwise} \end{cases} \tag{5.3}$$

$$p_{i,j} = \begin{cases} \frac{\tau(i,c,j)^{\alpha} h(i,c,j)^{\beta}}{\sum_{(i',j')\in P} \tau(i',c,j')^{\alpha} h(i',c,j')^{\beta}} & \text{if } (i,j) \in P \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Algorithm 11 Recreate_ACS(s,L)

```
1: Let s the ruined solution and L the set of removed customers
2: Initialization:
3: L ← sort(L)
4: while L ≠ ∅ do
5:     c ← first customer in L
6:     for all routes r ∈ R that are feasible for customer c do
7:         for all edges (i,j) ∈ R do
8:             detour(i,j) ← d(i,c) + d(c,j) − d(i,j)
9:             P ← Keep_z_Best_Positions(P,z,detour(i,j),(i,j))
10:        end for
11:    end for
12:    if P ≠ ∅ then
13:        (i_best, j_best) ← Transition_rule(P)
14:        Insert(c,(i_best, j_best))
15:    else
16:        r_0 ← create a route for customer c
17:        R ← R ∪ r_0
18:    end if
19:    L ← L \ {c}
20: end while
21: Return s
```

Pheromone update. In Algorithm 10, the pheromone values are updated each time a new solution s_{new} is produced through the ruin-and-recreate process. When s_{new} is accepted under the simulated annealing criterion, the update is performed through function *Update_Accepted* in the pseudo-code. The latter applies Equation (5.5) where some amount

of pheromone evaporates through its multiplication by (1 - ρ), with $\rho \in (0,1)$. Then, a fixed amount of pheromone Δ is added on the edges of s_{new}. This is to be opposed to the standard ACS scheme, where Δ is not constant and is multiplied by ρ. We decided to use a fixed Δ here for efficiency purposes, given that there is typically not much difference between the Δ values in the standard ACS framework (i.e., the inverse of the total distance of a solution is typically very small and there is very little difference between two different solutions). When s_{new} is rejected, the update is performed by the function *Update_Rejected*. The latter applies Equation (5.6) where only evaporation takes place on every edge.

$$\tau(i,j) = \begin{cases} (1-\rho)\tau(i,j) + \Delta & \text{if } (i,j) \in s_{new} \\ (1-\rho)\tau(i,j) & \text{otherwise} \end{cases} \tag{5.5}$$

$$\tau(i,j) = (1-\rho)\tau(i,j), \qquad \text{for all } (i,j) \tag{5.6}$$

6. Computational results

In this section, we report results obtained with our RR-ACS algorithm on the standard euclidean CVRP benchmark instances of Uchoa et al. [**82**]. This set is made of 100 instances made of small, medium and large instances ranging from 100 to 1000 customers. More precisely, small instances have between 100 and 250 customers, medium instances between 250 and 500 customers and large instances between 500 and 1000 customers. It should also be noted that the Euclidean distances are integer and obtained by rounding the real distances. In the following, we forget about the small instances because they are easy to solve and keep only the 36 medium instances and 32 large instances, for a total of 68 instances.

In this section, we first describe the search for the best parameter settings for our algorithm. Then, we compare the results reported for SISR in [**14**] with those obtained with our own implementation of this algorithm, called RR. Finally, we report the results produced by our hybrid RR-ACS algorithm. All tests were run on a 2.4 GHz Xeon Gold 6148 processor with 10G of memory.

6.1. RR Results

The first step was to compare RR with the original implementation of SISR. To this end, we ran RR 50 times on each one of the 68 test instances. The average solution values are show in Appendix A, along with those reported in [**14**] for SISR, also over 50 runs. Note that the average solution produced by RR is better when the corresponding entry is shaded. Then, we computed on each instance the gap between the two average solution values RR_{avg} and SISR_{avg}, using the formula :

$$\text{Gap} = 100 \times \frac{(\text{RR}_{\text{avg}} - \text{SISR}_{\text{avg}})}{\text{SISR}_{\text{avg}}} \tag{6.1}$$

The mean of these gaps over all benchmark instances was -0.0031. Although RR was slightly better on average, a p-value of 0.35 was obtained when we applied the signed-rank Wilcoxon test. This value is well above the standard significance level of 0.05. Thus, we cannot reject the null hypothesis, which states that the solutions produced by the two implementations are of the same quality. With regard to computation times, the ratio of RR over SISR (using the numbers in [**14**] based on a 2.6 GHz Xeon E5-2650 v2 processor) is 0.445, which means a reduction of about 55.4%, on average. But this observation must be mitigated because our processor and computer environment are not the same than those in [**14**]. According to the single thread rating found in www.cpubenchmark.net, the processor in [**14**] is about 23% slower than ours.

