

EXACT SOLUTION APPROACHES FOR NON-HAMILTONIAN VEHICLE ROUTING PROBLEMS

A DISSERTATION SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
INDUSTRIAL ENGINEERING

By
Amine Gizem Özbaygın
July 2017

EXACT SOLUTION APPROACHES FOR NON-HAMILTONIAN
VEHICLE ROUTING PROBLEMS

By Amine Gizem Özbaygın

July 2017

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Hande Yaman Paternotte(Advisor)

Oya Karaşan(Co-Advisor)

M. Selim Aktürk

İbrahim Akgün

Haldun Süral

Firdevs Ulus

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

EXACT SOLUTION APPROACHES FOR NON-HAMILTONIAN VEHICLE ROUTING PROBLEMS

Amine Gizem Özbaygın

Ph.D. in Industrial Engineering

Advisor: Hande Yaman Paternotte

Co-Advisor: Oya Karaşan

July 2017

In this thesis, we study different non-Hamiltonian vehicle routing problem variants and concentrate on developing efficient optimization algorithms to solve them.

First, we consider the split delivery vehicle routing problem (SDVRP). We provide a vehicle-indexed flow formulation for the problem, and then, a relaxation obtained by aggregating the vehicle-indexed variables over all vehicles. This relaxation may have optimal solutions where several vehicles exchange loads at some customers. We cut-off such solutions either by extending the formulation locally with vehicle-indexed variables or by node splitting. We compare these approaches using instances from the literature and new randomly generated instances. Additionally, we introduce two new extensions of the SDVRP by restricting the number of splits and by relaxing the depot return requirement, and modify our algorithms to handle these extensions.

Second, we focus on a problem unifying the notion of coverage and routing. In some real-life applications, it may not be viable to visit every single customer separately due to resource limitations or efficiency concerns. In such cases, utilizing the notion of coverage; i.e., satisfying the demand of multiple customers by visiting a single customer location, may be advantageous. With this motivation, we study the time constrained maximal covering salesman problem (TCMCSP) in which the aim is to find a tour visiting a subset of customers so that the amount of demand covered within a limited time is maximized. We provide flow and cut formulations and derive valid inequalities. Since the connectivity constraints and the proposed valid inequalities are exponential in the size of the problem, we devise different branch-and-cut schemes. Computational experiments performed on a set of problem

instances demonstrate the effectiveness of the proposed valid inequalities in terms of strengthening the linear relaxation bounds as well as speeding up the solution procedure. Moreover, the results indicate the superiority of using a branch-and-cut methodology over a flow-based formulation. Finally, we discuss the relation between the problem parameters and the structure of optimal solutions based on the results of our experiments.

Third, we study the vehicle routing problem with roaming delivery locations (VR-PRDL) in which a customer order has to be delivered to the trunk of the customer’s car during the time that the car is parked at one of the locations in the (known) customer’s travel itinerary. We formulate the problem as a set covering problem and develop a branch-and-price algorithm for its solution. The algorithm can also be used for solving a more general variant in which a hybrid delivery strategy is considered that allows a delivery to either a customer’s home or to the trunk of the customer’s car. We evaluate the effectiveness of the many algorithmic features incorporated in the algorithm in an extensive computational study and analyze the benefits of these innovative delivery strategies. The computational results show that employing the hybrid delivery strategy results in average cost savings of nearly 20% for the instances in our test set.

Finally, we consider the dynamic version of the VRPRDL in which customer itineraries may change during the execution of the planned delivery schedule, which can become infeasible or suboptimal as a result. We refer to this problem as the dynamic VRPRDL (D-VRPRDL) and propose an iterative solution framework in which the previously planned vehicle routes are re-optimized whenever an itinerary update is revealed. We use the branch-and-price algorithm developed for the static VR-PRDL both for solving the planning problem (to obtain an initial delivery schedule) and for solving the re-optimization problems. Since many re-optimization problems may have to be solved during the execution stage, it is critical to produce solutions to these problems quickly. To this end, we devise heuristic procedures through which the columns generated during the previous branch-and-price executions can be utilized when solving a re-optimization problem. In this way, we may be able to save time that would otherwise be spent in generating columns which have already been (partially) generated when solving the previous problems, and find optimal solutions

or at least solutions of good quality reasonably quickly. We perform preliminary computational experiments and report the results.

Keywords: Vehicle routing, split delivery, extended formulations, valid inequalities, covering salesman, branch-and-cut, branch-and-price, resource-constrained shortest path.

ÖZET

HAMILTON OLMAYAN ARAÇ ROTALAMA PROBLEMLERİ İÇİN KESİN ÇÖZÜM YAKLAŞIMLARI

Amine Gizem Özbaygın

Endüstri Mühendisliği, Doktora

Tez Danışmanı: Hande Yaman Paternotte

İkinci Tez Danışmanı: Oya Karaşan

Temmuz 2017

Bu tezde, Hamilton olmayan araç rotalama problemlerinin farklı varyasyonları üzerine çalışılmış ve bu problemleri etkin bir biçimde çözebilmek için eniyileme algoritmaları geliştirilmiştir.

İlk olarak, bölünmüş teslimli araç rotalama problemi (BTARP) ele alınmıştır. Bu problem için araç endeksli akış değişkenleri içeren bir matematiksel model önerilmiş ve karar değişkenleri tüm araç endeksleri üzerinden toplanarak gevşetilmiş bir model elde edilmiştir. Gevşetilmiş modelin eniyilenmesi sonucunda bulunan çözümlerde, bazı talep noktalarında, birden fazla araç arasında yük değişimi gerçekleşebildiği gözlemlenmiştir. Bu yapıya sahip çözümleri olurlu çözüm kümesinden elemek için gevşetilmiş modelin yerel olarak genişletilmesine dayalı yöntemler geliştirilmiştir. Önerilen yöntemler literatürde bulunan problem örnekleri ve yeni rastgele yaratılmış örnekler üzerinde test edilmiş ve karşılaştırılmıştır. Ayrıca, talep noktalarına yapılabilecek teslimat sayısı kısıtlanarak ve araçların depoya geri dönme zorunluluğu ortadan kaldırılarak BTARP'nin iki yeni varyasyonu tanımlanmış ve BTARP için geliştirilen algoritmalar kullanılarak bu varyasyonların da çözülebileceği gösterilmiştir.

Tezin ikinci bölümünde, kapsama ve rotalama kavramlarını bir araya getiren bir probleme odaklanılmıştır. Bazı durumlarda, kısıtlı kaynakların verimli bir biçimde kullanılabilmesi için, her talep noktasını ayrı ayrı ziyaret etmek yerine, bunlar arasından seçilen daha az sayıda talep noktasını içeren bir rota bulmak daha avantajlıdır. Çünkü bu şekilde, rota üzerinde olmayan, ancak ziyaret

edilen noktalara makul bir mesafe katederek ulaşabilecek olan noktaların da taleplerini kısmen karşılamak mümkün olabilir. Buradan yola çıkarak tanımlanan zaman kısıtlı maksimal kapsayan satıcı probleminde amaç, belirli bir zaman kısıtı altında, talep noktalarının bir altkümesini ziyaret ederek, toplamda kapsanan talep miktarını ençoklayan rotayı bulmaktır. Bu problem için akış ve kesi tabanlı formülasyonlar ve geçerli eşitsizlikler önerilmiştir. Alt tur eleme kısıtları ve önerilen eşitsizliklerden bazıları üstel sayıda olduğundan, problemi çözmek için dal ve kesi algoritmaları geliştirilmiştir. Yapılan sayısal analizler, dal ve kesi algoritmalarının problemi akış modeline kıyasla çok daha etkin bir biçimde çözebildiğini, önerilen geçerli eşitsizliklerin, kesi formülasyonunun doğrusal gevşetme sınırlarını oldukça güçlendirdiğini ve çözüm sürelerini ciddi oranda azalttığını göstermiştir. Ayrıca, sayısal analizlerden elde edilen sonuçlar kullanılarak, problem parametrelerindeki değişikliklerin eniyi çözümün yapısına olan etkileri de incelenmiştir.

Tezin üçüncü bölümünde, gezici teslimat noktalı araç rotalama problemi (GT-NARP) çalışılmıştır. Bu problemde, müşterilerin gün içinde ziyaret edip belirli bir zaman geçireceği konumların bilindiği varsayılmaktadır. Amaç, her müşterinin siparişinin, müşterinin aracının bagajına (araç verilen konumlardan herhangi birinde park halindeyken) teslim edilmesini sağlayacak, en düşük maliyetli rotaları belirlemektir. GTNARP, küme kapsama problemi olarak modellenmiş ve çözümü için etkin bir dal ve fiyat algoritması geliştirilmiştir. Bu algoritma, problemin daha genel ve hibrit bir teslimat stratejisi benimseyen (teslimatın bagaja veya eve yapılmasına izin veren) bir varyasyonunu da çözebilmektedir. Algoritmanın performansını geliştirmek için kullanılan yöntemleri test etmek ve yenilikçi teslimat stratejilerinin faydalarını araştırmak amacıyla sayısal analizler yapılmıştır. Elde edilen sonuçlar, önerilen dal ve fiyat algoritmasının büyük ölçekli problem örneklerini bile oldukça etkin bir biçimde çözebildiğini ve hibrit teslimat stratejisi kullanıldığında toplam maliyetin ortalama %20 oranında azaltılabileceğini göstermiştir.

Tezin son bölümünde, müşterilerin planlarının teslimatlar başladıktan sonra değişebileceği göz önünde bulundurularak, GTNARP'nin dinamik bir varyasyonu ele alınmıştır. Bu değişiklikler sonucu, gün başında planlanan teslimat rotalarına bağlı kalmak mümkün olmayabilir veya daha düşük maliyete sahip alternatif rotalar ortaya çıkabilir. Dinamik GTNARP için, tezin bir önceki bölümünde bahsi

geçen dal ve fiyat algoritmasını yinelemeli biçimde kullanan bir çözüm yaklaşımı önerilmiştir. Temel olarak, müşteri planlarındaki her değişiklikten sonra dal ve fiyat algoritması çalıştırılır ve teslimat rotalarının henüz yapılmamış teslimatları içeren kısımları yeniden planlanır. Rotaların sıklıkla güncellenmesi gerekebileceğinden, teslimatların aksamaması için yeniden planlamanın hızlı bir şekilde yapılması oldukça kritiktir. Bu sebeple, yeniden planlama problemleri çözülürken, dal ve fiyat algoritmasının önceki yinelemelerde türettiği sütunları kullanılabilir duruma getiren sezgisel yöntemler geliştirilmiştir. Böylece, dal ve fiyat algoritmasının, sütun türetmeye daha az zaman ayırarak, eniyi çözüme daha hızlı ulaşması veya kısa bir süre içinde eniyiye yakın çözümler bulması sağlanabilmektedir. Önerilen yöntemleri test etmek için bir ön hesaplama çözümlemesi gerçekleştirilmiş ve elde edilen sonuçlar rapor edilmiştir.

Anahtar sözcükler: Araç rotalama, bölünmüş teslimat, genişletilmiş formülasyonlar, geçerli eşitsizlikler, kapsayan satıcı, dal ve kesi, dal ve fiyat, kaynak kısıtlı en kısa yol.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my advisors Prof. Hande Yaman and Prof. Oya Karaşan for their everlasting support and guidance during my Ph.D. studies. They have been excellent mentors to me with their patience, enthusiasm, and immense knowledge. I consider myself very privileged and could not have imagined better advisors to complete this challenging yet spectacular journey with. As happy as I am to get my Ph.D., I know that I will soon miss being a Ph.D. student, and the blame is on them for making our time together so enjoyable. No words can describe my admiration for them, but I genuinely hope that I can be as much an inspiration to my students some day as they have been to me.

Second, I would like to thank the members of my thesis committee Prof. Selim Aktürk and Assoc. Prof. İbrahim Akgün for spending valuable time on meticulously reading each and every progress report I have written within the past five years, and of course, my thesis, for challenging me with their visionary questions, and for contributing their encouraging and insightful comments throughout my dissertation research. I am also very grateful to Prof. Haldun Süral and Asst. Prof. Firdevs Ulus for accepting to serve as members of my dissertation examination committee, taking the time off their busy schedules to read this thesis, and for their valuable suggestions.

My special appreciation is reserved for one of the kindest people I have met, Prof. Martin Savelsbergh, who gave me the amazing opportunity to visit Georgia Tech and the great pleasure of working with him. I genuinely enjoyed every moment I spent there and it turned out to be a life-changing experience for me. I cannot thank Prof. Savelsbergh enough for devoting his valuable time and resources to enrich my visit. I remain amazed at how fast he responds to my e-mails and his willingness to meet me even at short notice despite his busy schedule. He is truly an inspiration, and although he was not officially my Ph.D. advisor, I am indebted to him for acting like one.

Very special thanks to the Department of Industrial Engineering at Bilkent University for giving me a solid background and the opportunity to carry out my doctoral research. I am truly grateful to each faculty member, but I would like to express my

wholehearted gratitude to Prof. Barbaros Tansel, who was my Ph.D. advisor before he suddenly passed away, for believing in me in the first place and for his outstanding mentorship. In addition, I am particularly thankful to Prof. Nesim Erkip for his invaluable guidance and support, and for the many delightful conversations we had.

My heartfelt thanks go to the special friends I have met during my time at Bilkent: I would like to thank Ece Zeliha Demirci for being like a sister to me and a fantastic travel companion, we shared many great memories together and I sincerely wish that there are many more to come. I thank Esra Koca, Ahmed Burak Paç, Hatice Çalık, Ramez Kian, Sinan Bayraktar, Merve Meraklı, Okan Arslan, Barış Yıldız and Fırat Kılıcı, who are no longer at Bilkent, but our friendship never gets old. Fortunately, I also have many friends that are still around and I owe each and every one of them a huge “thank you”: my former housemate and dear friend Meltem Peker Sarhan, my awesome officemates Nihal Berktaş, Kamyar Kargar, Halil İbrahim Bayrak, Özge Şafak, Oğuzhan Efe Şakrak, and my close friends Bengisu-Okan Dükkancı, Halenur Şahin, İrfan Mahmutoğulları, Haşim Özlü and Kübra Şahin. My oldest and dearest friend Nur Timurlenk deserves the biggest appreciation for sticking with me for the past 22 years.

I have had the chance to meet many wonderful people while at Georgia Tech, so I would also like to take this opportunity to thank İlke Bakır, Ezgi Karabulut, İdil Arşık, Fatma Karagöz and Beste Başçiftci for their invaluable friendship. They are among the most sincere people I have ever known, and I feel incredibly lucky to have them as my friends.

There are no words to express the feelings I have for my parents Mine and Ömer Özbaygın, my brother Sinan, my sister Gülşah, and of course, and my lovely grandparents Altıngül and Yılmaz Akartuna. I am forever grateful for their constant, unconditional love and support, and for encouraging me to pursue my interests, even when they went beyond boundaries of language, field and geography. I would not be here if it were not for my family.

Last but not least, I would like to thank my best friend, my love, the constant source of my strength and inspiration, and my all-time pep-talker Murat Tiniç. I would not be able to thank him enough if I dedicated a hundred pages to him. His existence gives me the determination and the power to succeed in anything I work

for. I am thankful to him for never giving up on me even when I doubted myself and for bearing with me during this exceptional time of my life.

I gratefully acknowledge the financial support provided by The Scientific and Technological Research Council of Turkey (TÜBİTAK) with grant numbers BİDEB-2211 and BİDEB-2214A for funding this research.

Barbaros Hocam'a...

Contents

1	Introduction	1
2	Literature Review	7
2.1	Split delivery vehicle routing problem	7
2.2	Time constrained maximal covering salesman problem	9
2.3	Vehicle routing problem with roaming delivery locations	11
2.4	Dynamic vehicle routing problem with roaming delivery locations	12
3	New Exact Solution Approaches for the Split Delivery Vehicle Routing Problem	14
3.1	Formulation, relaxation and valid inequalities	15
3.1.1	An exact flow based formulation with vehicle indices	16
3.1.2	A flow based relaxation	18
3.1.3	An optimality property	18
3.1.4	Comparison with existing relaxations	20
3.1.5	Framed capacity inequalities	21
3.1.6	Rounded capacity and cutset inequalities	24
3.2	New exact methods for the SDVRP	24
3.2.1	Patching algorithm	26
3.2.2	Node-split algorithm	28
3.3	Computational Study	31
3.4	Extensions	36
3.4.1	SDVRP with at most r splits	38
3.4.2	SDVRP with open routes	41

3.5	Conclusion	44
4	Time Constrained Maximal Covering Salesman Problem with Weighted Demands and Partial Coverage	48
4.1	Formulation and Valid Inequalities	50
4.1.1	Mathematical formulations	51
4.1.2	Lifting connectivity constraints	53
4.1.3	Optimality cuts and simple cover inequalities	55
4.2	Branch-and-Cut Algorithms	57
4.2.1	Separation of connectivity constraints	57
4.2.2	Separation of lifted connectivity and cover inequalities	59
4.2.3	Other implementation details	61
4.3	Computational Study	61
4.3.1	Comparison of the proposed algorithms	62
4.3.2	Impacts of parameter changes on the optimal solutions	70
4.4	Conclusion	74
5	A Branch-and-Price Algorithm for the Vehicle Routing Problem with Roaming Delivery Locations	77
5.1	Problem definition and formulations	80
5.1.1	Pricing problem	82
5.1.2	Solving the pricing problem	83
5.2	A branch-and-price algorithm	91
5.2.1	Heuristic pricing	91
5.2.2	Bidirectional search	93
5.2.3	Branching	94
5.2.4	Initial set of columns and feasible solutions	95
5.2.5	Handling the tailing-off effect	96
5.2.6	Implementation details	98
5.3	Incorporating a home delivery option	101
5.4	Computational study	101
5.4.1	Instances and preprocessing	102
5.4.2	Evaluating the performance of the branch-and-price algorithm	106

5.4.3	Impact of distance of roaming delivery locations to the depot on algorithm performance	114
5.4.4	Assessing the benefits of employing trunk delivery services . .	115
5.5	Concluding remarks	118
6	An Iterative Re-optimization Framework for the Dynamic Vehicle Routing Problem with Roaming Delivery Locations	122
6.1	Problem definition and formulations	125
6.1.1	Static problem formulation	128
6.1.2	Pricing problems	129
6.2	An iterative re-optimization framework	131
6.2.1	Constructing and preprocessing the graph of SP	131
6.2.2	Solving the pricing problems	132
6.2.3	Initial set of columns for SP	133
6.2.4	Generating more initial columns	139
6.3	Computational study	144
6.3.1	Test instances & update generation scheme	144
6.3.2	Implementation and experimental setup	145
6.3.3	Preliminary results	146
6.4	Concluding remarks	149
7	Conclusion	157

List of Figures

3.1	The optimal solution of R-SDVRP for <i>eil30</i>	21
3.2	An optimal solution to R-SDVRP that cannot be cut-off by any framed capacity inequality	23
4.1	The optimal solutions of <i>p03</i> with $L = L_1$, $r = 10$ for $\alpha = 0.5$ and $\alpha = 0.75$ respectively	71
4.2	The optimal solutions of <i>p11</i> with $L = L_1$, $r = 10$ for $\alpha = 0.5$ and $\alpha = 0.75$ respectively	72
4.3	The optimal solutions of <i>p03</i> with $\alpha = 0.5$, $r = 10$ for $L = L_1$, $L = L_2$ and $L = L_3$ respectively	73
4.4	The optimal solutions of <i>p11</i> with $\alpha = 0.5$, $r = 10$ for $L = L_1$, $L = L_2$ and $L = L_3$ respectively	74
4.5	The optimal solutions of <i>p04</i> with $\alpha = 0.5$, $L = L_1$ for $r = 10$ and $r = 20$ respectively	75
4.6	The optimal solutions of <i>p12</i> with $\alpha = 0.75$, $L = L_2$ for $r = 10$ and $r = 20$ respectively	76
5.1	The optimal VRP, VRPRDL and VRPHRDL solutions for Instance 28 from top to bottom, respectively	120
6.1	Two feasible routes obtained by route splitting	136
6.2	Feasible routes obtained by removing nodes 3 and 5 from the route	138
6.3	An example tree of columns	143