6.2. RR-ACS Parameter Settings

Before testing our hybrid RR-ACS algorithm, we had to calibrate the parameters α, β, ρ, Δ and q_0 that are introduced by the new ACS component. For this purpose, we randomly chose half of the medium size instances from the benchmark. Since parameters α and β closely interact in the transition rule (5.3), we considered every combination of values $\alpha = \{1, 2, 3, 4, 5\}$ and $\beta = \{1, 2, 3, 4, 5\}$. The best combination turned out to be $\alpha = 1$ and $\beta = 5$. We optimized in the same way $(1 - \rho)$ and Δ, since they closely interact in the pheromone update equation (5.5). We considered the combination of values $\Delta = \{0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001\}$ with $(1 - \rho) = 0.999999827, 0.999999846, 0.999999861, 0.999999880, 0.9999999, 0.999999920$. The values for $(1 - \rho)$ were chosen around 0.999999861 which is such that half of the pheromone would evaporate on an edge after 5 000 000 iterations, assuming that this edge is never part of an accepted solution. We found that the best values for Δ and $(1 - \rho)$ were equal to 0.000001 and 0.9999999, respectively. The last parameter is the threshold q_0 in Equation (5.3). We tested the values $q_0 = \{0.82, 0.85, 0.87, 0.90, 0.92, 0.95\}$ and the best results were obtained with $q_0 = 0.90$

6.3. RR-ACS Results

Our hybrid RR-ACS algorithm was tested by running it 50 times on each of the 68 medium and large instances, using the best parameter setting, as identified in the previous subsection. The average solution values are shown in Appendix B. We computed the average gap between RR-ACS and SISR on each instance, simply by replacing RR by RR-ACS in Equation (6.1). The mean of these gaps over all test instances is -0.0055 . The negative value indicates that RR-ACS provides solutions of better quality than SISR. Furthermore,

even if the difference is small, the signed-rank Wilcoxon test returned a p-value of 0.041, which is below 0.05. Thus, there is a significant statistical difference in solution quality between the results produced by RR-ACS and those reported in [**14**] for SISR. Also, the ratio of computation times is 0.7086 (using the numbers in [**14**], based on a 2.6 GHz Xeon E5-2650 v2 processor), which means a reduction of 29.2%, on average. But, again, this observation must be mitigated because our processor and computer environment are not the same than those in [**14**].

The bad news is that the average gap between RR and RR-ACS over all test instances is only -0.0022. Although the integration of a learning component into our implementation of SISR has produced slightly better solutions on average, this improvement is not statistically significant. In addition, RR-ACS is 56% more computationally expensive than RR.

7. Conclusions

This paper has introduced a hybrid RR-ACS algorithm. After extensive computational experiments, this hybrid algorithm has produced slightly better solutions than the original algorithm. But the learning component leads to increased computation times. As future work, we would like to make our algorithm more efficient through parallel implementation, which would allow a larger number of ants to be used. We think that this is likely to lead to further improvements in solution quality.

Instance	Ave. cost (50 runs)		Ave. time (minutes)	
	SISR	**RR**	**SISR**	**RR**
X-n251-k28	38791.0	38772.16	9.8	7.64
X-n256-k16	18888.9	18883.20	11.5	11.17
X-n261-k13	26642.3	26598.38	11.8	9.37
X-n266-k58	75617.8	75627.70	10.8	7.69
X-n270-k35	35362.2	35369.84	11.4	6.57
X-n275-k28	21268.6	21259.36	13.3	6.14
X-n280-k17	33628.1	33640.14	17.7	11.12
X-n284-k15	20286.6	20276.78	15.3	8.41
X-n289-k60	95352.2	95347.02	14.3	9.33
X-n294-k50	47274.5	47277.54	14.7	9.73
X-n298-k31	34276.0	34277.94	14.5	8.16
X-n303-k21	21776.5	21772.74	17.3	10.64
X-n308-k13	26207.7	26270.30	25.7	16.18
X-n313-k71	94182.4	94248.14	18.9	11.86
X-n317-k53	78392.4	78378.58	22.0	7.55
X-n322-k28	29927.6	29935.44	16.9	8.50
X-n327-k20	27631.4	27635.08	21.6	12.29
X-n331-k15	31128.2	31130.22	20.4	9.99
X-n336-k84	139373.4	139352.68	22.8	14.80
X-n344-k43	42158.5	42142.42	21.5	8.81
X-n351-k40	25982.1	25979.38	26.5	14.76
X-n359-k29	51577.8	51570.86	23.1	11.10
X-n367-k17	22833.4	22832.84	36.1	18.84
X-n376-k94	147783.6	147775.16	32.0	12.43
X-n384-k52	66107.4	66094.18	25.9	10.38
X-n393-k38	38394.1	38376.46	30.4	11.52
X-n401-k29	66248.5	66242.88	38.0	19.44
X-n411-k19	19768.5	19773.26	58.4	29.47
X-n420-k130	107879.2	107887.20	47.9	21.90
X-n429-k61	65593.6	65591.62	35.0	13.58
X-n439-k37	36473.8	36447.22	42.1	16.85
X-n449-k29	55411.2	55402.86	38.0	16.89
X-n459-k26	24242.2	24207.72	56.5	22.54
X-n469-k138	222227.1	222234.54	48.0	25.16
X-n480-k70	89559.2	89546.18	50.5	17.46
X-n491-k59	66645.5	66653.58	51.4	22.89

Table 16. Average costs and times obtained with RR and SISR, for medium-size Uchoa instances.

Instance	Ave. cost (50 runs)		Ave. time (minutes)	
	SISR	**RR**	**SISR**	**RR**
X-n502-k39	69274.7	69270.08	60.9	17.16
X-n513-k21	24292.1	24284.26	77.1	40.29
X-n524-k153	154807.2	154891.22	151.4	74.83
X-n536-k96	95173.2	95146.18	74.7	25.86
X-n548-k50	86798.0	86797.58	64.5	17.28
X-n561-k42	42868.1	42866.92	73.8	29.67
X-n573-k30	50804.6	50832.46	113.0	44.45
X-n586-k159	190600.7	190653.88	86.3	34.63
X-n599-k92	108688.6	108663.82	75.4	24.47
X-n613-k62	59731.3	59691.14	88.1	32.19
X-n627-k43	62317.1	62286.26	89.3	22.69
X-n641-k35	63850.3	63861.14	92.5	26.49
X-n655-k131	106844.6	106848.64	109.6	30.10
X-n670-k130	146720.4	147001.58	198.9	81.70
X-n685-k75	68369.0	68384.22	135.1	45.99
X-n701-k44	82065.4	82049.98	122.5	36.24
X-n716-k35	43483.8	43489.56	158.3	53.54
X-n733-k159	136389.3	136414.90	143.2	55.78
X-n749-k98	77509.2	77494.02	146.3	48.14
X-n766-k71	114761.1	114810.48	174.4	55.08
X-n783-k48	72660.7	72624.98	170.2	65.54
X-n801-k40	73436.7	73421.56	137.1	32.15
X-n819-k171	158423.0	158412.78	172.5	54.69
X-n837-k142	193976.9	193970.86	166.8	46.10
X-n856-k95	89131.3	89093.50	160.0	38.27
X-n876-k59	99483.2	99468.14	217.4	77.09
X-n895-k37	54085.8	54073.62	212.5	81.27
X-n916-k207	329509.5	329508.72	215.3	80.39
X-n936-k151	133117.3	133258.40	412.7	166.94
X-n957-k87	85620.0	85577.98	202.4	47.70
X-n979-k58	119120.4	119096.98	276.6	88.46
X-n1001-k43	72528.1	72532.34	284.3	107.06

Table 17. Average costs and times obtained with RR and SISR, for large-size Uchoa instances.