List of Tables

3.1	Some results with the vehicle-indexed model	17
3.2	Results for the instances taking single iteration for the SDVRP	34
3.3	Results for the instances taking multiple iterations for the SDVRP . .	35
3.4	Results for the SDVRP instances with non-rounded costs	37
3.5	Results for the instances taking single iteration for the SDVRP with at most two splits	40
3.6	Results for the instances taking multiple iterations for the SDVRP with at most two splits	41
3.7	Results for the instances taking single iteration for the SDVRP with at most two splits when (3.43) is relaxed	42
3.8	Results for the instances taking multiple iterations for the SDVRP with at most two splits when (3.43) is relaxed	43
3.9	Results for the instances taking single iteration for the SDOVRP . . .	45
3.10	Results for the instances taking multiple iterations for the SDOVRP .	46
4.1	Results with the flow formulation	63
4.2	Results with branch-and-cut scheme 1	65
4.3	Results with branch-and-cut scheme 2	66
4.4	Results with branch-and-cut scheme 3	68
4.5	Results with branch-and-cut scheme 4	69
5.1	Characteristics of the instances in the first set	105
5.2	Characteristics of the instances in the second set	106
5.3	Results with the straightforward BAP	107
5.4	Results with CB , MF , and MFC arc selection rules	109

5.5	Results obtained with the selected settings on the large instances . . .	112
5.6	Straightforward branch-and-price vs. the default branch-and-price with parameter configuration $(5, 5, F)$ and the MF branching rule . . .	113
5.7	Results for Instances 41–50 obtained by enhanced branch-and-price with $MF(5, 5, F)$	114
5.8	Results for the VRPHRDL instances with parameter configuration $(10, 10, F)$ and the MFC branching rule	116
5.9	Comparison of the VRP, the VRPRDL, and the VRPHRDL solutions	119
5.10	Comparison of the VRP, the VRPRDL, and the VRPHRDL solutions for instances in the second set.	121
6.1	Results with <code>Soln_Cols</code> , <code>All_Feas_Cols</code> , and <code>All_Cols</code> strategies on small and medium instances	154
6.2	Results with <code>All_Feas_Cols</code> and <code>All_Cols</code> on large instances	155
6.3	Results obtained by iterative framework with <code>All_Cols</code> on small and medium instances	156

Chapter 1

Introduction

The vehicle routing problem (VRP) was introduced almost 60 years ago in [1] as “The truck dispatching problem”, where the aim was to determine the optimal vehicle routes for delivering gasoline from a bulk terminal to a number of service stations. The authors formulated the problem mathematically and developed an algorithm which gives a near optimal solution. Since then, the VRP has been extensively studied in the literature due to its practical significance in distribution management. Numerous companies and organizations involved in collection/delivery services face the VRP every day although the problem characteristics may differ depending on the application under consideration. Several variants of the problem have been proposed to address various objectives and constraints encountered in practice such as the capacitated VRP (CVRP), the VRP with time windows (VRPTW), the VRP with pickup and delivery (VRPPD), the split delivery VRP (SDVRP), the generalized VRP (GVRP) as well as many others including stochastic and dynamic VRPs.

The basic version of the VRP can be defined as the problem of identifying a least cost set of routes for a vehicle fleet to serve a set of geographically dispersed customers such that each route originates from and returns to a specified depot location and every customer is exactly in one of the routes. Depending on the problem variant,

there may be additional side constraints. For example, in the CVRP, vehicles are assumed to have limited capacity, and thus, the total demand covered by any of the routes should not exceed the vehicle capacity. In the VRPTW, every customer has an associated time window, and must be served within that window.

The VRP is NP-hard as it includes the well-known traveling salesman problem (TSP) –the problem of finding a minimum length Hamiltonian cycle in a given graph– as a special case [2]. Although the size of the largest VRP instance solved is orders of magnitude less compared to the TSP, significant progress has been made towards solving the VRP and its variants efficiently over the past few decades. In particular, various modeling approaches and solution methods have been proposed. Strong formulations have been derived as a result of the studies on the VRP polyhedron. Exact algorithms based on decomposition techniques have reached far beyond basic branch-and-bound schemes, and powerful heuristic and metaheuristic approaches have been developed which are capable of finding high quality solutions quickly. For a book length treatment of the methods and applications regarding the VRP and its popular variants, we refer to [3].

In any feasible solution of the basic VRP, every customer belongs to exactly one of the routes. Essentially, each route corresponds to a minimum cost Hamiltonian cycle of the complete graph defined over the customer nodes in the route. This is the case in many variants of the VRP. Nevertheless, visiting every customer exactly once may be too restrictive or invalid for some real-life applications, resulting in the emergence of a class of VRPs where this constraint is relaxed. We refer to such problems as the non-Hamiltonian VRPs, and in this thesis, we focus on designing efficient optimization algorithms for different variants of the non-Hamiltonian VRPs.

First, in Chapter 3, we study the SDVRP, which is a relaxation of the CVRP where the demand of each customer can be split and served by multiple vehicles. The SDVRP was introduced with the motivation that significant cost savings can be achieved when split deliveries are allowed. However, allowing split deliveries requires determining the quantity delivered to each customer by each vehicle as well, implying that more variables are needed to formulate the SDVRP compared to the

CVRP. Moreover, to the best of our knowledge, all the existing formulations of the SDVRP use vehicle-indexed variables to model delivery splitting choices, which leads to highly symmetric formulations since any permutation of the vehicle indices leads to an equivalent solution. Thus, the SDVRP is quite challenging especially when the problem size is large. We formulate the SDVRP using vehicle-indexed arc flow variables, and derive a relaxation by aggregating the variables over all vehicle indices to decrease the size of the formulation and to eliminate symmetry. This relaxation may have feasible solutions in which two or more vehicles visiting a certain customer node exchange loads. We develop a polynomial time procedure to identify such customer nodes, if there exists any. Then, we propose two methods, namely patching and node-split, to cut-off the solutions of the relaxation containing such customers. Both methods work by locally extending the relaxation, i.e., by adding new variables and constraints associated with the customer nodes violating the feasibility of the solution. This produces a tighter relaxation but may still yield infeasible SDVRP solutions. Hence, the extension is repeatedly performed until obtaining a relaxation whose optimal solution is feasible for the SDVRP. In addition, we introduce two new variants of the SDVRP and show how to adopt our approaches to solve these variants. We test and compare our algorithms on different sets of benchmark instances as well as on a set of newly generated instances.

Second, in Chapter 4, we study the time constrained maximal covering salesman problem (TCMCSP) in which the goal is to identify a tour visiting a subset of customers so that the demand covered is maximized subject to an upper bound on the tour length. This problem has practical relevance in cases where it is not efficient to visit every demand point separately. Integrating the notion of coverage into a routing scheme; i.e., satisfying the demand of multiple customers through each customer on the route, may provide means to increase system efficiency by utilizing the available resources more effectively. Mobile health facility routing, blood collection, distribution of supplies (food, drinking water, medicine etc.) in the aftermath of a disaster, routing of security patrol cars in rural regions for crime prevention, and routing of unmanned aerial vehicles (UAVs) for information gathering against intruders are among the real-life applications of the TCMCSP. We propose flow and cut formulations for the problem and derive valid inequalities. The cut formulation

involves exponentially many connectivity constraints. Hence, we devise different branch-and-cut schemes to solve it where the violated connectivity constraints and valid inequalities are separated throughout the search tree. We carry out a computational study and demonstrate the effectiveness of the proposed valid inequalities both in terms of strengthening the linear relaxation bounds and in terms of accelerating the solution procedure. Finally, we investigate the impacts of the changes in each problem parameter on the structure of the optimal solutions based on the results of our computations.

Third, in Chapter 5, we study the vehicle routing problem with roaming delivery locations (VRPRDL) motivated by the interest in trunk delivery services. The growth of the e-commerce sector with the ever-increasing push towards online-shopping poses a major supply chain challenge for many companies. Usually, *last-mile delivery*; i.e., the delivery of goods to the consumers is the most expensive and inefficient part of the supply chain. Year-over-year growing sales volumes, huge number of delivery locations, and the aggressive service levels promised to customers drive companies to seek innovative modes of delivery. Among these is the trunk delivery service introduced recently by Amazon, Audi and DHL, in which a customer's order has to be delivered to the trunk of the customer's car during the time that the car is parked at one of the locations in the customer's (known) travel itinerary. We formulate the VRPRDL as a set-partitioning problem and devise an efficient branch-and-price algorithm. To the best of our knowledge, ours is the first solution approach in the literature that solves the VRPRDL optimally. This algorithm can also be used for solving a more general variant of the problem in which a hybrid delivery strategy is considered that allows a delivery to either a customer's home or to the trunk of the customer's car. We perform an extensive computational study to evaluate the effectiveness of the many features of our algorithm and to analyze the benefits of these innovative delivery strategies against a pure home delivery strategy. We demonstrate that the cost savings achieved by the hybrid delivery strategy are in the order of 20% for the instances in our test set.

Finally, in Chapter 6, we consider a dynamic version of the VRPRDL, namely the D-VRPRDL, in which there may be deviations from the original customer itineraries

during the execution stage, and consequently, the planned delivery schedule may become infeasible or suboptimal. We propose an iterative re-optimization framework that solves a series of static problems over the planning horizon. In particular, an initial set of vehicle routes is determined by solving a VRPRDL, and these routes are then re-optimized each time an itinerary update is revealed. Every re-optimization problem is also a VRPRDL, but with an additional set of constraints specifying where the vehicles out for delivery will be originating from when the revised routing plan is put into effect. We use the branch-and-price approach developed for the VRPRDL to solve each static problem. Usually, sufficient time is available to solve the first problem, and thus, it is possible to find optimal solutions or at least solutions of good quality during the planning stage. On the other hand, it may be necessary to solve many re-optimization problems during the execution stage, which requires producing solutions to these problems quickly. Although the computation time allocated to solve a re-optimization problem can be quite limited, usually a large number of pricing iterations have already been performed and many columns have been generated during the solution of the previous problems. Transferring certain parts of this knowledge and using them when solving the re-optimization problem, optimal or sufficiently good solutions may still be obtained. With this motivation, we explore how to utilize the information collected during the previous executions of the branch-and-price algorithm to generate columns more efficiently in solving the subsequent problems. We conduct a preliminary computational analysis and report the results.

Briefly, the focus of this thesis is on developing exact solution approaches for different variants of the non-Hamiltonian VRPs which do not have the restriction that every node in a given graph should be visited exactly once. We consider three cases regarding this restriction. In the SDVRP, multiple vehicles are allowed to visit a node, so the number of visits to a node is at least one. In the TCMCSP, some nodes may not be visited at all due to the tour length constraint, implying that the number of visits to a node is at most one. The VRPRDL generalizes the VRPTW by imposing exactly one visit restriction to node clusters instead of individual nodes. Typical modeling approaches for the VRPs are flow, cut and route based. We consider all these approaches within this thesis and develop algorithms

based on iterative extensions of a relaxation (i.e. patching and node-split algorithms for the SDVRP), branch-and-cut, and branch-and-price, which are among the leading optimization methods for the VRP and its variants.

Chapter 2

Literature Review

In the following sections, we provide an overview of the related literature for the problems studied in this thesis.

2.1 Split delivery vehicle routing problem

The SDVRP is a relaxation of the CVRP, where the demand of a customer can be split and delivered by multiple vehicles. It is formally defined in [4] with the motivation that permitting split deliveries can result in considerable transportation cost savings. The problem is shown to be NP-hard in [5], and despite being a relaxation of the classical CVRP, it is not easier to tackle as the amounts to be delivered to each customer by each vehicle is also unknown.

In the past 28 years, several different exact and heuristic solution approaches as well as complexity-related analyses are proposed, and real-life problems are modeled and solved as variants of SDVRP. The first heuristic method is a two-stage local search algorithm developed in [4]. The subsequent studies [6]– [17] focus on hybrid

methods and metaheuristics. A comprehensive discussion on heuristics is provided in [18]. Various other heuristic methods exist in the literature that are used for solving more special routing problems incorporating the split delivery option within their framework such as [19]– [24].

The first exact algorithm to solve the SDVRP is a constraint relaxation branch-and-bound algorithm presented in [25]. The problem is formulated as an integer linear program and effective valid inequalities are derived. Branch-and-bound is used for achieving integrality, while the valid inequalities are added to cut-off the solutions that are inadmissible for the SDVRP.

In [26], the problem of scheduling helicopter flights to exchange crews is modeled as an SDVRP. The authors propose an integer linear programming formulation in which all feasible flight schedules are enumerated in advance, and solve its linear relaxation by means of column generation. A similar column generation approach is suggested in [10] for the SDVRP with large demands. The undirected version of SDVRP is considered in [27]. The authors provide an integer programming model and a relaxation of the SDVRP, and prove that all constraints in this relaxation are facet-defining for the convex hull of the incidence vectors of the SDVRP solutions. Computational experiments with 25 instances indicate that their cutting plane approach can solve instances with up to 50 customers optimally. A dynamic program with finite state and action spaces is given in [28]. Test instances containing at most 9 customers are solved with this method. A two-stage algorithm with valid inequalities (TSVI) is introduced in [29]. The first stage creates clusters while respecting vehicle capacity restrictions, and establishes a lower bound on the optimal cost. The second stage computes an upper bound by solving a TSP on each cluster. TSVI iteratively executes these steps until the lower bound in the first stage and the upper bound from the second stage are equal and solve instances with up to 21 customers. In [30] and [31], extended formulations are provided to compute lower bounds for the SDVRP. In both studies, Dantzig-Wolfe decomposition principle is employed and column generation procedures are implemented to solve the resulting master problems. Computational experiments show that [31] can in general produce tighter lower bounds than [30].

The first branch-and-price-and-cut method for the SDVRP is developed by [32] applying a similar decomposition to [33], who proposes a branch-and-price-and-cut technique for the SDVRP with time windows. The algorithm is tested on a large set of benchmark instances for both limited and unlimited vehicle fleet cases. The majority of the best available lower bounds and some of the best available upper bounds are improved. Although one instance with 144 customers is solved to optimality, the second largest instance optimized contains 48 customers. Two exact branch-and-cut solution methodologies are given in [34] where the optimality of 17 instances in the literature and a new instance involving 100 customers is established.

2.2 Time constrained maximal covering salesman problem

The first problem incorporating the coverage concept into a routing scheme is the covering salesman problem (CSP). CSP is the problem of identifying a minimum length Hamiltonian tour over a subset of vertices in a way that every vertex not on the tour lies within a certain distance of some visited vertex. The CSP is formally introduced in [35] where a heuristic algorithm is proposed to solve the problem. Later, the geometric version of the CSP is studied in [36] and polynomial time approximation algorithms are presented with a bounded error ratio regarding the optimal tour length.

Two multi-objective variants of the CSP are considered in [37]. These are the median tour problem (MTP) and the maximal covering tour problem (MCTP) where the tour should visit a predetermined number of vertices and the objectives are: (1) minimization of the tour length and (2) maximization of the accessibility to the tour for the vertices that are not visited. A heuristic approach is suggested to approximate the frontier of the efficient solutions.

A generalization of the CSP is studied in [38] and [16] where an additional cost

is incurred for every node visited by the tour and each node is associated with a weighted demand representing the minimum number of times it has to be covered. Another generalization of the CSP, called the generalized covering traveling salesman problem (GCTSP), is presented in [39]. In the GCTSP, one aims to find a minimum length tour passing through a subset of facilities while covering at least a predetermined number of customers. Node-based and flow-based formulations are presented and two metaheuristic approaches are developed for the problem. The TCMCSP is introduced in [40], where the goal is to maximize the number of covered customers with an upper bound on the total traveling time. In a sense, it is complementary to the GCTSP.

A very popular generalization of the CSP is the covering tour problem (CTP) introduced in [41]. Given an undirected graph $G = (V \cup W, E)$, the CTP is the problem of identifying a minimum length Hamiltonian tour in which the vertices in $T \subset V$ must be on the tour while the remaining vertices in V may or may not be visited, and the vertices in W should be covered without being visited.

The problem of planning mobile healthcare facilities in Suhum District of Ghana is modeled as the CTP in [42] and solved with the algorithm developed in [41]. In [43], a GRASP is devised for solving a generalization of the CTP in which the vertices in W can also be visited. A two-commodity flow formulation and three scatter search methods for the CTP are presented in [44]. Several other heuristics are proposed in [45].

The multi-vehicle variant of the CTP (m -CTP) is introduced in [46]. For each tour, there is an upper bound on its length and an upper bound on the number of vertices visited. The m -CTP is formulated as an integer linear program using vehicle flow variables and heuristic algorithms are developed. A covering tour perspective is adopted in [47] to tackle the problem of locating satellite distribution centers to supply humanitarian aid over a disaster area. The problem of planning routes for routine patrol cars is also modeled as the m -CTP in [48].

The CTP is investigated in a multi-objective setting as well. In [49], the bi-objective CTP (BOCTP) is introduced, and a multi-objective evolutionary algorithm is proposed. A post-natural-disaster related problem is addressed in [50] in which part of the infrastructure in the region affected by the disaster is destroyed. The problem of supplying food, medicine and shelter over the affected region is considered as a multi-objective CTP and heuristics are presented to solve the problem. A variant of the BOCTP with stochastic demands is introduced in [51] and modeled as a two-stage stochastic program with recourse, which is solved using an epsilon-constraint approach involving branch-and-cut.

The TCMCSP is also related to traveling salesman problems with profits, which are classified into three categories in [52] based on their objectives. They are (1) maximizing profit under a distance constraint, (2) minimizing distance under a profit constraint and (3) a combination of distance minimization and profit maximization. The TCMCSP is closest to the problems in class (1), which also contains the orienteering problem (OP) introduced in [53]. In the OP, also known as the selective traveling salesman problem [54] or the maximum collection problem [55], every vertex is associated with a profit and the objective is to find a tour with maximum profit subject to a time restriction. The OP is a special case of the TCMCSP with $r = 0$; that is, the demand of a vertex is covered only if it is visited. We refer the interested reader to [56] for a recent survey regarding OP.

2.3 Vehicle routing problem with roaming delivery locations

Motivated from trunk delivery applications, the VRPRDL has recently been introduced in [57], who developed various construction and improvement heuristics for the problem. Through a computational study, the authors demonstrate the benefits of this innovative mode of delivery especially when used in combination with home delivery.

The VRPRDL combines two well-studied problems, namely, the VRPTW and the GVRP. The VRPTW is the problem of determining an optimal set of delivery routes serving the demand of a set of customers within their respective time windows. There is a vast body of literature on the VRPTW and its variants (see, for example, [58–66]). The GVRP was introduced by [67] and it is another generalization of the VRP in which the set of delivery locations is partitioned into clusters and exactly one location from each cluster has to be visited in a solution. Despite its relatively recent introduction, the GVRP has already attracted the attention of many researchers, in part because it has many real-life applications [68–74].