Appendix B

Instance	Ave. cost (50 runs)		Ave. time (minutes)	
	SISR	**RR-ACS**	**SISR**	**RR-ACS**
X-n251-k28	38791.0	38771.30	9.8	10.97
X-n256-k16	18888.9	18885.72	11.5	15.07
X-n261-k13	26642.3	26596.46	11.8	13.30
X-n266-k58	75617.8	75617.74	10.8	10.91
X-n270-k35	35362.2	35357.66	11.4	10.25
X-n275-k28	21268.6	21255.80	13.3	9.78
X-n280-k17	33628.1	33636.16	17.7	15.57
X-n284-k15	20286.6	20271.78	15.3	12.92
X-n289-k60	95352.2	95346.70	14.3	13.10
X-n294-k50	47274.5	47276.54	14.7	13.89
X-n298-k31	34276.0	34273.98	14.5	12.78
X-n303-k21	21776.5	21774.72	17.3	16.13
X-n308-k13	26207.7	26274.26	25.7	23.27
X-n313-k71	94182.4	94208.92	18.9	16.21
X-n317-k53	78392.4	78390.92	22.0	12.12
X-n322-k28	29927.6	29921.30	16.9	13.78
X-n327-k20	27631.4	27636.38	21.6	19.23
X-n331-k15	31128.2	31136.72	20.4	16.56
X-n336-k84	139373.4	139348.62	22.8	20.04
X-n344-k43	42158.5	42148.02	21.5	14.69
X-n351-k40	25982.1	25980.88	26.5	22.57
X-n359-k29	51577.8	51556.42	23.1	18.55
X-n367-k17	22833.4	22837.74	36.1	29.73
X-n376-k94	147783.6	147772.12	32.0	19.00
X-n384-k52	66107.4	66099.68	25.9	17.39
X-n393-k38	38394.1	38362.10	30.4	20.75
X-n401-k29	66248.5	66243.02	38.0	30.75
X-n411-k19	19768.5	19774.54	58.4	44.52
X-n420-k130	107879.2	107889.82	47.9	30.77
X-n429-k61	65593.6	65569.42	35.0	23.14
X-n439-k37	36473.8	36460.92	42.1	27.89
X-n449-k29	55411.2	55396.38	38.0	29.94
X-n459-k26	24242.2	24207.90	56.5	37.87
X-n469-k138	222227.1	222222.18	48.0	35.40
X-n480-k70	89559.2	89568.06	50.5	29.92
X-n491-k59	66645.5	66644.50	51.4	37.63

Table 18. Average costs and times obtained with RR-ACS and SISR, for medium-size Uchoa instances.

Instance	Ave. cost (50 runs)		Ave. time (minutes)	
	SISR	**RR-ACS**	**SISR**	**RR-ACS**
X-n502-k39	69274.7	69265.50	60.9	32.86
X-n513-k21	24292.1	24277.36	77.1	65.16
X-n524-k153	154807.2	154877.62	151.4	99.20
X-n536-k96	95173.2	95132.14	74.7	41.54
X-n548-k50	86798.0	86795.88	64.5	33.23
X-n561-k42	42868.1	42863.00	73.8	52.21
X-n573-k30	50804.6	50837.32	113.0	76.53
X-n586-k159	190600.7	190653.82	86.3	49.76
X-n599-k92	108688.6	108651.24	75.4	42.51
X-n613-k62	59731.3	59699.68	88.1	54.84
X-n627-k43	62317.1	62285.86	89.3	46.30
X-n641-k35	63850.3	63852.52	92.5	52.97
X-n655-k131	106844.6	106839.90	109.6	53.18
X-n670-k130	146720.4	147032.22	198.9	116.92
X-n685-k75	68369.0	68377.74	135.1	79.09
X-n701-k44	82065.4	82023.00	122.5	70.85
X-n716-k35	43483.8	43489.00	158.3	100.83
X-n733-k159	136389.3	136401.34	143.2	83.23
X-n749-k98	77509.2	77515.00	146.3	82.80
X-n766-k71	114761.1	114789.52	174.4	102.25
X-n783-k48	72660.7	72652.18	170.2	113.70
X-n801-k40	73436.7	73431.16	137.1	70.09
X-n819-k171	158423.0	158437.68	172.5	82.75
X-n837-k142	193976.9	193983.62	166.8	76.97
X-n856-k95	89131.3	89096.48	160.0	76.63
X-n876-k59	99483.2	99456.82	217.4	137.80
X-n895-k37	54085.8	54076.70	212.5	145.95
X-n916-k207	329509.5	329499.16	215.3	112.17
X-n936-k151	133117.3	133321.32	412.7	244.70
X-n957-k87	85620.0	85587.98	202.4	97.60
X-n979-k58	119120.4	119096.08	276.6	158.39
X-n1001-k43	72528.1	72505.80	284.3	188.55

Table 19. Average costs and times obtained with RR-ACS and SISR, for large-size Uchoa instances.

Conclusions

In this thesis, we have developed two algorithms based on *SISR* to solve the *CVRP* through the integration of a learning component and the much more complex variant $TDVRPTWTP_{RN}$, which is defined on a road network. We have also performed a computational study on the impact of inaccurate distances on solution quality for the *CVRP*, using different heuristics and metaheuristics (including *SISR*), as well as an exact solver.

With regard to the $TDVRPTWTP_{RN}$, we had to extensively modify the original *SISR* metaheuristic, in particular by developing specific ruin and recreate operators that take into account the presence of transfer points when handling the routes of small and large vehicles. The multi-attribute nature of this problem was challenging, since it led to complicating factors associated with time-dependency, synchronization of the two types of vehicles at transfer points, etc. We also developed different techniques to make the algorithm more efficient, like assessing infeasibility and (approximately) evaluating insertion places in constant time. The results obtained on a set of test instances derived from a benchmark for the $TDVRPTW_{RN}$ showed good synchronization at transfer points, as well as good optimization capabilities. We also observed that some transfer points are more heavily used than others and that their removal from the road network had a significant impact on solution quality. Accordingly, it could be interesting to identify a priori those transfer points to help focus the search in particular regions of the solution space. This could be done, for example, through a learning component that would account for the topology of the road network, location of transfer points, location and distribution of customers, etc.

Next, the empirical analysis on the impact of data inaccuracies for the *CVRP* led to the observation that state-of-the-art metaheuristics are more robust to data inaccuracies when compared to a simple savings heuristic. Overall, the best metaheuristics remain the best, even in the presence of high levels of inaccuracies, because they have the ability to fix previous bad decisions induced by those inaccuracies. A simple construction heuristic like the savings heuristic does not have this ability. We also observed that when two methods exhibit a sufficiently small, although statistically significant, difference in performance

on true data, this statistically significant difference vanishes with increasing levels of inaccuracies. Overall, our conclusion is that sophisticated problem-solving methods are still indicated, even when data inaccuracies are of larger magnitude than the optimality gap of these methods. This is an important observation because such inaccuracies cannot be avoided in the real-world. Other well-known variants of the *CVRP*, like the *VRPTW* could also be the topic of similar computational studies.

The integration of learning components into metaheuristics to guide the search in the solution space has become very popular lately. The third paper presents a hybrid algorithm, based on the state-of-the-art *SISR* metaheuristic, that incorporates a reinforcement learning component inspired from ant colony systems to solve the *CVRP*. Accordingly, learning applies to the edges of the graph by reinforcing the most promising ones (i.e., those found frequently in previously encountered good solutions). Given that *SISR* requires a large number of iterations and given that the learning component induces additional computational costs, we had to find a compromise between computation time and solution quality (e.g., number of ants). Although the hybrid algorithm could find improved solutions on a standard benchmark over already quasi-optimal solutions, based on our own implementation of *SISR*, this improvement did not prove to be statistically significant. We now envision a parallel implementation of our hybrid algorithm that would allow more ants to be used, thus potentially leading to further improvements.

The combination of machine learning and optimization methods is a very active and promising area. However, a lesson learned from this thesis is that the problem to be solved must be chosen with care, as well as the problem-solving method. In our study on the *VRP*, using a metaheuristic that is already quasi-optimal prevented us to produce statistically significant improvements. Also, the learning component needs to be as simple as possible, to limit the additional computational burden imposed on the original metaheuristic. Other types of problems and methods could have provided a more fertile ground. For example, in vehicle routing problem with loading constraints, the loading aspect of the problem is often addressed with simple heuristics. Learning techniques could certainly be useful to enhance those heuristics. Such learning techniques could even be inspired from the way real people load vehicles in real applications. The same opportunities are present in other well chosen applications, like vehicle dispatching, where a real dispatcher must handle mutiple variables (some of which can hardly be integrated in a standard optimization method) before taking a decision.

Bibliography

[1] M.M. Alipour. A learning automata based algorithm for solving capacitated vehicle routing problem. In *International Journal of Computer Science Issues*, volume 9, 2012.

[2] Alexandra Anderluh, Vera Hemmelmayr, and Pamela Nolz. Synchronizing vans and cargo bikes in a city distribution network. *Central European Journal of Operations Research*, 25, 06 2017.