The integration of the features of these two problems leads to the generalized vehicle routing problem with time windows (GVRPTW). To the best of our knowledge, the first study on the GVRPTW is due to [75], who present an incremental tabu search algorithm to solve the problem. The VRPRDL can be seen as a special case of the GVRPTW in which the sets of delivery locations for the customers form the clusters. However, the time windows exhibit a special structure, as the time windows of the locations in a cluster, i.e., the time windows of the delivery locations for a single customer, are non-overlapping. To the best of our knowledge, prior to our work, no exact solution methods exist in the literature to solve the VRPRDL, or more generally, the GVRPTW.

2.4 Dynamic vehicle routing problem with roaming delivery locations

There is no literature on the D-VRPRDL, since it is introduced for the first time in this thesis. Therefore, we provide a brief review about dynamic VRPs. The literature on dynamic vehicle routing dates back to 40 years ago when a single vehicle dial-a-ride problem with dynamically arriving trip requests was presented in [76]. Later in [77], an immediate request version of the same problem was introduced in which the vehicle’s route should be re-planned immediately upon the arrival of a new trip

request so that the customer making the request is served as early as possible.

After [77], only a few papers have been published on dynamic vehicle routing until the late 1990s. However, there has been an increasing interest towards the subject starting with the launch of the Global Positioning System (GPS) in 1996, and the technological developments in navigation and positioning systems ever since. Moreover, the widespread use of Geographic Information Systems, Intelligent Transportation Systems, and smart devices with GPS tracking technology, combined with the advances in computing power and data processing capabilities, enable companies to monitor and manage their vehicle fleet in real-time more and more effectively. As a result, dynamic vehicle routing problems have attracted the attention of many researchers for the past 20 years.

Most studies in the dynamic vehicle routing literature focus on the arrival of customer orders during the operation of the planned vehicle routes as the source of dynamism, see for example [78–84]. Although fewer, there are also studies considering two other dynamic aspects of real-life vehicle routing problems, namely travel times [85–89] and vehicle breakdowns [90–92].

The existing solution methods on dynamic and deterministic vehicle routing problems are mostly based on periodic re-optimization either by dividing the planning horizon into fixed decision epochs (also known as time slices) or as soon as a certain number of changes occurs in the input data. Each re-optimization problem corresponds to a static problem defined based on the currently available input data. In order to obtain and start implementing an updated routing plan as soon as possible, it is critical to solve each static problem quickly. To this end, most approaches in the literature rely on heuristics as in [78, 79, 81, 86, 90, 91, 93–96].

For a comprehensive review and a detailed taxonomy of the dynamic vehicle routing problems and solution approaches, we refer the interested reader to the papers [97] and [98], and the book chapter [99].

Chapter 3

New Exact Solution Approaches for the Split Delivery Vehicle Routing Problem

The SDVRP is a relaxation of the classical CVRP, where the demand of a customer can be split and delivered by multiple vehicles. The task is to find a set of least cost delivery routes for a vehicle fleet starting and ending at the depot so that each customer belongs to at least one route, the demand of every customer is fully satisfied, and the total demand assigned to any (vehicle) route does not exceed the vehicle capacity.

In this chapter, we propose a new arc flow formulation for the SDVRP that uses variables with vehicle indices. To decrease the size and to eliminate the symmetry, we aggregate the variables over all vehicles. This resulting relaxation is similar to one of the relaxations in [34]. We give a family of valid inequalities that includes the generalized capacity inequalities of [27] as a special case and show that these inequalities are not sufficient to obtain a formulation. To eliminate solutions of the

relaxation infeasible for SDVRP, we propose to locally extend the relaxation either by adding vehicle-indexed variables for some customer nodes or by node splitting. Our computational experiments reveal that iterating for an optimal solution of the SDVRP with our methods can be performed effectively as long as the relaxation can be solved effectively.

Though split deliveries save costs, they come at the expense of additional handling time. We introduce the problem SDVRP with restricted number of splits to the literature. We extend our exact solution methodologies to solve this variant. Against intuition, it is not any easier to solve this restricted version of the SDVRP. Another variation we handle is the SDVRP with open routes where the depot return requirement is relaxed. Though some theoretical results no longer are valid for this variation, our computational experiments reveal favorable results.

The rest of this chapter is organized as follows. In Section 3.1, the arc flow formulation and its relaxed version are presented along with some simplifications for the relaxation. We propose a family of valid inequalities that generalize the generalized capacity inequalities of [27] and give an example where these inequalities cannot cut-off the optimal solution of the relaxation that is infeasible for the SDVRP. In Section 3.2, the methods to eliminate the optimal solutions of the relaxed model that are not feasible for the SDVRP as well as the exact solution algorithms are introduced. The results of the computational experiments are given in Section 3.3. Section 3.4 is reserved for the two extensions of SDVRP along with their computational results. Finally, Section 3.5 provides a summary of our findings.

3.1 Formulation, relaxation and valid inequalities

Let $G = (N, A)$ be a directed complete graph with the set of nodes $N = \{0, 1, \dots, n\}$ and the set of arcs $A = \{a = (i, j) : i, j \in N, i \neq j\}$. Suppose that the depot is located at node 0 and each node $i \in N \setminus \{0\}$ represents a customer location. There are m identical vehicles available at the depot to serve the customers, each having a

capacity of Q units. We define $K = \{1, \dots, m\}$. The cost of traversing arc $a \in A$ is c_a and the demand of customer $i \in N \setminus \{0\}$ is $0 < d_i \leq Q$. We assume that the costs are non-negative and they satisfy the triangle inequality.

3.1.1 An exact flow based formulation with vehicle indices

We first present a flow based formulation with vehicle indices. We use the following decision variables:

- $y_a^k = \begin{cases} 1 & \text{if vehicle } k \in K \text{ travels on arc } a \in A, \\ 0 & \text{otherwise,} \end{cases}$
- $g_a^k =$ the amount of flow carried on arc $a \in A$ by vehicle $k \in K$,
- $w_i^k =$ fraction of the demand of customer $i \in N \setminus \{0\}$ delivered by vehicle $k \in K$.

For a given set $S \subset N$, let $\delta^-(S)$ denote the set of arcs (i, j) with $i \in N \setminus S$ and $j \in S$ and $\delta^+(S)$ denote the set of arcs (i, j) with $i \in S$ and $j \in N \setminus S$. We use $\delta^-(i)$ and $\delta^+(i)$ for $\delta^-(\{i\})$ and $\delta^+(\{i\})$. For a vector $\alpha \in R^{|U|}$ and $U' \subseteq U$, we let $\alpha(U') = \sum_{u \in U'} \alpha_u$.

(SDVRP)

$$\min \sum_{a \in A} \sum_{k \in K} c_a y_a^k \tag{3.1}$$

$$g^k(\delta^-(i)) - g^k(\delta^+(i)) = d_i w_i^k \quad i \in N \setminus \{0\}, k \in K, \tag{3.2}$$

$$y^k(\delta^-(i)) - y^k(\delta^+(i)) = 0 \quad i \in N \setminus \{0\}, k \in K, \tag{3.3}$$

$$y^k(\delta^+(0)) = 1 \quad k \in K, \tag{3.4}$$

$$\sum_{k \in K} w_i^k = 1 \quad i \in N \setminus \{0\}, \tag{3.5}$$

Table 3.1: Some results with the vehicle-indexed model

Instance	Number of nodes	Number of vehicles	Lower bound	Upper bound	Gap (%)	Time (sec)	Nodes in b&c tree
eil22	22	4	375	375	0	108.57	115403
eil23	23	3	569	569	0	9.87	14475
eil30	30	3	510	510	0	2149.89	1065899
eil33	33	4	819.64	835	1.84	7200	1364276

$$g_a^k \leq Qy_a^k \quad a \in A, k \in K, \quad (3.6)$$

$$w_i^k \geq 0 \quad i \in N \setminus \{0\}, k \in K, \quad (3.7)$$

$$g_a^k \geq 0 \quad a \in A, k \in K, \quad (3.8)$$

$$y_a^k \in \{0, 1\} \quad a \in A, k \in K. \quad (3.9)$$

The objective function (3.1) aims to minimize the global transportation cost. Constraints (3.2) and (3.3) require commodity flow and vehicle flow conservation for every customer and for every vehicle. Constraints (3.4) force all the vehicles to leave the depot for service, and (3.5) guarantee that the demand of each customer is fully satisfied. Constraints (3.6) are the coupling constraints ensuring that the flow on an arc carried by a vehicle does not exceed the vehicle capacity. Finally, (3.7)–(3.9) specify variable restrictions.

The formulation given in (3.1)–(3.9) contains $O(n^2m)$ variables and $O(n^2m)$ constraints. Due to its large size and due to the symmetry induced by the homogeneous fleet of vehicles, it can be solved to optimality for small size problems. Table 3.1 shows our results with a time bound of 7200 seconds regarding the four smallest instances in [27]. It can be observed that the number of nodes in the branch-and-cut tree is quite large even for these instances.

3.1.2 A flow based relaxation

In this section, we present a relaxed model obtained by aggregating the decision variables over all vehicles, i.e., by letting $f_a = \sum_{k \in K} g_a^k$ and $x_a = \sum_{k \in K} y_a^k$ for every arc $a \in A$. Our aim is to decrease the size of the vehicle-indexed formulation and to eliminate symmetry. The relaxed model is as follows:

(R-SDVRP)

$$\min \sum_{a \in A} c_a x_a \quad (3.10)$$

$$\text{s.t. } f(\delta^-(i)) - f(\delta^+(i)) = d_i \quad i \in N \setminus \{0\}, \quad (3.11)$$

$$x(\delta^-(i)) - x(\delta^+(i)) = 0 \quad i \in N \setminus \{0\}, \quad (3.12)$$

$$x(\delta^+(0)) = m, \quad (3.13)$$

$$f_a \leq Q x_a \quad a \in A, \quad (3.14)$$

$$f_a \geq 0 \quad a \in A, \quad (3.15)$$

$$x_a \in \mathbb{Z}_+ \quad a \in A. \quad (3.16)$$

Similar to the exact model, the objective is to minimize the total cost of transportation. Constraints (3.11) ensure that the demand of every customer is fulfilled. Vehicle flow conservation is enforced by constraints (3.12) and constraint (3.13) guarantees that exactly m vehicles are dispatched from the depot for service. Constraints (3.14) relate variables x_a and f_a based on the vehicle capacity. Domain restrictions on the decision variables are imposed by (3.15) and (3.16).

3.1.3 An optimality property

A *k-split cycle* is defined in [4] as a subgraph on a set of customers $i_1, \dots, i_k \subset N \setminus \{0\}$ with $k \geq 2$ in which there exist $1 \leq h \leq k$ vehicle routes such that i_t and i_{t+1} are on the same route for every $t = 1 \dots, k-1$, and that i_1 and i_k are on the same route. Accordingly, the authors establish the *k-split cycle property*, which guarantees the

existence of an optimal SDVRP solution free of k -split cycles for any $k \geq 2$ under the condition that the cost matrix satisfies the triangle inequality. Based on the k -split cycle property, we can impose binary requirements on the x_a variables for arcs a with customers at both endpoints, i.e., $a \in A \setminus (\delta^-(0) \cup \delta^+(0))$. This helps in reducing computation times. In Proposition 3.1.1, we prove that if the costs are symmetric, then we can also restrict the x variables associated either with the arcs originating from the depot or with those ending at the depot to take 0-1 values. Based on initial computational trials, we prefer to apply the restriction to the arcs emanating from the depot.

Proposition 3.1.1 *If the costs are symmetric and if they satisfy the triangle inequality, then there exists an optimal SDVRP solution x for which $x_a \in \{0, 1\}$ for all $a \in A \setminus \delta^-(0)$.*

Proof. First note that since the cost matrix is symmetric, one can reverse the direction of any route and attain an alternative optimal solution. Also, there exists an alternative optimal solution in which a customer on a dedicated route, i.e., a route with a single customer, is visited only once. To show this, assume that i is a customer who is visited by routes C_1 and C_2 where C_1 is a dedicated route. Since $d_i \leq Q$ and the costs satisfy triangle inequality, it is possible to attain another solution with the same cost by excluding i from route C_2 .

Assume that x is an optimal solution to a given SDVRP instance that is free of k -split cycles (for any $k \geq 2$) and that customers receiving dedicated service are not split nodes. If $x_{0i} \leq 1$ for every customer i , then we are done. Otherwise, we shall iteratively construct another optimal solution satisfying the proposed condition. Take a customer i for which $x_{0i} = \mu$, where $\mu \geq 2$. Pick any one of these μ routes, say C_1 , and let j_1 be the last customer on this route (where i is the first customer). Note that $j_1 \neq i$ otherwise customer i would be a split node receiving dedicated service. If $x_{0j_1} = 0$, then reversing the direction of route C_1 will result in an alternative optimal solution with x_{0i} decremented and no x_a for $a \in \delta^+(0)$ incremented beyond value 1. Otherwise, let C_1, \dots, C_l be a sequence of routes such that for any two consecutive

routes C_t and C_{t+1} for $t = 1, \dots, l-1$, the last customer of C_t and the first customer of C_{t+1} are identical. Moreover, let l be the largest possible such number. Consider any two routes C_p and C_q such that $q > p + 1$. These two routes cannot intersect, otherwise routes C_p, C_{p+1}, \dots, C_q will constitute a $(q-p+1)$ -split cycle and we know that the optimal solution is free of such cycles. In particular, this implies that if j_l is the last customer in route C_l , then $x_{0j_l} = 0$, otherwise we violate either the fact that l is not the largest possible consecutive route number or that there is no k -split cycle. Now, reversing all the routes C_1, \dots, C_l will result in an optimal solution with x_{0i} decremented and no x_a for $a \in \delta^+(0)$ incremented beyond value 1. Repeating this procedure for every customer i with $x_{0i} \geq 2$, an alternative optimal solution can be attained in which $x_a \in \{0, 1\}$ for all $a \in A \setminus \delta^-(0)$. \square

3.1.4 Comparison with existing relaxations

Next, we compare our relaxed model to other relaxed models in the literature. A similar model to R-SDVRP is given by [34]. Different from our model, [34] do not force all vehicles to be used. They use additional variables to keep the number of visits to each node and put upper bounds on these variables. Using the k -split cycle property, they restrict the variables associated with the arcs between customer pairs to be 0-1. In addition, they force the flow on return arcs to the depot to be zero.

Note that if we project out the flow variables in R-SDVRP, we obtain the fractional capacity inequalities

$$x(\delta^-(S)) \geq \frac{d(S)}{Q} \tag{3.17}$$

for all $S \subseteq N \setminus \{0\}$ (see [100] and [101] for more projection results). Hence R-SDVRP is equivalent to a directed version of the relaxation used by [27]. These authors depict a solution of their relaxation for the instance *eil30* which is not feasible for SDVRP. In Figure 3.1, we depict the solution found by solving our relaxation. We obtain the same solution, but we also have the flow values on the arcs. We report these values for the arcs adjacent to node 18. Three vehicles visit node 18, one of which is empty

upon arrival while the other two are not. The empty vehicle returns to the depot after passing through node 18, while the nonempty vehicles arrive at node 18 with 4500 and 625 units of load and leave the node with 3175 and 1800 units of load, respectively. This can only be possible if 1175 units of load is unloaded from the first vehicle and loaded on the second vehicle while the vehicles are at node 18. This is not admissible for the SDVRP.

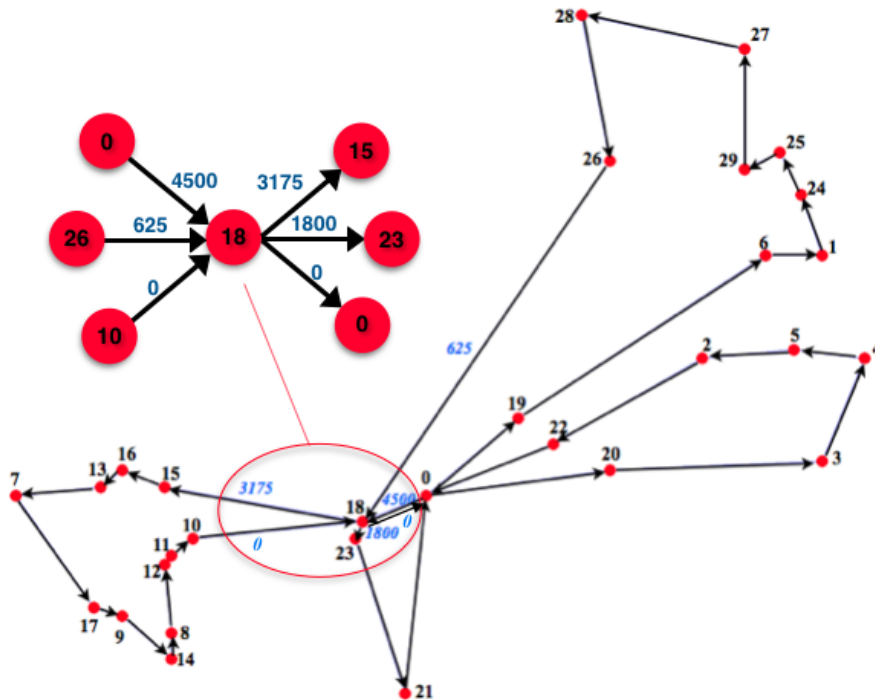


Figure 3.1: The optimal solution of R-SDVRP for *eil30*

3.1.5 Framed capacity inequalities

In [27], the authors propose to cut-off the infeasible solution given in Figure 3.1 using a valid inequality. In this section, we present a family of valid inequalities that generalizes the inequalities used by [27]. These inequalities are called “framed

capacity inequalities” and their undirected variants are proposed for the CVRP (see, e.g., the review by [102]).

Proposition 3.1.2 *Let $H \subseteq N \setminus \{0\}$ and S_1, \dots, S_t be disjoint non-empty subsets of H . Define $b(S_1, \dots, S_t)$ to be the optimal value of the bin packing problem with items $1, \dots, t$ of size $d(S_1), \dots, d(S_t)$ (if there exists u with $d(S_u) > Q$, then as done by [27], we consider the demand of set S_u to be $d(S_u) - \lfloor \frac{d(S_u)}{Q} \rfloor Q$ in the bin packing problem and add $\lfloor \frac{d(S_u)}{Q} \rfloor$ to the bin packing value). The framed capacity inequality*

$$x(\delta^-(H)) + \sum_{u=1}^t x(\delta^-(S_u)) \geq \sum_{u=1}^t \left\lfloor \frac{d(S_u)}{Q} \right\rfloor + b(S_1, \dots, S_t) \quad (3.18)$$

is valid for the feasible set of SDVRP.

Proof. If $x(\delta^-(S_u)) = \lfloor \frac{d(S_u)}{Q} \rfloor$ for all $u = 1, \dots, t$, then we need at least $b(S_1, \dots, S_t)$ vehicles to enter set H to satisfy the demand of $\cup_{u=1}^t S_u$. Hence $x(\delta^-(H)) \geq b(S_1, \dots, S_t)$. Since each split in S_u can reduce the number of required vehicles by at most 1, the result follows. \square

Note that, for the CVRP, the bin packing value is computed using all customers in H . In our case, if $b(S_1, \dots, S_t) \leq \lfloor \frac{d(H)}{Q} \rfloor$, then the inequality is dominated by the sum of rounded capacity inequalities $x(\delta^-(H)) \geq \lfloor \frac{d(H)}{Q} \rfloor$ and $x(\delta^-(S_u)) \geq \lfloor \frac{d(S_u)}{Q} \rfloor$ over all $u = 1, \dots, t$. If $b(S_1, \dots, S_t) > \lfloor \frac{d(H)}{Q} \rfloor$, considering all customers of H by letting splits for the ones in $H \setminus \cup_{u=1}^t S_u$ does not change the result of the bin packing problem since $b(S_1, \dots, S_t)Q > d(H)$.