[3] Abhinav Atla, Rahul Tada, Victor Sheng, and Naveen Singireddy. Sensitivity of different machine learning algorithms to noise. *Journal of Computing Sciences in Colleges*, 26(5):96–103, 2011.

[4] S.R. Balseiro, I. Loiseau, and J. Ramonet. An ant colony algorithm hybridized with insertion heuristics for the time dependent vehicle routing problem with time windows. *Computers & Operations Research*, 38(6):954 – 966, 2011.

[5] John Beasley. Adapting the savings algorithm for varying inter-customer travel times. *Omega*, 9:658–659, 02 1981.

[6] Hamza Ben Ticha. *Vehicle Routing Problems with road-network information.* PhD thesis, 11 2017.

[7] Hamza Ben Ticha, Nabil Absi, Dominique Feillet, and Alain Quilliot. Empirical analysis for the vrptw with a multigraph representation for the road network. *Computers & Operations Research*, 88:103–116, 2017.

[8] Hamza Ben Ticha, Nabil Absi, Dominique Feillet, and Alain Quilliot. Multigraph modeling and adaptive large neighborhood search for the vehicle routing problem with time windows. *Computers & Operations Research*, 104:113–126, 2019.

[9] Hamza Ben Ticha, Nabil Absi, Dominique Feillet, Alain Quilliot, and Tom Van Woensel. A branch-and-price algorithm for the vehicle routing problem with time windows on a road network. *Networks*, 73, 09 2018.

[10] Hamza Ben Ticha, Nabil Absi, Dominique Feillet, Alain Quilliot, and Tom Van Woensel. The time-dependent vehicle routing problem with time windows and road-network information. *SN Operations Research Forum*, 2, 01 2021.

[11] Panagiotis Bouros, Dimitris Sacharidis, Theodore Dalamagas, and Timos Sellis. Dynamic pickup and delivery with transfers. In *Advances in Spatial and Temporal Databases*, pages 112–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[12] Bernd Bullnheimer, Richard Hartl, and Christine Strauss. An improved ant system algorithm for the vahicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[13] Félix Caron. Résolution d'un problème de collecte et livraison dynamique sur un réseau routier avec temps de parcours variables. Master's thesis, University of Montreal, 03 2022.

[14] Jan Christiaens and Greet Van den Berghe. A fresh ruin & recreate implementation for the capacitated vehicle routing problem. 2016.

[15] Jan Christiaens and Greet Van den Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433, 2020.

[16] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[17] Anthony C. Constantinou, Yang Liu, Kiattikun Chobtham, Zhigao Guo, and Neville K. Kitson. Large-scale empirical validation of Bayesian network structure learning algorithms with noisy data. *International Journal of Approximate Reasoning*, 131:151–188, 2021.

[18] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Multi-start heuristics for the two-echelon vehicle routing problem. volume 6622, pages 179–190, 04 2011.

[19] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Models for evaluating and planning city logistics systems. *Transportation Science*, 43:432–454, 11 2009.

[20] Said Dabia, S Röpke, Tom Van Woensel, and Ton de Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Journal of Graph Algorithms and Applications - JGAA*, 47, 01 2011.

[21] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[22] J. De-gang and H. Dong-mei. A research based on k-means clustering and artificial fish-swarm algorithm for the vehicle routing optimization. In *2012 8th International Conference on Natural Computation*, pages 1141–1145, 2012.

[23] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–157, 2000.

[24] DIMACS. 12th DIMACS implementation challenge: Vehicle routing. `http://dimacs.rutgers.edu/programs/challenge/vrp/`, April 2022.

[25] Alberto Donati, Roberto Montemanni, Norman Casagrande, Andrea-Emilio Rizzoli, and Luca Maria Gambardella. Time dependent vehicle routing problem with a multi ant colony. *European Journal of Operational Research*, 185:1174–1191, 02 2008.

[26] M Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477 Vol. 2, July 1999.

[27] M Dorigo and L.M. Gambardella. Ant colony system:a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[28] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[29] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1:28–39, 12 2006.

[30] Raafat Elshaer and Hadeer Awad. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, 140:106242, 2020.

[31] Yuvraj Gajpal and Prakash Abad. Multi-ant colony system (macs) for a vehicle routing problem with backhauls. *European Journal of Operational Research*, 196:102–117, 07 2009.

[32] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pages 63–76. McGraw-Hill, 1999.