The inequalities used by [27] are special cases of inequalities (3.18) with $H = V \setminus \{0\}$ and consequently $x(\delta^-(H)) = m$.

Next, we show with an example that even if we include all framed capacity inequalities into our relaxed model, the resulting model is still a relaxation and may

have optimal solutions that are not feasible for the SDVRP. In other words, there exist optimal R-SDVRP solutions that are not admissible for the SDVRP, yet cannot be eliminated using any framed capacity inequality. Such a solution is depicted in Figure 3.2 along with the cost matrix associated with the problem instance. The demands are $d_1 = 4$, $d_2 = 2$, $d_3 = 6$, $d_4 = 15$ and $d_5 = 1$. There are two vehicles, each with a capacity of 15 units. The number on each arc corresponds to its flow value. Notice that a load exchange takes place between the vehicles at node 5. The total cost associated with this solution is 60, while the optimal SDVRP solution has cost 61. Therefore, there does not exist an optimal SDVRP solution using the edges in this R-SDVRP solution.

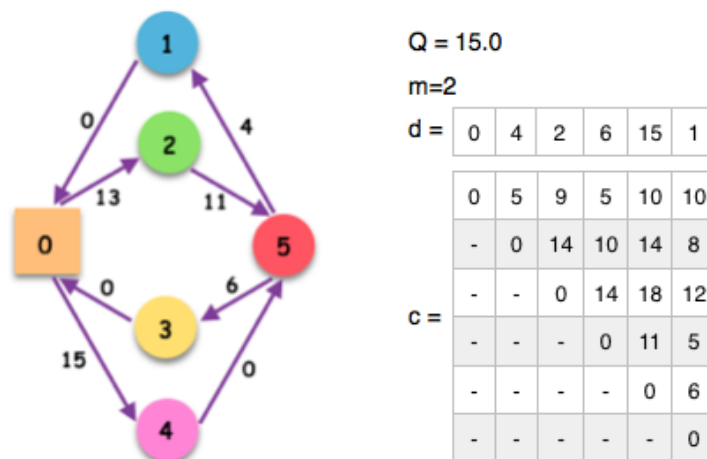


Figure 3.2: An optimal solution to R-SDVRP that cannot be cut-off by any framed capacity inequality

First note that the bin packing value cannot be larger than two for all possible subsets H and S_1, \dots, S_t . If $x(\delta^-(H)) \geq 2$, then as $b(S_1, \dots, S_t) \leq 2$ and $x(\delta^-(S_u)) \geq \left\lceil \frac{d(S_u)}{Q} \right\rceil$ for $u = 1 \dots, t$, inequality (3.18) is satisfied. Now for $x(\delta^-(H)) = 1$, we need $H \subset N \setminus \{0, 5\}$ and $|H| = 1$. Then $S_1 = H$ or $S_1 = \emptyset$ and accordingly the bin packing value $b(S_1)$ is 1 or 0 and the inequality is again satisfied. Hence, the framed capacity inequalities fail to omit the solution in this example from the feasible set of the R-SDVRP.

3.1.6 Rounded capacity and cutset inequalities

To conclude this section, we describe two classes of valid inequalities that are employed for strengthening our relaxation. Let \mathcal{Y} be the feasible set of the R-SDVRP and $S \subseteq N \setminus \{0\}$. The rounded capacity inequality

$$x(\delta^-(S)) \geq \left\lceil \frac{d(S)}{Q} \right\rceil \quad (3.19)$$

is valid for \mathcal{Y} .

Now consider the relaxation

$$f(\delta^-(S)) - f(\delta^+(S)) = d(S), \quad (3.20)$$

$$0 \leq f_a \leq Qx_a \quad a \in \delta^-(S) \cup \delta^+(S), \quad (3.21)$$

$$x_a \in \mathbb{Z}_+ \quad a \in \delta^-(S) \cup \delta^+(S). \quad (3.22)$$

The convex hull of the solutions of the above set is defined by trivial inequalities and the following cutset inequalities (see [103]). Let $A^- \subseteq \delta^-(S)$, $A^+ \subseteq \delta^+(S)$, $\eta = \left\lceil \frac{d(S)}{Q} \right\rceil$ and $r = d(S) - \left\lfloor \frac{d(S)}{Q} \right\rfloor Q$. The cutset inequality is

$$f(\delta^-(S) \setminus A^-) + rx(A^-) + (Q - r)x(A^+) - f(A^+) \geq r\eta \quad (3.23)$$

and is valid for \mathcal{Y} . If $A^- = \delta^-(S)$ and $A^+ = \emptyset$, the cutset inequality reduces to the rounded capacity inequality.

3.2 New exact methods for the SDVRP

Two novel iterative algorithms are devised for solving the SDVRP to optimality. Essentially, the mechanism behind both algorithms is the same. First, an optimal solution (f^*, x^*) of the R-SDVRP is obtained. If the solution (f^*, x^*) is feasible

for SDVRP, then it is also an optimal SDVRP solution. Otherwise, new variables and constraints are added to the formulation R-SDVRP such that when the new variables are projected out, some portion of \mathcal{Y} , including the vector (f^*, x^*) is cut-off. The relaxation is solved again over a more constrained region. This process continues iteratively until an optimal SDVRP solution is found. The two methods are distinguished by the routines they use for eliminating the solution (f^*, x^*) at every iteration. Before elaborating more on these routines, we describe what we refer to as the *regularity property*.

Definition (Regularity Property): A feasible solution of R-SDVRP possesses the regularity property; or equivalently, it is called regular, if for any node $i \in N \setminus \{0\}$, the following holds:

$$f^-(i, j) \geq f^+(i, j) \text{ for all } j = 1, \dots, i_n,$$

where i_n is the number of vehicles passing through node i , $f^-(i, j)$ and $f^+(i, j)$ are the amounts of the j^{th} largest incoming and outgoing flows associated with the node i , respectively.

Note that the regularity of an R-SDVRP solution can be established in $O(m^2 \log m)$ time since there can be at most $m - 1$ split nodes (see [104]), and for each one, ordering the incoming and outgoing flow values takes at most $O(m \log m)$ time. For a node i for which $x_{i0} > 1$, f_{i0} should be decomposed into x_{i0} flow values having the potential of satisfying the regularity property which can easily be handled within the same time complexity.

[34] prove that if an optimal solution of the R-SDVRP has the regularity property then it solves SDVRP optimally. This result establishes an equivalence between the regular R-SDVRP solutions and the feasible SDVRP solutions. Given a solution to the R-SDVRP, one can check in polynomial time whether it is regular and thus feasible for the SDVRP. However, deciding on the regularity of an R-SDVRP solution is different from checking whether a given solution x is feasible for the SDVRP, which is shown to be NP-complete by [27].

We can construct an optimal SDVRP solution from an optimal regular solution (f, x) of the R-SDVRP in the following way. Consider the arcs in the corresponding

support graph; that is, the arcs $a \in A$ with $x_a \geq 1$. We shall apply depth first search traversals in the support graph in order to construct m viable routes. Start each route with an arc emanating from the depot and perform depth first search making sure at every node among the potential outgoing arcs, the one having the largest flow value less than or equal to the flow value of the used incoming arc is selected. For a node i such that $x_{i0} \geq 2$, such a route extension is not that obvious. Suppose our traversal enters such a node i using arc (j, i) . We shall split arc $(i, 0)$ into x_{i0} identical arcs. The route will be completed by choosing one of these arcs with flow value as $\min(f_{ji}, f_{i0})$. Now, take out this constructed route from the support graph, update the demands and the flow values on multiple arcs entering the depot and repeat the same steps for another route. Note that our arc selection preserves regularity and after m steps we construct an optimal solution for the SDVRP. Since the support graph has at most nm arcs and since each arc can be visited at most m times during our traversals, the complexity of this algorithm is $O(nm^2)$.

In the following subsections, the details of the exact solution methods we propose are discussed.

3.2.1 Patching algorithm

Even though our vehicle-indexed flow formulation is not computationally efficient, it may be reasonable to use vehicle indices, at least for some arcs, to be able to find an optimal SDVRP solution by solving a relaxation. The patching algorithm is based on the idea of locally extending the R-SDVRP formulation with vehicle-indexed variables when needed. More precisely, at each iteration of the algorithm, a node violating the regularity property is identified and vehicle-indexed variables are introduced associated with the arcs incident to this node. These variables allow us to formulate the constraints necessary to enforce the regularity at this node. The steps of the patching algorithm are given below.

Step 0. Initialization: Solve the R-SDVRP, and let (\bar{f}, \bar{x}) denote the optimal solution found. Set current solution to (\bar{f}, \bar{x}) .

Step 1. Check the regularity of the current solution. If it is regular, stop. The current solution is optimal for the SDVRP.

Step 2. Let $\bar{G} = (N, \bar{A})$ represent the support graph corresponding to the current solution; i.e., the graph induced by the arcs (i, j) for which $\bar{x}_{ij} \geq 1$. Update \bar{G} by adding it the arcs (j, i) for all $(i, j) \in \bar{A}$, and solve the exact (vehicle-indexed) SDVRP formulation on the updated graph \bar{G} . If a feasible solution exists, stop; it is optimal for the SDVRP.

Step 3. Among the nodes violating the regularity of the current solution, select the first one encountered during the regularity check. Denote this node by i^* . Add vehicle-indexed variables for the arcs in $\delta^-(i^*) \cup \delta^+(i^*)$, and introduce the following set of constraints to the model solved in the previous iteration.

$$g^k(\delta^-(i^*)) - g^k(\delta^+(i^*)) \geq 0 \quad k \in K, \quad (3.24)$$

$$y^k(\delta^-(i^*)) - y^k(\delta^+(i^*)) = 0 \quad k \in K, \quad (3.25)$$

$$y^k(\delta^-(i^*)) \leq 1 \quad k \in K, \quad (3.26)$$

$$\sum_{k \in K} g_a^k = f_a \quad a \in \delta^-(i^*) \cup \delta^+(i^*), \quad (3.27)$$

$$\sum_{k \in K} y_a^k = x_a \quad a \in \delta^-(i^*) \cup \delta^+(i^*), \quad (3.28)$$

$$g_a^k \leq Q y_a^k \quad a \in \delta^-(i^*) \cup \delta^+(i^*), k \in K, \quad (3.29)$$

$$g_a^k \geq 0, y_a^k \in \{0, 1\} \quad a \in \delta^-(i^*) \cup \delta^+(i^*), k \in K. \quad (3.30)$$

Constraints (3.24) force the regularity property at node i^* and constraints (3.25) ensure that vehicle flow is conserved at this node. Constraints (3.26) prevent node i^* from being visited more than once by the same vehicle. The vehicle-indexed variables g_a^k and y_a^k are linked to the original decision variables f_a and x_a with the constraints (3.27) and (3.28), respectively. Constraints (3.29) set the upper bounds on the flows for the arcs in $\delta^-(i^*) \cup \delta^+(i^*)$. Finally, non-negativity and binary requirements for the new variables are given by (3.30).

Step 4. Solve the modified model and update the current solution accordingly. Return to Step 1.

The patching algorithm guarantees convergence to an optimal solution of the SDVRP by fixing the regularity violation for at least one node from one iteration to another. Adding vehicle-indexed variables and regularity-related restrictions for a node makes it possible to distinguish between different vehicles visiting the node and prevents load exchanges. Although the R-SDVRP grows in terms of the number of variables and constraints with the increasing number of iterations, as our computational results in Section 3.3 will attest to, this algorithm is capable of reaching the optimum much faster compared to the vehicle-indexed model, which can be seen by comparing the results in Table 3.1 to those that will be provided in Tables 3.2 and 3.3.

3.2.2 Node-split algorithm

The patching algorithm adds vehicle indexed variables for all vehicles at a node violating regularity. In most practical cases, the demand of a node is split among two or three vehicles. Hence, by patching, we may use unnecessary variables and constraints. The node-split method provides a way to make a distinction between the vehicles visiting a certain node without using vehicle-indexed variables. It is similar to the patching algorithm in the following respects: (1) the R-SDVRP is solved at the initialization step, (2) an extended version of the R-SDVRP obtained by adding new variables and constraints is solved at each iteration, (3) regularity violations are detected and eliminated iteratively until an optimal SDVRP solution is obtained. However, it differs from the patching algorithm in terms of the approach adopted for enforcing the regularity property at a violating node.

The idea of the node-split algorithm is to create duplicates of the nodes violating regularity and to constrain the net incoming flow to each such node and every one of its duplicates to take non-negative values. Duplicating a certain node provides means to decompose the flow carried on the incoming arcs of the original node and the flow carried on its outgoing arcs into distinct vehicles. Note that the network associated with the original problem is enlarged every time a node is duplicated

because both the number of nodes and the number of arcs increase. Hence, after a number of iterations, a regular solution is found on an extended network, for which the corresponding solution on the original network can be obtained simply by merging each node with its duplicates (if there is any).

We present the generic version of the model solved at each iteration of the node-split algorithm below along with some additional notation. Suppose that $N' = \cup_{i \in N \setminus \{0\}} N_i$, where N_i represents the set of nodes containing node $i \in N$ and its duplicates. Let $A' = \{(k, l) : \exists (i, j) \in A, k \in N_i \text{ and } l \in N_j\} \cup \{(0, i) \cup (i, 0) : i \in N'\}$ and $\bar{c}_{kl} = c_{ij}$ if $k \in N_i$ and $l \in N_j$. Similarly, let $\bar{c}_{0k} = c_{0i}$ and $\bar{c}_{k0} = c_{i0}$ if $k \in N_i$. Assume that N_i is ordered so that a node $j \in N_i$ is denoted by (i, l) , where l is the order of node j in the set N_i . Also, define:

$$v_{i,l} = \begin{cases} 1 & \text{if node } (i, l) \in N' \text{ is visited,} \\ 0 & \text{otherwise.} \end{cases}$$

(Node-Split Model)

$$\min \sum_{a \in A'} \bar{c}_a x_a \tag{3.31}$$

$$\text{s.t. } \sum_{j \in N_i} (f(\delta^-(j)) - f(\delta^+(j))) = d_i \quad i \in N \setminus \{0\}, \tag{3.32}$$

$$x(\delta^+(0)) = m, \tag{3.33}$$

$$x(\delta^-(j)) - x(\delta^+(j)) = 0 \quad j \in N', \tag{3.34}$$

$$f(\delta^-(i, l)) - f(\delta^+(i, l)) \geq 0 \quad (i, l) \in N' : |N_i| \geq 2, \tag{3.35}$$

$$x(\delta^-(i, l)) = v_{i,l} \quad (i, l) \in N' : |N_i| \geq 2, l \neq |N_i|, \tag{3.36}$$

$$x(\delta^-(i, |N_i|)) \leq (m - |N_i| + 1)v_{i,|N_i|} \quad i \in N \setminus \{0\} : |N_i| \geq 2, \tag{3.37}$$

$$v_{i,l} \geq v_{i,l+1} \quad (i, l) \in N' : |N_i| \geq 2, l \neq |N_i|, \tag{3.38}$$

$$0 \leq f_a \leq Qx_a \quad a \in A', \tag{3.39}$$

$$v_{i,l} \in \{0, 1\} \quad (i, l) \in N', \tag{3.40}$$

$$x_a \in \{0, 1\} \quad a \in A' \setminus \delta^-(0), \tag{3.41}$$

$$x_a \in \mathbb{Z}_+ \qquad a \in \delta^-(0). \quad (3.42)$$

The objective of the node-split model is to minimize the total transportation cost. Constraints (3.32) guarantee that the demand of each customer is completely satisfied. Exactly m vehicles depart from the depot due to (3.33), and constraints (3.34) ensure that the vehicle flow is conserved everywhere. For the customers having at least one duplicate; i.e., the nodes that have caused regularity violation at a previous iteration, constraint set (3.35) intends to enforce regularity property at these customers together with the constraints (3.36). More specifically, for every violating customer i , inequalities (3.35) impose non-negativity restrictions on the net incoming flow to every node in N_i and equalities (3.36) prevent more than one visit to all but the last node in N_i . In this way, only the last node in N_i can cause regularity violation during the succeeding iterations if $N_i < m$, which would be eliminated later by adding more duplicates as necessary. Eventually, the regularity is established at a violating customer by (3.35) and (3.36) after creating at most $m - 1$ duplicates. The number of visits v to every duplicate node is determined by the inequalities (3.37) and (3.38). In particular, these constraints ensure that multiple entries are allowed only for the last duplicate of a particular node and that duplicate nodes are visited in the increasing order; i.e., if l -th duplicate of a node is visited, then all the preceding duplicates must have been visited once. Lower and upper bounds on the arc flows are imposed by (3.39). Finally, constraints (3.40)–(3.42) are integrality and binary restrictions on the variables.

The node-split algorithm follows the same steps as the patching algorithm except Step 3. In this step of the node-split algorithm, among the nodes violating the regularity of the current solution, we select the first one encountered during the regularity check. We denote this node by i , create a duplicate i' of node i , and update N' by setting $N_i = N_i \cup \{i'\}$ and A' by establishing the arcs between i' and the nodes in $(N' \cup \{0\}) \setminus N_i$. We redefine the node-split model over the enlarged sets N' and A' , and then proceed to the next step.

Convergence to an optimal solution of the SDVRP is guaranteed by the node-split algorithm since the regularity violation is eliminated for a given node after $m - 1$

iterations in the worst case. More precisely, if all of the m vehicles visit a certain node, there will be at most $m - 1$ copies of the node after $m - 1$ iterations, and constraints (3.33) will force regularity for all copies, and thus, for the original node. Essentially, creating $m - 1$ duplicates of a node in this algorithm is analogous to adding vehicle-indexed variables in the patching algorithm. Even if the number of iterations required to reach an optimum is higher compared to the patching algorithm, the node-split algorithm usually works faster as will be apparent through our computational results.

3.3 Computational Study

We implemented our algorithms in Java using the mixed integer linear programming solver CPLEX 12.6 and performed a computational study on a 64-bit machine with Intel Xeon E5-2630 v2 processor at 2.60 GHz and 96 GB of RAM. The experiments were conducted on a total of 50 problem instances including benchmark instances proposed by [27], [6], [9], and a new set of randomly generated instances. In each of these instances, the number of vehicles is equal to the minimum number of vehicles required to serve the total demand, i.e., $|K| = \left\lceil \frac{d(N \setminus \{0\})}{Q} \right\rceil$. We attempted to solve the problems up to 75 customers with rounded costs. We check triangle inequality and set $c_{ik} = c_{ij} + c_{jk}$ for $(i, k) \in A$ with $c_{ik} > c_{ij} + c_{jk}$. Parallel processing is employed in our study with eight threads or 24 threads depending on the problem size. For the instances containing less than 50 customers, we use eight threads; while for larger problems, the processing is performed on 24 threads. Based on the results of preliminary computational tests, flow cover, flow path, and the mixed integer rounding cuts are switched off.

The R-SDVRP is strengthened by adding rounded capacity inequalities and cut-set inequalities at the root node of the branch-and-bound tree. Starting with a fractional solution obtained by solving the linear relaxation of the R-SDVRP, we separate the rounded capacity inequalities employing a heuristic procedure known as the connected component heuristic in the CVRP literature (see [105] for details).