[33] Thierry Garaix, Christian Artigues, Dominique Feillet, and Didier Josselin. Vehicle routing problems with alternative paths: An application to on-demand transportation. *European Journal of Operational Research*, 204(1):62–75, 2010.

[34] S. Geetha, G. Poonthalir, and P.T. Vanathi. Nested particle swarm optimization for multi-depot vehicle routing problem. *Int. J. of Operational Research*, 16:329–348, 01 2013.

[35] Michel Gendreau, Gianpaolo Ghiani, and Emanuela Guerriero. Time-dependent routing problems: A review. *Computers & Operations Research*, 64:189–197, 2015.

[36] H. Ghaziri and I. H. Osman. Self-organizing feature maps for the vehicle routing problem with backhauls. *Journal of Scheduling*, 9:97–114, 2006.

[37] Hassan Ghaziri and Ibrahim H Osman. A neural network algorithm for the traveling salesman problem with backhauls. *Computers & Industrial Engineering*, 44(2):267 – 281, 2003.

[38] Maha Gmira, Michel Gendreau, Andrea Lodi, and Jean-Yves Potvin. Tabu search for the time-dependent vehicle routing problem with time windows on a road network. *European Journal of Operational Research*, 288(1):129–140, 2021.

[39] E. Göçmen and E. Rızvan. Transportation problems for intermodal networks: Mathematical models, exact and heuristic algorithms, and machine learning. *Expert Systems with Applications*, 135:374 – 387, 2019.

[40] Philippe Grangier, Michel Gendreau, Fabien Lehuédé, and Louis-Martin Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1):80–91, 2016.

[41] Shivani Gupta and Atul Gupta. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474, 2019.

[42] Ali Haghani and Soojung Jung. A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, 32(11):2959–2986, 2005.

[43] Reihaneh H. Hariri, Erik M. Fredericks, and Kate M. Bowers. Uncertainty in big data analytics: Survey, opportunities, and challenges. *Journal of Big Data*, 6:44, 2019.

[44] Arthur V. Hill and W. C. Benton. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *Journal of the Operational Research Society*, 43:343–351, 04 1992.

[45] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379–396, 2003.

[46] Shengyang Jia, Lei Deng, Quanwu Zhao, and Yunkai Chen. An adaptive large neighborhood search heuristic for multi-commodity two-echelon vehicle routing problem with satellite synchronization. *Journal of Industrial and Management Optimization*, 19(2):1187–1210, 2023.

[47] Nabin Kafle, Bo Zou, and Jane Lin. Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery. *Transportation Research Part B: Methodological*, 99:62–82, 2017.

[48] Can B. Kalayci and Can Kaya. An ant colony system empowered variable neighborhood search algorithm for the vehicle routing problem with simultaneous pickup and delivery. *Expert Systems with Applications*, 66:163–175, 2016.

[49] Jorge Kanda, André Carlos Ponce de Leon Ferreira de Carvalho, Eduardo R. Hruschka, and Carlos Soares. Using meta-learning to recommend meta-heuristics for the traveling salesman problem. *2011 10th International Conference on Machine Learning and Applications and Workshops*, 1:346–351, 2011.

[50] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 41(3):552–568, 2011.

[51] Byung-In Kim, Seongbae Kim, and Surya Sahoo. Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, 33:3624–3642, 2006.

[52] Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43:408–416, 11 2009.

[53] Gilbert Laporte, Paolo Toth, and Daniele Vigo. Vehicle routing: Historical perspective and recent contributions. *EURO Journal on Transportation and Logistics*, 2, 05 2013.

[54] A. Le Bouthillier, T. G. Crainic, and P. Kropf. A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems*, 20(4):36–42, July 2005.

[55] Adam N. Letchford, Saeideh D. Nasiri, and Amar Oukil. Pricing routines for vehicle routing with time windows on road networks. *Computers & Operations Research*, 51:331–337, 2014.

[56] Fei Liu and Guangzhou Zeng. Study of genetic algorithm with reinforcement learning to solve the tsp. *Expert Systems with Applications*, 36(3):6995–7001, 2009.

[57] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[58] Chryssi Malandraki and Mark Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, 26:185–200, 08 1992.

[59] P. Melville, N. Shah, L. Mihalkova, and R.J. Mooney. Experiments on ensembles with missing and noisy data. *Lecture Notes in Computer Science*, 3077:293–302, 2004.