Consider the support graph \bar{G} associated with a given fractional solution \bar{x} . First, we find the connected components of \bar{G} excluding the depot node. We denote these components by S_1, \dots, S_t , and for every $u = 1, \dots, t$ we check whether S_u violates the rounded capacity inequality (3.19). If no violation is detected, we try to identify a node $i \in S_u$ for which

$$\left\lceil \frac{d(S_u \setminus \{i\})}{Q} \right\rceil = \left\lceil \frac{d(S_u)}{Q} \right\rceil$$

and

$$x(\delta^-(S_u \setminus \{i\})) < x(\delta^-(S_u)),$$

remove node i from the set S_u and check for violation for the new set $S_u \setminus \{i\}$. If the new set still does not violate (3.19), we repeat the same steps until either a violated rounded capacity inequality is detected, or no node whose removal would induce a violated rounded capacity inequality exists. For the cutset inequality, separation can be performed by checking violation for subsets $A^- = \{a \in \delta^-(S) : f_a \geq rx_a\}$ and $A^+ = \{a \in \delta^+(S) : (Q - r)x_a - f_a < 0\}$ given a fractional solution (f, x) and a set $S \subseteq N \setminus \{0\}$. We apply this separation procedure for the sets S with $|S| = 1$ only; i.e., we check violation for subsets $A^- = \{a \in \delta^-(i) : f_a \geq rx_a\}$ and $A^+ = \{a \in \delta^+(i) : (Q - r)x_a - f_a < 0\}$ for every $i \in N \setminus \{0\}$. A violated rounded capacity or cutset inequality is introduced to the model if its violation is at least 10%, and the search is terminated when the improvement in the objective function value cannot exceed 5% in the last two iterations. Additionally, the variables x_a are restricted to take 0-1 values for the arcs $a \in A \setminus \delta^-(0)$ by Proposition 3.1.1.

For each problem instance, the time limit is set to two hours after violated rounded capacity cuts and cutset inequalities are separated at the root node of the search tree. If an optimal solution to the R-SDVRP cannot be found within two hours, we investigate the existence of a feasible SDVRP solution on the support graph associated with the incumbent solution (\bar{f}, \bar{x}) , which is induced by the arcs (i, j) such that $\bar{x}_{ij} \geq 1$ or $\bar{x}_{ji} \geq 1$, by employing our vehicle-indexed formulation, for which the time limit is an additional 30 minutes. Under the above settings, our results regarding the patching and the node-split algorithms are summarized in Tables 3.2 and 3.3.

For the majority of the instances in the literature, either the optimal solution of the RSDVRP is also feasible for the SDVRP; that is, the R-SDVRP yields an optimal SDVRP solution, or the R-SDVRP cannot be solved within the time limit of two hours. In fact, there is only one instance, namely *eil30*, for which our algorithms perform more than a single iteration. We note that the results in [34] show that the undirected formulation without flow variables performs better in solving most of these instances. Our aim here is not to compare different formulations with and without flow variables but to test whether the idea of extending the formulation iteratively can be useful in solving the problem. We use the formulation with flow variables and only change the way we extend the formulation in applying different methods. We need instances that are solved after several iterations to be able to compare the performances of the patching and the node-split algorithms and to see if there is a gain in extending the formulation iteratively. To this end, we introduce nine new instances to the literature (available at ozbaygin.bilkent.edu.tr), namely *r1* through *r9*, with the number of customers ranging between 30 and 47, and the number of vehicles three, four or five.

Among these instances, *r1* and *r2* are completely random. For the remaining ones, the coordinates were taken from the existing CVRP instances, while the demands are randomly generated according to three different scenarios; that is, between $[0.01Q, 0.1Q]$, $[0.01Q, 0.15Q]$, or $[0.01Q, 0.2Q]$, and the demand of one customer is increased by $Q/2$ to enhance the possibility of having at least one split customer.

The results regarding the instances for which an optimal R-SDVRP solution cannot be obtained at the end of two hours as well as the instances for which the optimal solution of our relaxation yields an admissible SDVRP solution are provided in Table 3.2. Both the patching and the node-split algorithms solve the R-SDVRP in their first iteration, hence, the two algorithms yield the same results for these instances. For the remaining instances, we give the results in Table 3.3.

We can solve 23 instances to optimality, 17 of which take a single iteration to solve because either the optimal R-SDVRP solution satisfies the regularity property, or an alternative regular solution of the same cost exists. The solution times and

Table 3.2: Results for the instances taking single iteration for the SDVRP

Instance	Number of nodes	Number of vehicles	Best known upper bound	Lower bound	Upper bound	Gap (%)	Time (sec)
eil22	22	4	375	375	375	0	3.07
eil23	23	3	569	569	569	0	1.56
eil33	33	4	835	835	835	0	19.09
eil51	51	5	521	521	521	0	264.98
eilA76	76	10	818	777.42	-	-	7200
eilB76	76	14	1002	941.25	-	-	7200
eilC76	76	8	733	709.14	-	-	7200
eilD76	76	7	681	657.46	-	-	7200
S51D1	51	3	458	458.00	458	0	21.68
S51D2	51	9	703	682.01	-	-	7200
S51D3	51	15	943	911.64	945	3.53	7200
S51D4	51	27	1551	1504.67	1555	3.24	7200
S51D5	51	23	1328	1297.37	1329	2.38	7200
S51D6	51	41	2163	2093.05	2153	2.78	7200
S76D1	76	4	592	592	592	0	1728.26
S76D2	76	15	1081	1011.45	-	-	7200
S76D3	76	23	1419	1349.64	-	-	7200
S76D4	76	37	2071	1979.51	-	-	7200
SD1	9	6	228	228	228	0	0.03
SD2	17	12	708	708	708	0	0.38
SD3	17	12	432	432	432	0	0.11
SD4	25	18	630	630	630	0	0.44
SD5	33	24	1392	1392	1392	0	6137.18
SD6	33	24	832	832	832	0	4.32
SD7	41	30	3640	3484.12	-	-	7200
SD8	49	36	5068	4790.15	-	-	7200
SD9	49	36	2046	2005.48	2046	1.98	7200
SD10	65	48	2688	2620.33	2696	2.81	7200
p01-110	51	3	458	458	458	0	21.97
p01-1030	51	11	753	722.38	755	4.32	7200
p01-1050	51	16	998	969.97	998	2.81	7200
p01-1090	51	26	1481	1440.76	1480	2.65	7200
p01-3070	51	26	1473	1433.04	1478	3.04	7200
p01-7090	51	41	2212	2075.84	2142	3.09	7200
p02-110	76	5	612	599.56	-	-	7200
p02-1030	76	16	1157	1044.54	-	-	7200
p02-1050	76	24	-	1433.98	-	-	7200
p02-1090	76	40	-	2212.47	-	-	7200
p02-3070	76	39	-	2133.99	-	-	7200
p02-7090	76	61	-	3103.35	3205	3.17	7200
r2	30	4	-	2622	2622	0	18.21
r5	36	5	-	442	442	0	29.67
r7	41	4	-	434	434	0	57.41
r8	41	5	-	468	468	0	1403.02

Table 3.3: Results for the instances taking multiple iterations for the SDVRP

Instance	Number of nodes	Number of vehicles	Best known upper bound	Patching Algorithm				Number of iterations
				Lower bound	Upper bound	Gap (%)	Time (sec)	
eil30	30	3	510	510	510	0	30.58	3
r1	30	4	-	708	708	0	105.41	2
r3	36	3	-	398	398	0	857.07	4
r4	36	4	-	421	421	0	698.62	2
r6	41	3	-	410	410	0	1033.69	3
r9	48	3	-	37025	-	-	7200	3

Instance	Number of nodes	Number of vehicles	Best known upper bound	Node-Split Algorithm				Number of iterations
				Lower bound	Upper bound	Gap (%)	Time (sec)	
eil30	30	3	510	510	510	0	183.55	4
r1	30	4	-	708	708	0	27.22	2
r3	36	3	-	398	398	0	465.49	5
r4	36	4	-	421	421	0	115.08	2
r6	41	3	-	410	410	0	382.43	2
r9	48	3	-	37234	37234	0	3686.20	4

iterations performed by both algorithms are provided in Table 3.3 for the remaining instances. Accordingly, the node-split algorithm converges to an optimal solution faster than the patching algorithm in five of the six instances.

We obtain an upper bound for 12 problem instances, and we are able to improve the best known upper bound in the literature for four of the instances that are highlighted bold in Table 3.2. In fact, regarding the instance $p02 - 7090$, we report an upper bound for the first time in the literature. In general, once the optimal R-SDVRP solution is attained, iterating for an optimal solution of the SDVRP with our patching or node-split algorithms can be effectively done. In particular, as Table 3.3 also depicts, this time is much lower for the node-split algorithm. However, as Table 3.2 clearly points out, solving even the relaxed form of the SDVRP could be quite challenging.

We also implemented and tested a variant of the patching algorithm where the integer problem is solved in one search tree. The computation times turned out to be significantly larger.

Finally, we also provide the best known upper bounds as well as the upper and lower bounds for each instance using non-rounded costs in Table 3.4. These values are obtained with the node-split algorithm. Overall, we observe that the results are similar to those in the rounded cost case.

3.4 Extensions

In this section, we introduce two new extensions of the SDVRP: (1) SDVRP with at most r splits and (2) SDVRP with open routes (SDOVRP). To the best of our knowledge, no results have been presented previously regarding these extensions, both of which can be modelled by slightly modifying our flow-based formulations.

Table 3.4: Results for the SDVRP instances with non-rounded costs

Instance	Number of nodes	Number of vehicles	Best known upper bound	Lower bound	Upper bound	Gap (%)	Time (sec)
eil22	22	4	375.28	375.28	375.28	0	3.59
eil23	23	3	568.56	568.56	568.56	0	0.81
eil30	30	3	512.72	512.72	512.72	0	465.09
eil33	33	4	837.06	837.06	837.06	0	600.90
eil51	51	5	524.61	524.61	524.61	0	890.85
eilA76	76	10	823.89	783.58	-	-	7200
eilB76	76	14	1009.04	949.56	-	-	7200
eilC76	76	8	738.67	713.34	-	-	7200
eilD76	76	7	687.60	663.44	-	-	7200
S51D1	51	3	459.50	459.50	459.50	0	17.22
S51D2	51	9	708.42	679.81	-	-	7200
S51D3	51	15	947.97	909.75	951.08	4.54	7200
S51D4	51	27	1560.88	1506.14	1569.08	4.18	7200
S51D5	51	23	1333.67	1302.63	1335.98	2.56	7200
S51D6	51	41	2169.10	2101.62	2183.02	3.87	7200
S76D1	76	4	598.94	598.94	598.94	0	4453.60
S76D2	76	15	1087.99	1023.28	-	-	7200
S76D3	76	23	1427.81	1362.89	-	-	7200
S76D4	76	37	2079.76	1994.38	-	-	7200
SD1	9	6	228.28	228.28	228.28	0	0.03
SD2	17	12	708.28	708.28	708.28	0	0.74
SD3	17	12	430.58	430.58	430.58	0	0.20
SD4	25	18	631.05	631.05	631.05	0	0.25
SD5	33	24	1390.57	1390.57	1390.57	0	2330.64
SD6	33	24	831.24	831.24	831.24	0	4.46
SD7	41	30	3640	3557.53	3640.00	2.32	7200
SD8	49	36	5068.28	4798.36	5068.28	5.63	7200
SD9	49	36	2044.20	1998.32	2044.20	2.30	7200
SD10	65	48	2684.88	2622.56	2684.88	2.38	7200
p01-110	51	3	459.50	459.50	459.50	0	18.44
p01-1030	51	11	756.71	730.04	756.71	3.65	7200
p01-1050	51	16	1005.75	980.92	1005.75	2.53	7200
p01-1090	51	26	1487.41	1457.01	1488.76	2.18	7200
p01-3070	51	26	1481.71	1439.81	1481.76	2.91	7200
p01-7090	51	41	2162.58	2093.48	2159.81	3.17	7200
p02-110	76	5	617.85	607.11	-	-	7200
p02-1030	76	16	1122.91	1050.71	-	-	7200
p02-1050	76	24	1509.79	1441.15	-	-	7200
p02-1090	76	40	2372.22	2224.98	-	-	7200
p02-3070	76	39	2235.61	2147.40	-	-	7200
p02-7090	76	61	3259.36	3131.52	3240.92	3.49	7200
r1	30	4	711.50	711.50	711.50	0	20.07
r2	30	4	2624.73	2624.73	2624.73	0	72.96
r3	36	3	399.04	399.04	399.04	0	894.00
r4	36	4	419.79	419.79	419.79	0	46.86
r5	36	5	442.91	442.91	442.91	0	102.71
r6	41	3	410.81	410.81	410.81	0	2144.34
r7	41	4	434.34	434.34	434.34	0	359.84
r8	41	5	468.91	468.91	468.91	0	1903.99
r9	48	3	37232.93	37232.93	37232.93	0	6017.46

3.4.1 SDVRP with at most r splits

Even though delivery splitting has a potential for cost savings, customers might not be willing to receive several separate deliveries due to handling inefficiencies in practice. In the SDVRP with at most r splits, split deliveries are allowed, but the demand of any customer may be covered by at most r vehicles where $1 < r < m$. Notice that when $r = 1$, the problem reduces to the CVRP, and when $r = m$, it becomes the SDVRP. There are some studies in the literature that impose a restriction on minimum delivery amounts for the vehicles visiting a customer. However, we are not aware of any work in which the number of splits is limited. A mathematical model for SDVRP with at most r splits is readily available by adding the following constraints to SDVRP model (3.1)–(3.9)

$$\sum_{k \in K} y^k(\delta^-(i)) \leq r \quad i \in N \setminus \{0\}.$$

Similarly, introducing the restriction

$$x(\delta^-(i)) \leq r \quad i \in N \setminus \{0\} \tag{3.43}$$

to the R-SDVRP model (3.10)–(3.16) provides a relaxation to SDVRP with at most r splits.

Regarding the solution approach, the patching algorithm can be implemented directly when the constraint set (3.43) is added to the R-SDVRP, and the node-split algorithm can be employed by replacing (3.37) with the following set of constraints:

$$x(\delta^-(i, |N_i|)) \leq (r - |N_i| + 1)v_{i, |N_i|} \quad i \in N \setminus \{0\} : |N_i| \geq 2 \tag{3.44}$$

since fulfilling the demand of a customer with at most r vehicles means that the customer can have no more than r duplicates at any iteration of the algorithm.

Here we consider the case $r = 2$ and provide the results of our computational experiments for the SDVRP with at most two splits. Notice that when $r = 2$ the node-split model can be simplified further, because in this case, we do not need the

variable v , and regularity violation at a node can be eliminated at once by creating a single duplicate of the node (unlike the SDVRP, which may take $m - 1$ iterations to establish regularity at a node in the worst case). More precisely, the node-split model reduces to the following:

$$\begin{aligned}
& \min \sum_{a \in A'} \bar{c}_a x_a \\
& \text{s.t.} \quad \sum_{j \in N_i} (f(\delta^-(j)) - f(\delta^+(j))) = d_i && i \in N \setminus \{0\}, \\
& \quad f(\delta^-(i, l)) - f(\delta^+(i, l)) \geq 0 && (i, l) \in N' : |N_i| = 2, \\
& \quad x(\delta^+(0)) = m, \\
& \quad x(\delta^-(j)) - x(\delta^+(j)) = 0 && j \in N', \\
& \quad x(\delta^-(i, 1)) = 1 && i \in N \setminus \{0\} : |N_i| = 2, \\
& \quad x(\delta^-(i, 2)) \leq 1 && i \in N \setminus \{0\} : |N_i| = 2, \\
& \quad 0 \leq f_a \leq Qx_a && a \in A', \\
& \quad x_a \in \{0, 1\} && a \in A' \setminus \delta^-(0), \\
& \quad x_a \in \mathbb{Z}_+ && a \in \delta^-(0).
\end{aligned}$$

Since our node-split algorithm proved more effective than the patching algorithm for the SDVRP, we attempted to solve at most two splits version using only the node-split algorithm. Table 3.5 and 3.6 indicate our results. In this case, we can solve 21 instances optimally, and obtain an upper bound for 16 instances. Different from our results for the SDVRP, we cannot reach an optimal solution for the instance $r9$.

Another way to tackle the problem with at most two splits is to solve the R-SDVRP without adding the constraint (3.43), and create duplicates of the customers receiving more than two separate deliveries in addition to those violating the regularity of the solution. We also tried to solve the SDVRP with at most two splits in this manner. The results are demonstrated in Tables 3.7 and 3.8. We can reach an optimum for the instance $r9$ in addition to the 21 instances that can be solved

Table 3.5: Results for the instances taking single iteration for the SDVRP with at most two splits

Instance	Number of nodes	Number of vehicles	Lower bound	Upper bound	Gap (%)	Time (sec)
eil22	22	4	375	375	0	4.93
eil23	23	3	569	569	0	1.53
eil33	33	4	835	835	0	60.55
eil51	51	5	521	521	0	676.38
eilA76	76	10	775.91	828	6.29	7200
eilB76	76	14	940.62	1015	7.33	7200
eilC76	76	8	708	-	-	7200
eilD76	76	7	657.01	684	3.95	7200
S51D1	51	3	458	458	0	18.94
S51D2	51	9	677.53	-	-	7200
S51D3	51	15	908.62	944	3.75	7200
S51D4	51	27	1504.19	-	-	7200
S51D5	51	23	1293.61	1329	2.66	7200
S51D6	51	41	2088.57	2206	5.32	7200
S76D1	76	4	592	592	0	1351.40
S76D2	76	15	1019.85	-	-	7200
S76D3	76	23	1349.70	-	-	7200
S76D4	76	37	1989.93	-	-	7200
SD1	9	6	228	228	0	0.02
SD2	17	12	708	708	0	1.60
SD3	17	12	432	432	0	0.28
SD4	25	18	630	630	0	0.27
SD5	33	24	1392	1392	0	10.66
SD6	33	24	832	832	0	3.39
SD7	41	30	3606.23	3640	0.93	7200
SD8	49	36	4875.15	5068	3.81	7200
SD9	49	36	2007.52	2046	1.88	7200
SD10	65	48	2612.83	2688	2.80	7200
p01-110	51	3	458	458	0	22.94
p01-1030	51	11	726.02	755	3.84	7200
p01-1050	51	16	967.69	-	-	7200
p01-1090	51	26	1445.06	1483	2.56	7200
p01-3070	51	26	1440.55	1479	2.60	7200
p01-7090	51	41	2077.65	2166	4.08	7200
p02-110	76	5	600.76	-	-	7200
p02-1030	76	16	1043.23	-	-	7200
p02-1050	76	24	1434.84	-	-	7200
p02-1090	76	40	2210.41	-	-	7200
p02-3070	76	39	2134.39	-	-	7200
p02-7090	76	61	3108.36	3343	7.02	7200
r2	30	4	2622	2622	0	22.95
r5	36	5	442	442	0	53.37
r7	41	4	434	434	0	90.95
r8	41	5	468	468	0	1085.68

Table 3.6: Results for the instances taking multiple iterations for the SDVRP with at most two splits

Instance	Number of nodes	Number of vehicles	Node-Split Algorithm				Number of iterations
			Lower bound	Upper bound	Gap (%)	Time (sec)	
eil30	30	3	510	510	0	42.20	3
r1	30	4	708	708	0	22.43	2
r3	36	3	398	398	0	327.93	4
r4	36	4	421	421	0	151.58	2
r6	41	3	410	410	0	234.69	2
r9	48	3	37105	37234	0.34	7200	4

optimally in the presence of (3.43), while we can obtain an upper bound only for the instance *eilD76*. Observe that in general, when the number of vehicles is large and the instance cannot be solved to optimality, an upper bound cannot be obtained because the solution found at the end of the two-hour time limit usually contains customers that are visited by at least three vehicles. Besides, even though some instances with large number of vehicles can be solved optimally, finding an optimal solution takes many iterations without (3.43), yielding longer computational times. Therefore, relaxing the constraint (3.43) makes it harder to terminate with an optimal or a feasible solution to the SDVRP with at most two splits for the instances containing large number of vehicles. When the number of vehicles is small, not imposing the restriction (3.43) usually improves the solution times if the optimal SDVRP solution is also feasible to the at most two splits version. Nonetheless, if the number of iterations performed to reach an optimum increases due to the relaxation of (3.43), solution times may get worse.