[60] Snezana Minic and Gilbert Laporte. The pickup and delivery problem with time windows and transshipment. *INFOR*, 44:217–227, 08 2006.

[61] Zahra Nazari, Masooma Nazari, Mir Sayed Shah Danish, and Dongshik Kang. Evaluation of class noise impact on performance of machine learning algorithms. *International Journal of Computer Science and Network Security*, 18(8):148–153, 2018.

[62] Binbin Pan, Zhenzhen Zhang, and Andrew Lim. A hybrid algorithm for time-dependent vehicle routing problem with time windows. *Computers & Operations Research*, 128:105193, 2021.

[63] Binbin Pan, Zhenzhen Zhang, and Andrew Lim. Multi-trip time-dependent vehicle routing problem with time windows. *European Journal of Operational Research*, 291(1):218–231, 2021.

[64] Guido Perboli, Roberto Tadei, and Roberto Tadei. New families of valid inequalities for the two-echelon vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 36:639–646, 2010. ISCO 2010 - International Symposium on Combinatorial Optimization.

[65] Guido Perboli, Roberto Tadei, and Daniele Vigo. The two-echelon capacitated vehicle routing problem: Models and math-based heuristics. *Transportation Science*, 45, 01 2009.

[66] Laurent Perron and Vincent Furnon. *Or-tools*. Google, France, 2019. `https://developers.google.com/optimization/`.

[67] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183:483–523, 2020.

[68] Caroline Prodhon and Christian Prins. *Metaheuristics for Vehicle Routing Problems*, pages 407–437. Springer International Publishing, Cham, 2016.

[69] Marc Reimann, Michael Stummer, and Karl Doerner. A savings based ant system for the vehicle routing problem. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, page 1317–1326. Morgan Kaufmann Publishers Inc., 2002.

[70] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, 11 2006.

[71] Afonso Sampaio, Martin Savelsbergh, Lucas Veelenturf, and Tom Van Woensel. Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers. *Networks*, 76, 07 2020.

[72] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139 – 171, 2000.

[73] B. Śnieżyński, W. Wójcik, J. D. Gehrke, and J. Wojtusiak. Combining rule induction and reinforcement learning: An agent-based vehicle routing. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 851–856, 2010.

[74] M.Grazia Speranza, Gianfranco Guastaroba, and Daniele Vigo. Intermediate facilities in freight transportation planning: A survey. *Transportation Science*, 50:763–789, 08 2016.

[75] N. Sushma Rani, Anurag, and P. Srinivasa Rao. Study and analysis of noise effect on big data analytics. *International Journal of Management, Technology and Enginnering*, 8(12):5841–5850, 2019.

[76] José A. Sáez, Mikel Galar, Julián Luengo, and Francisco Herrera. Tackling the problem of classification with noisy data using multiple classifier systems: Analysis of the performance and robustness. *Information Sciences*, 247:1–20, 2013.

[77] José A. Sáez, Julián Luengo, and Francisco Herrera. Evaluating the classifier behavior with noisy data considering performance and robustness: The equalized loss of accuracy measure. *Neurocomputing*, 176:26–35, 2016.

[78] C.D. Tarantilis and C.T. Kiranoudis. Boneroute: An adaptive memory-based method for effective fleet management. *Annals of Operations Research*, 115(1):227–241, 2002.

[79] Tulio A. M. Toffolo, Jan Christiaens, Sam Van Malderen, Tony Wauters, and Greet Vanden Berghe. Stochastic local search with learning automaton for the swap-body vehicle routing problem. *Computers and Operations Research*, 89:68–81, 2018.

[80] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

[81] Raras Tyasnurita, Ender Ozcan, Asta Shahriar, and Robert John. Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing. 2015.

[82] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845 – 858, 2017.

[83] Jan Van Belle, Paul Valckenaers, and Dirk Cattrysse. Cross-docking: State of the art. *Omega*, 40(6):827–846, 2012. Special Issue on Forecasting in Management Science.

[84] Jason Van Hulse and Taghi Khoshgoftaar. Knowledge discovery from imbalanced and noisy data. *Data & Knowledge Engineering*, 68(12):1513–1542, 2009.

[85] T. Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[86] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013.

[87] Mahmoud Zennaki and Adnane Cherif. A new machine learning based approach for tuning metaheuristics for the solution of hard combinatorial optimization problems. 2010.