3.4.2 SDVRP with open routes

Another extension we present is the SDVRP with open routes (SDOVRP), which is essentially the SDVRP where vehicles are not required to return to the depot upon

Table 3.7: Results for the instances taking single iteration for the SDVRP with at most two splits when (3.43) is relaxed

Instance	Number of nodes	Number of vehicles	Lower bound	Upper bound	Gap (%)	Time (sec)
eil22	22	4	375	375	0	2.90
eil23	23	3	569	569	0	1.50
eil33	33	4	835	835	0	18.81
eil51	51	5	521	521	0	266.69
eilA76	76	10	777.40	-	-	7200
eilB76	76	14	941.24	-	-	7200
eilC76	76	8	709.16	-	-	7200
eilD76	76	7	657.46	684	3.88	7200
S51D1	51	3	458	458	0	21.45
S51D2	51	9	681.97	-	-	7200
S51D3	51	15	911.59	-	-	7200
S51D4	51	27	1504.59	-	-	7200
S51D5	51	23	1297.38	-	-	7200
S51D6	51	41	2092.83	-	-	7200
S76D1	76	4	592	592	0	1789.73
S76D2	76	15	1011.41	-	-	7200
S76D3	76	23	1349.60	-	-	7200
S76D4	76	37	1979.51	-	-	7200
SD7	41	30	3483.04	-	-	7200
SD8	49	36	4790.31	-	-	7200
SD9	49	36	2005.46	-	-	7200
SD10	65	48	2620.32	-	-	7200
p01-110	51	3	458	458	0	21.91
p01-1030	51	11	722.39	-	-	7200
p01-1050	51	16	969.95	-	-	7200
p01-1090	51	26	1440.63	-	-	7200
p01-3070	51	26	1432.99	-	-	7200
p01-7090	51	41	2075.79	-	-	7200
p02-110	76	5	599.38	-	-	7200
p02-1030	76	16	1044.37	-	-	7200
p02-1050	76	24	1433.67	-	-	7200
p02-1090	76	40	2212.56	-	-	7200
p02-3070	76	39	2134.03	-	-	7200
p02-7090	76	61	3103.14	-	-	7200
r2	30	4	2622	2622	0	21.12
r5	36	5	442	442	0	28.49
r7	41	4	434	434	0	57.91
r8	41	5	468	468	0	1401.16

Table 3.8: Results for the instances taking multiple iterations for the SDVRP with at most two splits when (3.43) is relaxed

Instance	Number of nodes	Number of vehicles	Node-Split Algorithm				Number of iterations
			Lower bound	Upper bound	Gap (%)	Time (sec)	
eil30	30	3	510	510	0	31.12	3
SD1	9	6	228	228	0	0.14	3
SD2	17	12	708	708	0	21.69	11
SD3	17	12	432	432	0	0.21	2
SD4	25	18	630	630	0	3.20	3
SD5	33	24	1392	-	-	7200	3
SD6	33	24	832	832	0	1572.21	15
r1	30	4	708	708	0	24.52	2
r3	36	3	398	398	0	360.81	5
r4	36	4	421	421	0	88.26	2
r6	41	3	410	410	0	850.77	3
r9	48	3	37234	37234	0	2322.83	4

completing their service, or they may return by visiting the customers on their route in the reverse order. The notion of open routes is mentioned for the first time by [106], but the open vehicle routing problem (OVRP) did not receive much attention until the formal introduction of the problem by [107]. Hence, it is relatively new compared to the SDVRP and the majority of the research effort on this problem seems to focus on heuristic methods (See [107], [108], [109], [110] for some examples). One exact solution approach for the problem is the branch-and-cut algorithm due to [111]. For a review of the OVRP algorithms, the reader is referred to [112]. Several variants of the OVRP have been studied so far, including capacitated OVRP, the OVRP with time windows and the OVRP with fuzzy demands. Also, there are studies involving split deliveries and open routes under the same framework as in [113] and [24], but the former is the part of a rich VRP, and the latter is within the context of a location-routing problem. To the best of our knowledge, the only study incorporating the open route structure into the classical SDVRP is due to [114], who present a tabu search heuristic for the problem. However, we are not aware of any published work in which an exact solution algorithm is proposed for the SDOVRP.

Our vehicle-indexed formulation can be adapted to the SDOVRP by simply omitting the cost terms associated with the arcs returning to the depot in the objective function; that is, the objective function of the SDOVRP is expressed as

$$\sum_{a \in A \setminus \delta^-(0)} \sum_{k \in K} c_a y_a^k.$$

In the exact same way, we can modify the objective function of the R-SDVRP as

$$\sum_{a \in A \setminus \delta^-(0)} c_a x_a$$

and employ our algorithms to solve the SDOVRP. It is important to note here that Proposition 3.1.1 does not remain valid, because reversing the direction of a route can change the total transportation cost in an open route setting. However, we can still restrict the variables x_a to take binary values for $a \in A \setminus (\delta^-(0) \cup \delta^+(0))$ as the feasible region associated with the problem does not change; i.e., we only modify the objective function. Since $x_a \in \mathbb{Z}_+$ for $a \in \delta^-(0) \cup \delta^+(0)$, the procedure for checking the regularity of a solution is adapted to handle the cases breaking symmetry. Both the patching and the node-split algorithms are used for solving the SDOVRP, and the results we obtain are reported in Tables 3.9 and 3.10. In this case, we can find an optimal solution for 28 instances while we obtain an upper bound for nine instances. Similar to the results obtained for the SDVRP, the node-split algorithm yields more favorable solution times when our algorithms perform multiple iterations. Overall, the results indicate that the problem becomes easier to handle when the depot return requirement is relaxed.

3.5 Conclusion

The SDVRP is considered in this study. A vehicle-indexed arc flow formulation is proposed for the problem as well as a relaxed model (R-SDVRP) obtained from this flow formulation. A new property regarding the optimal SDVRP solutions is derived, which guarantees the existence of an optimal SDVRP solution in which any

Table 3.9: Results for the instances taking single iteration for the SDOVRP

Instance	Number of nodes	Number of vehicles	Lower bound	Upper bound	Gap (%)	Time (sec)
eil22	22	4	252	252	0	0.55
eil23	23	3	426	426	0	1.52
eil33	33	4	511	511	0	8.33
eil51	51	5	413	413	0	60.55
eilA76	76	10	542.18	-	-	7200
eilB76	76	14	592.99	-	-	7200
eilC76	76	8	532	532	0	4015.37
eilD76	76	7	520	520	0	262.22
S51D1	51	3	405	405	0	24.63
S51D2	51	9	449.18	-	-	7200
S51D3	51	15	526.32	541	2.71	7200
S51D4	51	27	798.70	-	-	7200
S51D5	51	23	698.18	-	-	7200
S51D6	51	41	1083.54	-	-	7200
S76D1	76	4	515	515	0	141.94
S76D2	76	15	617.80	-	-	7200
S76D3	76	23	742.03	-	-	7200
S76D4	76	37	1040.08	-	-	7200
SD1	9	6	128	128	0	0.03
SD2	17	12	368	368	0	0.25
SD3	17	12	232	232	0	0.06
SD4	25	18	330	330	0	2.38
SD5	33	24	712	712	0	2.18
SD6	33	24	432	432	0	1.64
SD7	41	30	1820	1820	0	12.12
SD8	49	36	2548	2548	0	8.23
SD9	49	36	1050	1050	0	8.92
SD10	65	48	1377.90	1392	1.01	7200
p01-110	51	3	405	405	0	19.91
p01-1030	51	11	466.41	474	1.60	7200
p01-1050	51	16	588.29	-	-	7200
p01-1090	51	26	770.42	791	2.60	7200
p01-3070	51	26	764.53	786	2.73	7200
p01-7090	51	41	1069.77	1100	2.75	7200
p02-110	76	5	513	513	0	92.20
p02-1030	76	16	625.15	-	-	7200
p02-1050	76	24	781.50	809	3.40	7200
p02-1090	76	40	1153.18	-	-	7200
p02-3070	76	39	1115.61	-	-	7200
p02-7090	76	61	1580.73	1634	3.26	7200
r3	36	3	332	332	0	12.83
r4	36	4	334	334	0	7.10
r5	36	5	334	334	0	27.73
r6	41	3	349	349	0	15.19
r7	41	4	351	351	0	18.04
r8	41	5	347	347	0	5.77
r9	48	3	30787	30787	0	12

Table 3.10: Results for the instances taking multiple iterations for the SDOVRP

Instance	Number of nodes	Number of vehicles	Patching Algorithm				Number of iterations
			Lower bound	Upper bound	Gap (%)	Time (sec)	
eil30	30	3	375	375	0	329.84	3
r1	30	4	506	507	0.19	7200	5
r2	30	4	1796	1796	0	11.65	2

Instance	Number of nodes	Number of vehicles	Node-Split Algorithm				Number of iterations
			Lower bound	Upper bound	Gap (%)	Time (sec)	
eil30	30	3	375	375	0	145.65	5
r1	30	4	506	507	0.19	7200	5
r2	30	4	1796	1796	0	8.64	2

arc emanating from the depot is traversed at most once. We devise two new exact solution algorithms based on the idea of iteratively extending the relaxation by adding variables and constraints until finding a solution satisfying the regularity property. Additionally, we introduce two extensions of the SDVRP, namely, the SDVRP with at most r splits, and the SDVRP with open routes (SDOVRP). We adapt our relaxation and algorithms to tackle these extensions. Computational experiments are performed on 50 problem instances in total, 41 of which are benchmark instances from the literature, and nine of which are randomly generated new instances. Results are reported regarding the SDVRP, SDVRP with open routes and SDVRP with at most r splits for the case of $r = 2$. Accordingly, we can remark that our algorithms effectively iterate until an optimal SDVRP solution is found as long as the R-SDVRP can be solved quickly. Nevertheless, solving the R-SDVRP is a difficult task, especially for large sized problem instances. Therefore, focusing on the ways to handle this relaxation more efficiently would be a useful contribution in the future. Also, the lower bounds of the R-SDVRP can be strengthened by investigating new classes of valid inequalities for the SDVRP. In this way, it might be possible to tackle larger instances. Finally, the structural properties of the SDVRP with open routes can be further investigated, which may lead to new solution methodologies for the

problem.

Chapter 4

Time Constrained Maximal Covering Salesman Problem with Weighted Demands and Partial Coverage

The TSP, one of the most intensively studied combinatorial optimization problems, aims to identify a least cost Hamiltonian tour on a given network. All nodes of the network must be visited exactly once in the TSP. But, this may not be viable in many real life applications due to resource limitations. Hence, identifying a tour over a subset of the nodes so that the others are within a reasonable distance of some tour stop can be more desirable. For instance, consider the postal delivery services. In a region with many demand points (customers), it may be very costly and inconvenient to visit every single demand point separately. Instead, siting postboxes at a subset of customer locations and collecting mails through these boxes is more efficient. In such a system, each postbox is used for covering the demand of multiple customers; i.e.,

the customers without a postbox at their location can drop off mails to their closest box. A relevant objective in this context would be the minimization of tour length (cost). However, there may also be cases where cost is not the primary concern. Consider the routing of mobile health facilities as an example. The main issue here is to ensure, with the available resources, that the number of patients who receive health care is maximized.

In this study, we consider the TCMCSP, in which the aim is to find a tour visiting a subset of the demand points, so as to maximize the demand covered subject to a time constraint. We assume that the demands of the vertices that are on the tour are fully covered while only a certain percentage α of the demand of a vertex is covered if it is not visited but is within a specified distance r of some tour stop. This is a realistic assumption because not every demand point may be willing to travel a distance of r units to reach a tour stop. As an example, a passenger may not want to take the bus to his destination if he does not want to walk to the closest bus stop because he is closer to a metro stop. Thus, we can presume that $\alpha\%$ of the demands is covered regarding the points not on the tour. Due to the upper bound on the tour duration, some vertices may be left isolated (uncovered); i.e., they are not served at all.

Potential real-life applications of TCMCSP include the routing of mobile health facilities, collection of blood, distribution of food, drinking water and medical supplies in the aftermath of a disaster, routing of security patrol cars in rural regions for crime prevention, routing of unmanned aerial vehicles (UAVs) for information gathering against intruders, deciding on the order and the location of the meetings for a political campaign and so on. In all of these applications, there is a restriction on the tour length, and the primary objective is to maximize coverage.

To the best of our knowledge, only a single study exists related to the TCMCSP (due to [40]). The problem structure in this study is different from ours since the authors in [40] distinguish the set of customers and the set of facilities, and they identify a tour over a subset of the facilities. In addition, they consider the allocation of customers to facilities. The problem we study is a special case of the one in [40]

where the set of facilities and the set of customers are the same and the allocation is disregarded. For this special case, we are able to propose stronger and smaller formulations and efficient solution methods.

Our contributions can be summarized as follows. We study the TCMCSP with weighted demands and partial coverage. We propose two mathematical formulations and valid inequalities for the problem, and develop branch-and-cut solution methodologies. We are able to solve instances of realistic sizes to optimality within a time limit of one hour.

The rest of this chapter is organized as follows. The literature on several problems that are related to TCMCSP are reviewed in the next section. Mathematical formulations and valid inequalities for TCMCSP are given in Section 4.1. Section 4.2 presents four branch-and-cut schemes to solve the problem. The results of our computational study along with a discussion of the sensitivity of optimal solutions to changes in problem parameters are reported in Section 4.3. Finally, Section 4.4 concludes the chapter with a summary of our findings.

4.1 Formulation and Valid Inequalities

In this section, we formally define TCMCSP and propose two mathematical models. Afterwards, we present several classes of valid inequalities for the problem.

Let $G = (V, E)$ be an undirected complete graph with the set of vertices $V = \{0, 1, \dots, n\}$ and the set of edges $E = \{\{i, j\} : i, j \in V, i < j\}$. Suppose that vertex 0 represents a central depot and the remaining vertices correspond to demand points. We assume that the tour contains at least three vertices and that $|V| \geq 4$. Every edge $e \in E$ is associated with a nonnegative length c_e and every vertex $i \in V \setminus \{0\}$ has a positive demand denoted by d_i . We assume that the edge lengths satisfy the triangle inequality. Define N_i to be the set of vertices within the coverage distance r_i of vertex i other than the vertex i itself, i.e., $N_i = \{j \in V : c_{\{i,j\}} \leq r_i, j \neq i\}$. The

time constrained maximal covering salesman problem is the problem of determining a route over a subset of vertices in V that maximizes the total demand covered. The route must start and end at the depot and its length should not exceed a prespecified threshold value L . A vertex i can be either visited or covered by some vertex $j \in N_i$, or left isolated. We consider the demand of the vertex i as fully covered if it is on the tour while only a percentage $\alpha > 0$ of its demand can be covered if i is not visited and the tour contains at least one vertex from the set N_i .

4.1.1 Mathematical formulations

Our first model is a cut based model. We use the following variables in this model:

$$x_e = \begin{cases} 1 & \text{if edge } e \in E \text{ is on the tour,} \\ 0 & \text{otherwise,} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if vertex } i \in V \text{ is on the tour,} \\ 0 & \text{otherwise,} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if vertex } i \in V \text{ is not visited but covered by some vertex in } N_i, \\ 0 & \text{otherwise.} \end{cases}$$

For $S \subset V$, we let $\delta(S) = \{e \in E : |e \cap S| = 1\}$ and $E(S) = \{e \in E : |e \cap S| = 2\}$. If $S = \{i\}$ we simply use $\delta(i)$ instead of $\delta(\{i\})$. Finally, we write $x(E') = \sum_{e \in E'} x_e$ for $E' \subseteq E$ and $y(S) = \sum_{i \in S} y_i$ for $S \subseteq V$. Our first mathematical model for the TCMCSP is given below:

$$\max \sum_{i \in V \setminus \{0\}} d_i(y_i + \alpha z_i), \tag{4.1}$$

$$\text{s.t. } \sum_{e \in E} c_e x_e \leq L, \quad (4.2)$$

$$y_i + z_i \leq 1 \quad i \in V \setminus \{0\}, \quad (4.3)$$

$$x(\delta(i)) = 2y_i \quad i \in V, \quad (4.4)$$

$$z_i \leq y(N_i) \quad i \in V \setminus \{0\}, \quad (4.5)$$

$$x(\delta(S)) \geq 2y_i \quad S \subset V \setminus \{0\}, 3 \leq |S| \leq n - 2, i \in S, \quad (4.6)$$

$$y_0 = 1, \quad (4.7)$$

$$x_e \in \{0, 1\} \quad e \in E, \quad (4.8)$$

$$y_i \in \{0, 1\} \quad i \in V \setminus \{0\}, \quad (4.9)$$

$$z_i \in \{0, 1\} \quad i \in V \setminus \{0\}. \quad (4.10)$$

The objective (4.1) is to maximize the total demand covered. Constraint (4.2) ensures that the total tour length does not exceed L . Due to (4.3), a vertex can be isolated, on the tour, or covered by the tour stops within its coverage distance. Degree requirement for each vertex is imposed by (4.4), i.e., a visited vertex has degree two whereas a vertex that is not visited cannot have any edge adjacent to it. Constraints (4.5) guarantee that a vertex i cannot be covered if none of the vertices in N_i is visited. Connectivity cuts in (4.6) prevent subtours and are exponential in the size of the problem. Constraint (4.7) enforces the depot to be on the tour. Finally, domain restrictions on the variables are given in (4.8)–(4.10).

Next, we modify the formulation presented in [40] to model our problem. This model is a flow based directed model. For this reason, we define the set of arcs $A = \{(i, j), (j, i) : \{i, j\} \in E\}$ and we let the length of arcs $\hat{c}_{ij} = \hat{c}_{ji} = c_{\{i, j\}}$ for each $\{i, j\} \in E$. In addition to the y and z variables defined above, we use the following decision variables: \hat{x}_a is 1 if arc $a \in A$ is part of the tour and it is 0 otherwise. Also, f_{ij} is the total traveled time (distance) from the depot to vertex j , when traversing arc $(i, j) \in A$. For $A' \subseteq A$, we let $\hat{x}(A') = \sum_{a \in A'} \hat{x}_a$ and $f(A') = \sum_{a \in A'} f_a$. Then, the TCMCSP can be modeled as:

$$\max \sum_{i \in V \setminus \{0\}} d_i(y_i + \alpha z_i), \quad (4.11)$$

s.t. (4.3), (4.5), (4.7), (4.9), (4.10)

$$\sum_{a \in A} \hat{c}_a \hat{x}_a \leq L, \quad (4.12)$$

$$\hat{x}(\delta^+(i)) = \hat{x}(\delta^-(i)) = y_i \quad i \in V, \quad (4.13)$$

$$f(\delta^+(i)) - f(\delta^-(i)) = \sum_{a \in \delta^+(i)} \hat{c}_a \hat{x}_a \quad i \in V \setminus \{0\}, \quad (4.14)$$

$$f_{0i} = \hat{c}_{0i} \hat{x}_{0i} \quad i \in V \setminus \{0\}, \quad (4.15)$$

$$f_{ij} \leq (L - \hat{c}_{j0}) \hat{x}_{ij} \quad (i, j) \in A, j \neq 0, \quad (4.16)$$

$$f_{i0} \leq L \hat{x}_{i0} \quad i \in V \setminus \{0\}, \quad (4.17)$$

$$f_{ij} \geq (\hat{c}_{0i} + \hat{c}_{ij}) \hat{x}_{ij} \quad (i, j) \in A, i \neq 0, \quad (4.18)$$

$$\hat{x}_a \in \{0, 1\}, f_a \geq 0 \quad a \in A. \quad (4.19)$$

Here constraints (4.13) are degree constraints. Constraints (4.14)–(4.18) relate the variables f 's and \hat{x} 's and eliminate subtours. Note that this formulation is valid only for instances with positive arc lengths. Otherwise, it is possible to obtain solutions that contain zero length subtours and the nodes on these subtours are counted as visited.

4.1.2 Lifting connectivity constraints

Our initial computational experiments (presented in Section 4) show that we are able to solve larger instances using the first model based on connectivity cuts. In the sequel, we present valid inequalities for the feasible set of this model that we denote by Y .

For $\hat{V} \subseteq V$, we define $Y^0(\hat{V}) = \{(x, y, z) \in Y : z_j = 0, j \in \hat{V}\}$. $Y^0(V)$ is the feasible set of an orienteering problem. In other words, the polytope of the orienteering problem is a face of the polytope of TCMCSP where all z_j variables are fixed to zero. The family of valid inequalities that we present is obtained by lifting the connectivity cuts with variables z_i 's. These inequalities are strong when the

connectivity cut that we lift is strong for the polytope associated with the orienteering problem and some mild conditions are satisfied.

Theorem 4.1.1 *Let $S \subset V \setminus \{0\}$ with $3 \leq |S| \leq n - 2$ and $i \in S$ such that $\emptyset \neq N_i \subset S$. The lifted connectivity inequality (LCI)*

$$x(\delta(S)) \geq 2y_i + 2z_i \tag{4.20}$$

is valid for Y .

Proof. If $z_i = 0$, then (4.20) reduces to constraint (4.6). If $z_i = 1$, then $y_i = 0$ and $y(N_i) \geq 1$. Since $N_i \subset S$ and $y(N_i) \geq 1$, there exists $j \in S$ with $y_j = 1$. Constraint (4.6) for set S and node j implies $x(\delta(S)) \geq 2$. Hence, inequality (4.20) is satisfied in both cases. \square

Let $V' = \{j \in V \setminus \{0\} : N_j \neq \emptyset\}$. If $j \notin V'$, then we know that $z_j = 0$ in all feasible solutions. Hence $Y^0(V) = Y^0(V')$.

Theorem 4.1.2 *Suppose that the connectivity constraint (4.6) for set $S \subset V \setminus \{0\}$ with $3 \leq |S| \leq n - 2$ and node $i \in S \cap V'$ is facet defining for $\text{conv}(Y^0(V))$, $N_i \subset S$ and the cycle on nodes $\{0, k, l\}$ satisfies constraint (4.2) for any two distinct nodes k and l in $V \setminus \{0\}$. Then inequality (4.20) is facet defining for $\text{conv}(Y)$.*

Proof. We lift the connectivity constraint (4.6) first with z_i and then with z_j for $j \in V' \setminus \{i\}$. We first would like to find σ_i such that the inequality $x(\delta(S)) \geq 2y_i + \sigma_i z_i$ is satisfied by all solutions in $Y^0(V' \setminus \{i\})$. The inequality is clearly satisfied for all σ_i when $z_i = 0$. If $z_i = 1$, then $y_i = 0$ and $y(N_i) \geq 1$. In this case, we need $\sigma_i \leq x(\delta(S))$ for all (x, y, z) in $Y^0(V' \setminus \{i\})$ with $z_i = 1$. Equivalently, we want $\sigma_i \leq \min_{(x,y,z) \in Y^0(V' \setminus \{i\}): z_i=1} x(\delta(S))$. We know that $\min_{(x,y,z) \in Y^0(V' \setminus \{i\}): z_i=1} x(\delta(S)) \geq 2$

since $N_i \subset S$ and $y(N_i) \geq 1$. Consider the solution where $z_i = 1$, $y_i = 0$, $y_k = y_l = 1$, $z_k = z_l = 0$ for some $k \in N_i$ and some $l \in V \setminus \{0, i, k\}$ and for all other nodes j , we have $y_j = z_j = 0$. We let $x_{\{0,k\}} = x_{\{k,l\}} = x_{\{l,0\}} = 1$ and other edge variables be zero. This solution is in $Y^0(V' \setminus \{i\})$ with $z_i = 1$ and $x(\delta(S)) = 2$. Hence $\min_{(x,y,z) \in Y^0(V' \setminus \{i\}): z_i=1} x(\delta(S)) = 2$. Consequently, the inequality $x(\delta(S)) \geq 2y_i + \sigma_i z_i$ is valid for all $\sigma_i \leq 2$ and the inequality $x(\delta(S)) \geq 2y_i + 2z_i$ is facet defining for $\text{conv}(Y^0(V' \setminus \{i\}))$.

Now let π be a permutation on $V' \setminus \{i\}$. Next we lift inequality $x(\delta(S)) \geq 2y_i + 2z_i$ with variables z_l for $l \in V' \setminus \{i\}$ in the order π . To lift inequality $x(\delta(S)) \geq 2y_i + 2z_i$ with $z_{\pi(1)}$, we would like to compute $\min_{(x,y,z) \in Y^0(V' \setminus \{i, \pi(1)\}): z_{\pi(1)}=1} (x(\delta(S)) - 2y_i - 2z_i)$. We know that $x(\delta(S)) - 2y_i - 2z_i \geq 0$ for all $(x, y, z) \in Y^0(V' \setminus \{i, \pi(1)\})$. If $i \in N_{\pi(1)}$, then let k be any node in $V \setminus \{0, i, \pi(1)\}$. If $i \notin N_{\pi(1)}$, then let k be a node in $N_{\pi(1)}$. We set $z_{\pi(1)} = 1$, $y_{\pi(1)} = 0$, $y_k = y_i = 1$, $z_k = z_i = 0$ and for other nodes j , we have $y_j = z_j = 0$. We use edges $\{0, k\}$, $\{k, i\}$, and $\{i, 0\}$. This solution is in $Y^0(V' \setminus \{i, \pi(1)\})$ with $z_{\pi(1)} = 1$ and $x(\delta(S)) - 2y_i - 2z_i = 0$. Hence the optimal lifting coefficient for $z_{\pi(1)}$ is zero and the inequality $x(\delta(S)) \geq 2y_i + 2z_i$ is facet defining for $\text{conv}(Y^0(V' \setminus \{i, \pi(1)\}))$. Now suppose that the inequality is facet defining for $\text{conv}(Y^0(V' \setminus \{i, \pi(1), \dots, \pi(u-1)\}))$ for $2 \leq u \leq |V'|$ and that we are lifting it with $z_{\pi(u)}$. We can show using the same arguments that the optimal coefficient for $z_{\pi(u)}$ is also zero. Hence inequality (4.20) is facet defining for $\text{conv}(Y)$. \square

If $S = N_i \cup \{i\}$, then we call the resulting inequality (4.20) a simple lifted connectivity inequality (SLCI). Note that these inequalities are polynomial in number.

4.1.3 Optimality cuts and simple cover inequalities

We also derive some optimality cuts and simple valid inequalities for TCMCSP based on the idea of knapsack covers.

Theorem 4.1.3 *For $i \in V'$ such that $0 \in N_i$, in an optimal solution the following equality holds*

$$y_i + z_i = 1. \tag{4.21}$$

Proof. Since both d_i and α are positive, the result follows. \square

Hence, we use (4.21) instead of constraint (4.3) for such vertices.

Theorem 4.1.4 *Let i and j be distinct vertices in $V \setminus \{0\}$. If $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$, then every feasible solution satisfies $x_{ij} = 0$. In addition, the inequality $y_i + y_j \leq 1$ holds.*

Proof. For two distinct vertices i and j satisfying the above condition, we have $x_{ij} = 0$ because we assume that c satisfies the triangle inequality. Moreover, at most one of the vertices i and j can be visited in any feasible solution since even the length of a shortest cycle on $\{0, i, j\}$ exceeds the bound L . Hence, the inequality $y_i + y_j \leq 1$ is valid. \square

Theorem 4.1.5 *Let $S \subset V \setminus \{0\}$ and $i, j \in S$ be two distinct vertices such that $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$. Then, the following cover inequality*

$$x(\delta(S)) \geq 2y_i + 2y_j \tag{4.22}$$

is valid and dominates constraint (4.6) for both i and j .

Proof. We know that at most one of i and j can be included in a feasible solution by the previous theorem. If $y_i = 1$, then $y_j = 0$ and (4.22) reduces to the connectivity constraint (4.6) for S and i . The case for $y_j = 1$ is similar. Hence, the result follows. \square

4.2 Branch-and-Cut Algorithms

We devise branch-and-cut algorithms to solve the TCMCSP since our cut formulation involves exponentially many constraints. We propose four branch-and-cut schemes. The most basic version starts by solving the relaxation (4.1)–(4.5), (4.7)–(4.10) and the violated connectivity constraints (4.6) are introduced only for integer solutions of the branch-and-cut tree. In the second scheme, we separate the connectivity constraints also for fractional solutions at the root node of the tree. The last two schemes are similar to the second one in terms of where in the solution tree the separation procedures are executed. However, in the third scheme, we add the SLCIs for $i \in V \setminus \{0\}$ with $0 \notin N_i$ to the initial relaxation. Moreover, before checking whether a connectivity constraint $x(\delta(S)) \geq 2y_i$ is violated, we investigate the corresponding LCI given by $x(\delta(S)) \geq 2(y_i + z_i)$ if $0 \notin N_i$ and $N_i \subset S$. It may be the case that the condition $0 \notin N_i$ holds, yet there exists $j \in N_i \setminus S$. In that case, we extend the set S by adding the vertices in $N_i \setminus S$ to it and explore whether the inequality $x(\delta(S \cup N_i)) \geq 2(y_i + z_i)$ is satisfied. If either of these LCIs is violated, we introduce it instead of the connectivity constraint. The last branch-and-cut scheme is similar to the third one. Besides adding SLCIs to the initial relaxation (4.1)–(4.5), (4.7)–(4.10), we add the inequalities $y_i + y_j \leq 1$ and set $x_{ij} = 0$ for every vertex pair $i, j \in V \setminus \{0\}$ such that $i < j$ and $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$. During the separation of connectivity constraints, we search for violated cover inequalities in addition to LCIs. For a set $S \subset V \setminus \{0\}$ and a vertex $i \in S$, if we detect $j \in S$ with $j \neq i$ for which the corresponding cover inequality is not satisfied or if we identify a violated LCI, we do not check the violation of the connectivity constraint induced by i and S . Our separation procedures are described in detail in the next subsections.

4.2.1 Separation of connectivity constraints

In separating the connectivity constraints, we use the ideas proposed in [115]. Suppose that $\tilde{G} = (\tilde{V}, \tilde{E})$ is the support graph induced by a given solution vector $(\bar{x}, \bar{y}, \bar{z})$;

i.e., $\bar{V} = \{i \in V : \bar{y}_i > 0\}$ and $\bar{E} = \{e \in E : \bar{x}_e > 0\}$. Let S_j , $j = 0, 1, \dots, t$ be the connected components of \bar{G} where $0 \in S_0$. There are two possibilities regarding the solution vector $(\bar{x}, \bar{y}, \bar{z})$: either it is integral or it has at least one fractional component. In the former case, the solution is feasible for the TCMCSP if and only if $t = 0$; that is, the corresponding support graph \bar{G} is connected. When $t \geq 1$, there is a connectivity cut violated by S_j and each $i \in S_j$ for every $j = 1, \dots, t$. Hence, the solution vector induces $\sum_{j=1}^t |S_j|$ violated constraints and introducing any one of them to the model cuts off the current solution. Nevertheless, instead of adding a single cut at a time, we add the cut (4.6) for every S_j , $j = 1, \dots, t$ and for every $i \in S_j$ in order to speed up the solution procedure.

Now consider the case where the solution is fractional. If $t \geq 1$ for the corresponding support graph \bar{G} , a violated connectivity constraint is induced by every S_j and $i \in S_j$ for $j = 1, \dots, t$ as in the previous case. However, if \bar{G} is connected, exact separation of violated connectivity cuts can be performed by solving a series of minimum cut problems on the graph \bar{G} . Checking violation of the inequality $x(\delta(S)) \geq 2\bar{y}_k$ for a vertex $k \in \bar{V} \setminus \{0\}$ and for every $S \subset V \setminus \{0\}$ such that $k \in S$ is equivalent to checking whether the capacity of a minimum cut separating the vertices k and 0 is greater than or equal to $2\bar{y}_k$ when the capacity of each edge $e \in \bar{E}$ is set to \bar{x}_e . If the capacity of a minimum cut separating vertex k and vertex 0 is at least $2\bar{y}_k$, then every $S \subset V \setminus \{0\}$ containing k satisfies (4.6). Otherwise, we obtain a violated connectivity cut corresponding to k and the vertex partition $S^*(k)$, where $[S^*(k), V \setminus S^*(k)]$ defines a minimum cut with respect to source k and sink 0 . In particular, $S^*(k)$ is the vertex partition containing k .

In some cases, there may be a more efficient way to identify violated connectivity cuts than solving $|\bar{V}| - 1$ minimum cut problems. Let $cap(S)$ denote the capacity of the cut defined by the set S of vertices. Suppose that $S^* \subset \bar{V} \setminus \{0\}$ is the set of vertices inducing a global minimum cut of the graph \bar{G} . Then, we have $cap(S^*(k)) = cap(S^*)$ and thus, $S^*(k) = S^*$ for every $k \in S^*$. This means that S^* is a minimum cut separating any $k \in S^*$ from the vertex 0 , and we do not have to solve a separate minimum cut problem for each vertex in S^* . Besides, we know that $cap(S) \geq cap(S^*)$ for any cut S of the graph \bar{G} . This implies that the corresponding

connectivity constraints are satisfied for the vertices $i \in (\bar{V} \setminus \{0\}) \setminus S^*$ such that $cap(S^*) \geq 2y_i$. Therefore, we can eliminate these vertices from consideration as well. For each of the remaining vertices, we solve a minimum cut problem to determine if there are any violated connectivity constraints.

4.2.2 Separation of lifted connectivity and cover inequalities

We investigate violated lifted connectivity and cover inequalities during the execution of connectivity constraint separation procedure. More specifically, for a particular vertex i and a set $S \subset V \setminus \{0\}$ with $i \in S$, we separate LCIs and cover inequalities prior to the corresponding connectivity constraint. In the following, we describe our separation subroutines for LCIs and cover inequalities.

First, consider the case with an integer solution containing subtours denoted by S_0, S_1, \dots, S_t where $0 \in S_0$. Take any S_k for $k = 1, \dots, t$. Let $i \in S_k$ and j be a vertex such that $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$. Then $y_j = 0$ must hold. Clearly, we have $x(\delta(S_k \cup \{j\})) = 0$ because both $x(\delta(S_k))$ and $x(\delta(j))$ are equal to zero. Now observe that the set $S_k \cup \{j\}$ and the vertices i, j induce a violated cover inequality. Hence, while examining a certain vertex $i \in S_k$ for a particular $k = 1, \dots, t$, we introduce a cover inequality for i and each j such that $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$. After exploring these inequalities, we evaluate whether the LCI induced by the vertex i and the set $N_i \cup S_k$ is satisfied, provided that $0 \notin N_i$. If $N_i \subset S_k$, then we have a violated LCI since $x(\delta(S_k)) = 0$. Else, we should verify that $x(\delta(N_i \setminus S_k)) = 0$ to be able to add the corresponding LCI, as otherwise $x(\delta(N_i \cup S_k))$ is at least two and the inequality is satisfied. We add the connectivity constraint induced by i and S_k if no cover inequality involving the vertex i is identified so far and either one of the following conditions holds:

- $0 \notin N_i$ but no LCI violation is detected,
- $0 \in N_i$.

Suppose now that we have a fractional solution. If the solution contains subtours S_0, S_1, \dots, S_t with $0 \in S_0$, we separate LCIs and connectivity constraints in the same manner as we do in the integer solution case. However, our cover inequality separation procedure is slightly different. For a subtour S_k for $k = 1, \dots, t$ and a vertex $i \in S_k$, we add the cover inequalities for all $j \in S_k$ such that $i < j$ and $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$.

If the support graph associated with a given fractional solution is connected, we embed our search for LCI and cover inequalities into our connectivity constraint separation algorithm as well. Once a global minimum cut S^* of the support graph \bar{G} is determined, we check violation of the LCIs and cover inequalities for each $i \in S^*$ before exploring the corresponding connectivity constraint. The capacity of the cut $S^* \cup N_i$ is compared with the value $2(y_i + z_i)$ to investigate whether the LCI associated with i and $S^* \cup N_i$ is satisfied (if $0 \notin N_i$). If it is violated, then we add it to the model. Afterwards, for every $j \in S^*$ such that $i < j$ and $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$, we introduce the cover inequality induced by i, j and S^* if the capacity of S^* is less than $2(y_i + y_j)$. Upon completing the search for LCIs and cover inequalities for all $i \in S^*$, we evaluate violation of the connectivity constraint regarding each vertex in S^* for which no violated LCI or cover inequality is identified.

Among the vertices in $\bar{V} \setminus (S^* \cup \{0\})$, we eliminate those with $2(y_k + z_k) \leq \bar{x}(\delta(S^*))$ because the capacity of any cut of \bar{G} is at least as large as that of S^* . Hence, our algorithm will not detect any violated LCI or connectivity constraint for such vertices. For each remaining vertex i , we find the minimum cut $S^*(i)$ as in our connectivity constraint separation procedure, and check whether the LCI induced by i and $S^*(i) \cup N_i$ is satisfied (again if $0 \notin N_i$). Then, we inspect every $j \in S^*(i)$ such that $j \neq i$ and $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$ in case a violated cover inequality exists. If the LCI and cover inequalities associated with the vertex i are all found to be satisfied, we investigate the corresponding connectivity constraint.

4.2.3 Other implementation details

We invoke *StoerWagnerMinimumCut* and *MinSourceSinkCut* procedures from the Java graph theory (*jgrapht*) library to find a global minimum cut of a given undirected graph (it implements the algorithm in [116]) and a minimum cut between a specified pair of source and sink nodes in a directed graph (it implements the algorithm in [117]), respectively. Note that since *MinSourceSinkCut* works on a directed graph, we transform \bar{G} into a directed graph by replacing each edge $\{i, j\} \in \bar{E}$ with the arcs (i, j) and (j, i) , and by assigning a capacity of \bar{x}_{ij} units to both arcs after solving the global minimum cut problem.

In our branch-and-cut schemes that involve separation for fractional solutions, all cuts (LCIs, cover inequalities, connectivity constraints) are separated only at the root node of the branch-and-cut tree. Obviously, the separation for integer solutions is conducted everywhere. For a fractional solution, we consider an inequality to be violated if its violation exceeds 5% based on the results of our preliminary experiments. The violation of an inequality $LHS \geq RHS$, where LHS and RHS represent the left-hand side and the right-hand side values of the inequality respectively, is determined by the ratio $(RHS - LHS)/RHS$. We apply parallel processing with 12 threads of implementation and use default branching rules provided by CPLEX.

4.3 Computational Study

We performed a computational study on a set of test problems based on seven VRP instances (available at <http://neumann.hec.ca/chairedistributique/data/>), namely, *p01*, *p02*, *p03*, *p04*, *p05*, *p11* and *p12* in which the number of vertices range between 51 and 200. Note that there are actually 14 instances on this website; however, only seven of them are different with respect to node coordinates and demand values. The remaining seven instances differ from the ones above based on the number of vehicles involved and service times, which are not relevant parameters for our problem. The

reason for choosing these instances for our tests instead of TSP instances is that we consider the TCMCSP with arbitrary demands rather than unit demands. We experimented with varying values of the parameters r , L and α in our tests.

The experiments were conducted on a 64-bit machine with Intel Xeon E5-2630 v2 processor at 2.60 GHz and 96 GB of RAM. All models and algorithms were implemented in Java by invoking CPLEX 12.6 with Concert Technology. The time limit is set to one hour.

In our computations, we assume that $r_i = r$ for every $i \in V$, and we used 10 and 20 for the coverage distance r . These values are about 25% and 50% of the average edge cost, which is approximately 37 when computed over all instances. To choose the time/distance constraint parameter L , we first solved the TSP for each instance. Let $z_i^*(TSP)$ denote the cost of an optimal TSP solution for instance i . Then, three different values of L were obtained by rounding 25%, 50% and 75% of the number $z_i^*(TSP)$ for instance i . Finally, the partial coverage parameter α was set to 50% and 75%. Therefore, we performed our computational study with 12 different parameter configurations for each of the seven test problems, which means that we attempted to solve 84 instances in total.

4.3.1 Comparison of the proposed algorithms

The results of our computational study are provided in Tables 4.1–4.5. Each table demonstrates the solution times (in seconds) and root gap values (percentage gap between the final root relaxation bound and the optimal value) for all instances obtained through a different solution scheme. If an instance cannot be solved to optimality within the time limit, we report the final gap (percentage gap between the objective function value of the best integer solution and the best upper bound) in paranthesis.

We tested the flow-based formulation on a total of 48 instances containing up to

Table 4.1: Results with the flow formulation

Instance			Solution Times (sec)					
name	n	α	$r = 10$			$r = 20$		
			L_1	L_2	L_3	L_1	L_2	L_3
$p01$	51	0.50	99.57	23.83	21.28	39.65	7.57	3.23
		0.75	266.89	47.21	30.07	102.28	19.63	3.69
$p02$	76	0.50	(5.72)	91.20	92.06	1229.98	13.86	57.81
		0.75	(12.79)	175.50	123.81	(4.49)	66.35	70.93
$p03$	101	0.50	(13.76)	1842.29	291.56	(5.56)	267.11	213.54
		0.75	(24.64)	2328.21	750.29	(6.65)	416.44	248.80
$p12$	101	0.50	(70.54)	(29.76)	(8.97)	(36.84)	(22.17)	(8.85)
		0.75	(70.89)	(14.51)	(4.06)	(26.91)	(12.95)	(5.02)
			Root gaps (%)					
$p01$	51	0.50	21.57	3.65	1.23	8.02	1.89	0.26
		0.75	25.65	4.02	0.65	5.42	1.07	0.21
$p02$	76	0.50	21.99	3.34	1.77	8.39	0.47	0.82
		0.75	26.33	2.97	0.71	6.56	0.47	0.34
$p03$	101	0.50	21.73	3.20	0.84	10.84	1.68	0.49
		0.75	21.66	1.58	0.49	6.58	0.88	0.32
$p12$	101	0.50	87.56	33.06	14.82	48.58	30.45	14.92
		0.75	76.50	19.03	6.94	42.09	15.17	6.94

100 customers, namely with $p01$, $p02$, $p03$ and $p12$. The results are provided in Table 4.1.

Next, we tested our initial branch-and-cut method, where only the connectivity constraints are separated at integer points of the solution tree. The results in Table 4.2 show that 61 of the 84 instances can be solved optimally within one hour and the average solution time for these instances is 372.13 seconds. Regarding the remaining 23 instances, the final gap is 29.49% on the average. Additionally, the root gap values indicate that the upper bounds given by the relaxation (4.1)–(4.5), (4.7)–(4.10) are quite weak. In particular, the average and worst gaps are 15.75% and 134.40%, respectively, for the initial branch-and-cut scheme.

When we compare the results in Tables 4.1 and 4.2, we observe the following: with the flow formulation, 19 out of 42 instances cannot be solved to optimality within one hour while 12 of these 19 instances can be solved optimally by our first branch-and-cut scheme. Regarding the remaining 7 instances, the final gaps of our first branch-and-cut algorithm are 51.5% less (on the average) than that of the flow formulation. Finally, the instances for which an optimal solution can be found by the flow formulation require 98% less solution time on average when solved by our first branch-and-cut algorithm. Hence, we can conclude that the first branch-and-cut algorithm outperforms the flow formulation.

Next, the test instances are solved with our branch-and-cut scheme in which connectivity constraints are separated not only at the integer points, but also at the fractional points of the solution tree. Based on some preliminary experiments, separation for fractional solutions is performed only at the root node of the branch-and-cut tree as mentioned in the previous section. Note that this is also the case for the remaining two schemes. Table 4.3 reports the results obtained with our second branch-and-cut algorithm. The number of instances for which an optimal solution is found within one hour is 67 with an average solution time of 216 seconds, and the average final gap is 14.5% for the remaining 17 instances. Moreover, we can observe that introducing some violated connectivity constraints at the root node of the solution tree strengthens the linear relaxation bounds of (4.1)–(4.5), (4.7)–(4.10)

Table 4.2: Results with branch-and-cut scheme 1

Instance			Solution Times (sec)					
name	n	α	$r = 10$			$r = 20$		
			L_1	L_2	L_3	L_1	L_2	L_3
$p01$	51	0.50	4.77	1.43	0.71	1.58	0.78	0.37
		0.75	7.16	9.32	0.58	3.33	1.10	0.44
$p02$	76	0.50	92.24	28.29	13.06	26.79	0.82	2.43
		0.75	1236.86	19.27	3.10	121.28	2.86	4.65
$p03$	101	0.50	(3.29)	76.31	3.28	360.18	21.64	7.48
		0.75	(11.31)	75.81	5.07	2278.70	20.15	7.85
$p04$	151	0.50	(14.76)	82.71	29.67	1275.24	49.62	22.84
		0.75	(19.06)	47.65	26.08	1965.48	45.01	46.20
$p05$	200	0.50	(8.95)	915.06	(0.09)	(1.57)	360.07	3046.80
		0.75	(16.78)	2400.86	(0.05)	(2.26)	455.59	2238.77
$p11$	121	0.50	(117.76)	(28.17)	43.68	(70.41)	(20.10)	176.11
		0.75	(114.68)	(22.81)	246.03	(106.07)	(9.60)	190.07
$p12$	101	0.50	(34.41)	1140.06	120.74	(4.43)	584.69	65.27
		0.75	(60.27)	(2.21)	232.87	(9.27)	2322	131.35

			Root gaps (%)					
$p01$	51	0.50	17.46	3.07	0.35	6.08	0.69	0.25
		0.75	23.92	2.71	0.38	4.57	0.71	0.20
$p02$	76	0.50	16.01	3.07	0.41	8.23	0.69	0.27
		0.75	22.89	1.47	0.15	6.04	0.08	0.13
$p03$	101	0.50	17.44	1.80	0.18	9.42	0.25	0.22
		0.75	19.47	1.04	0.09	5.74	0.11	0.11
$p04$	151	0.50	18.79	2.84	0.12	5.27	2.23	0.96
		0.75	21.23	0.30	0.47	2.36	1.01	0.47
$p05$	200	0.50	12.87	1.65	1.56	4.55	3.43	1.30
		0.75	18.43	1.04	0.76	3.05	1.57	0.63
$p11$	121	0.50	134.40	44.31	10.85	101.86	32.24	10.71
		0.75	126.07	28.52	5.14	116.82	14.64	5.08
$p12$	101	0.50	89.72	30.20	11.17	53.41	30.30	14.03
		0.75	93.87	18.25	5.20	42.84	14.62	6.94

Table 4.3: Results with branch-and-cut scheme 2

Instance			Solution Times (sec)					
name	n	α	$r = 10$			$r = 20$		
			L_1	L_2	L_3	L_1	L_2	L_3
$p01$	51	0.50	4.65	2.76	0.63	2.52	0.74	0.86
		0.75	6.06	3.77	0.96	3.63	0.81	0.69
$p02$	76	0.50	38.63	7.45	6.68	34.22	1.26	3.76
		0.75	1030.43	8.26	2.46	416.02	1.73	3.14
$p03$	101	0.50	405.66	49.01	2.33	60.21	14.83	3.01
		0.75	(10.70)	23.63	7.69	548.01	27.62	2.49
$p04$	151	0.50	(11.95)	33.54	14.84	351.40	34.15	20.19
		0.75	(17.83)	31.80	68.87	718.63	21.26	31.76
$p05$	200	0.50	(5.66)	682.59	(0.02)	973.62	118.26	1039.73
		0.75	(14.44)	1253.14	1771.94	1499.60	113.69	323.85
$p11$	121	0.50	(16.05)	(7.07)	40.25	(19.57)	(0.63)	82.88
		0.75	(55.53)	(4.40)	109.03	(69.85)	(0.93)	101.50
$p12$	101	0.50	(4.79)	98.19	14.59	132.85	72.91	19.80
		0.75	(5.29)	(1.91)	58.05	1186.50	688.58	39.93
Root gaps (%)								
$p01$	51	0.50	19.22	2.24	0.27	5.89	0.73	0.22
		0.75	25.56	2.45	0.27	4.74	0.38	0.11
$p02$	76	0.50	15.75	1.24	0.27	6.09	0.10	0.21
		0.75	23.44	1.13	0.01	5.54	0.05	0.10
$p03$	101	0.50	15.50	0.21	0.07	6.20	0.03	0.06
		0.75	19.43	0.16	0.04	4.44	0.01	0.02
$p04$	151	0.50	15.18	0.06	0.04	3.02	0.09	0.06
		0.75	20.31	0.07	0.06	1.49	0.04	0.03
$p05$	200	0.50	7.90	0.33	0.32	2.01	0.18	0.16
		0.75	15.74	0.36	0.16	1.11	0.08	0.08
$p11$	121	0.50	77.38	20.06	0.49	80.17	15.05	0.41
		0.75	101.55	9.73	0.23	102.69	6.10	0.19
$p12$	101	0.50	72.02	14.53	4.53	38.72	12.96	4.65
		0.75	72.68	11.34	2.09	37.79	7.40	2.18

and reduces the average and maximum root gap values to 11.02% and 102.69%, respectively. Compared to the initial scheme, the solution time (or the optimality gap for the instances that cannot be solved optimally within the time limit) and the root gaps improve in almost all of the instances.

In our third scheme, we add SLCIs to the initial relaxation (4.1)–(4.5), (4.7)–(4.10) for i with $0 \notin N_i$ and separate LCIs in addition to connectivity constraints. Based on the results in Table 4.4, all instances can be solved to optimality within at most 1043.97 seconds. The average solution time is 57.11, indicating a 74% decrease compared to the previous scheme. There is also a significant improvement regarding the root gaps. In particular, the average root gap decreases to 1.18%, which implies a 90% reduction with respect to our second scheme. Overall, introducing SLCIs and violated LCIs during the solution procedure leads to a remarkable improvement both in terms of the solution times and the root relaxation bounds.

Finally, we test the effect of using cover inequalities. In addition to LCIs and connectivity constraints, we also separate cover inequalities in our last branch-and-cut scheme. Furthermore, for every pair of vertices $i, j \in V \setminus \{0\}$ with $c_{\{0,i\}} + c_{\{i,j\}} + c_{\{j,0\}} > L$, we include the inequalities $y_i + y_j \leq 1$ and set $x_{ij} = 0$ in the initial relaxation besides adding SLCIs. Although the average solution time decreases to 39.31 and the average root gap is as low as 0.35% in the final scheme, these reductions are mostly caused by the instance $p11$. In other words, there is no considerable change in terms of the solution times and the root gaps compared to the previous scheme except for a few instances based on $p11$. No violated inequalities of this family are detected for most of the instances. However, we observe that it may be possible to obtain significant improvements in some cases. As an example, the instance $p11$, $\alpha = 0.75$, $r = 20$ and $L = 0.25z_{p11}^*(TSP)$ can be solved within 9.55 seconds with the help of cover inequalities, while the solution time for the same instance was 1043.97 seconds in our previous scheme.

As indicated above, remarkable improvements can be achieved regarding solution times and linear relaxation bounds with the introduction of LCIs and cover inequalities. Based on the results of our worst and best branch-and-cut algorithms, decrease

Table 4.4: Results with branch-and-cut scheme 3

Instance			Solution Times (sec)					
name	n	α	$r = 10$			$r = 20$		
			L_1	L_2	L_3	L_1	L_2	L_3
p01	51	0.50	1.33	0.54	0.56	0.37	0.53	0.74
		0.75	1.03	0.60	0.70	0.17	0.10	0.94
p02	76	0.50	2.01	3.17	3.75	0.48	1.47	2.93
		0.75	4.65	1.51	0.63	2.57	1.40	3.23
p03	101	0.50	5.78	5.74	3.22	6.23	2.55	1.88
		0.75	8.76	6.74	5.48	5.98	3.08	2.09
p04	151	0.50	30.93	15.96	16.30	13.88	29.92	26.70
		0.75	24.19	3.76	30.18	19.11	23.72	39.55
p05	200	0.50	93.32	160.44	341.83	70.62	99.51	662.62
		0.75	139.33	236.39	211.76	41.50	144.36	270.73
p11	121	0.50	38.73	50.56	8.51	165.79	118.82	11.13
		0.75	260.89	84.13	9.13	1043.97	24	12.49
p12	101	0.50	9.12	4.85	9.89	5.11	6.54	16.23
		0.75	11.07	22.09	12.63	13.33	10.15	8.33

			Root gaps (%)					
p01	51	0.50	0.59	0.17	0.22	0.97	0.05	0.22
		0.75	0.67	0.01	0.23	0.05	0	0.11
p02	76	0.50	1.30	0.22	0.11	0	0.06	0.21
		0.75	1.42	0.01	0.01	0.69	0.02	0.10
p03	101	0.50	0.07	0.14	0.07	0.08	0	0.06
		0.75	0.97	0.02	0.04	0.43	0	0.03
p04	151	0.50	0.52	0.05	0.02	0	0.07	0.05
		0.75	0.02	0	0.06	0	0.03	0.02
p05	200	0.50	0.04	0.11	0.26	0.07	0.11	0.16
		0.75	0.19	0.10	0.13	0	0.05	0.08
p11	121	0.50	10.89	3.52	0.06	13	1.48	0.15
		0.75	20.33	0.24	0.03	29.90	0.03	0.07
p12	101	0.50	0.98	0.34	0.20	0.99	0.44	0.32
		0.75	0.71	1.35	0.11	2.14	0	0.15

Table 4.5: Results with branch-and-cut scheme 4

Instance			Solution Times (sec)					
name	n	α	$r = 10$			$r = 20$		
			L_1	L_2	L_3	L_1	L_2	L_3
$p01$	51	0.50	1.66	0.52	0.54	0.67	0.45	0.67
		0.75	0.98	0.61	0.76	0.26	0.09	0.78
$p02$	76	0.50	2.78	3.08	3.75	0.52	1.39	2.80
		0.75	4.63	1.61	0.61	2.08	1.45	3.06
$p03$	101	0.50	6.87	5.92	3.02	8.15	2.49	1.78
		0.75	9.72	6.55	5.58	6.69	3.07	2.04
$p04$	151	0.50	45.22	15.56	15.58	12.24	30.25	26.27
		0.75	43.74	3.85	30.78	18.37	23.68	40.51
$p05$	200	0.50	93.24	162.44	343.41	70.75	99.10	657.46
		0.75	139.98	233.22	214.95	40.75	144.74	263.60
$p11$	121	0.50	3.12	42.25	8.08	4.64	101.34	10.82
		0.75	8.22	53.28	9.13	9.55	43.34	12.01
$p12$	101	0.50	7.72	4.82	9.47	9.57	6.22	15.75
		0.75	12.87	22.13	12.34	10.16	10.23	8.09

			Root gaps (%)					
$p01$	51	0.50	0.42	0.17	0.22	0.72	0.05	0.22
		0.75	0.82	0.01	0.23	0.16	0	0.11
$p02$	76	0.50	1.29	0.22	0.11	0	0.06	0.21
		0.75	1.50	0.01	0.01	0.69	0.02	0.10
$p03$	101	0.50	0.16	0.14	0.07	0.08	0	0.06
		0.75	0.86	0.02	0.04	0.45	0	0.03
$p04$	151	0.50	0.52	0.05	0.02	0.03	0.07	0.05
		0.75	0.01	0	0.06	0.01	0.03	0.02
$p05$	200	0.50	0.04	0.11	0.26	0.07	0.11	0.16
		0.75	0.19	0.10	0.13	0	0.05	0.08
$p11$	121	0.50	0.18	3.52	0.06	0.03	1.47	0.15
		0.75	0.31	0.08	0.03	5.06	0.04	0.07
$p12$	101	0.50	0.98	0.34	0.20	0.86	0.44	0.32
		0.75	0.56	1.35	0.11	2.22	0	0.15

in the average solution time is about 90% and the average root gap reduces by 98%. In our best algorithm (the last scheme), the maximum solution time is 657.46 seconds while there are 23 instances that cannot be solved optimally within the time limit of one hour by our initial algorithm. This shows the effectiveness of LCIs and cover inequalities in terms of speeding up the solution procedure. Moreover, the largest root gap with our initial algorithm is 134.40%, whereas it is only 5.06% in the last one, which is another evidence of the power of these inequalities in strengthening the linear relaxation bounds of our formulation.

4.3.2 Impacts of parameter changes on the optimal solutions

In a network optimization problem, topology of the underlying graph and the adopted parameter setting are among the key determinants of the optimal solution structure. In order to understand the effects of these factors, we investigate the structure of the optimal TCMCSP solutions by evaluating the percentage of demand covered, the number of tour stops and partially covered vertices as well as tour and coverage diameters. The diameter of a tour is defined as the largest distance between two vertices on the tour. Similarly, the coverage diameter is the largest distance between any pair of vertices that are fully or partially covered. Note that the vertices are scattered throughout the graph for $p01$, $p02$, $p03$, $p04$, $p05$, while they are located in several clusters for $p11$ and $p12$. In all the figures below, the filled circles represent tour stops and partially covered vertices while the empty ones correspond to isolated vertices.

As expected, the percentage of total demand covered increases with larger values of the partial coverage parameter α . The tour diameter tends to increase as well for the instances where the underlying graph is scattered. This is mostly achieved by visiting fewer vertices in an attempt to cover more vertices partially. The increase in the tour diameter and in the total demand covered obtained as a result of using a greater α value is most notable when the value of L ; i.e., the maximum tour length parameter, is small, because the coverage diameter is usually large enough to accommodate

almost all vertices of the graph sufficiently close to the tour when L is large. Figure 4.1 shows an example for $\alpha = 0.50$ and $\alpha = 0.75$ when L and the coverage distance r are fixed. For the instances where the underlying graph is clustered, changing the value of α does not usually affect the tour diameter, because the coverage cannot be enhanced unless the tour is expanded to reach an unvisited cluster, which is not possible without increasing L sufficiently. An example is depicted in Figure 4.2.

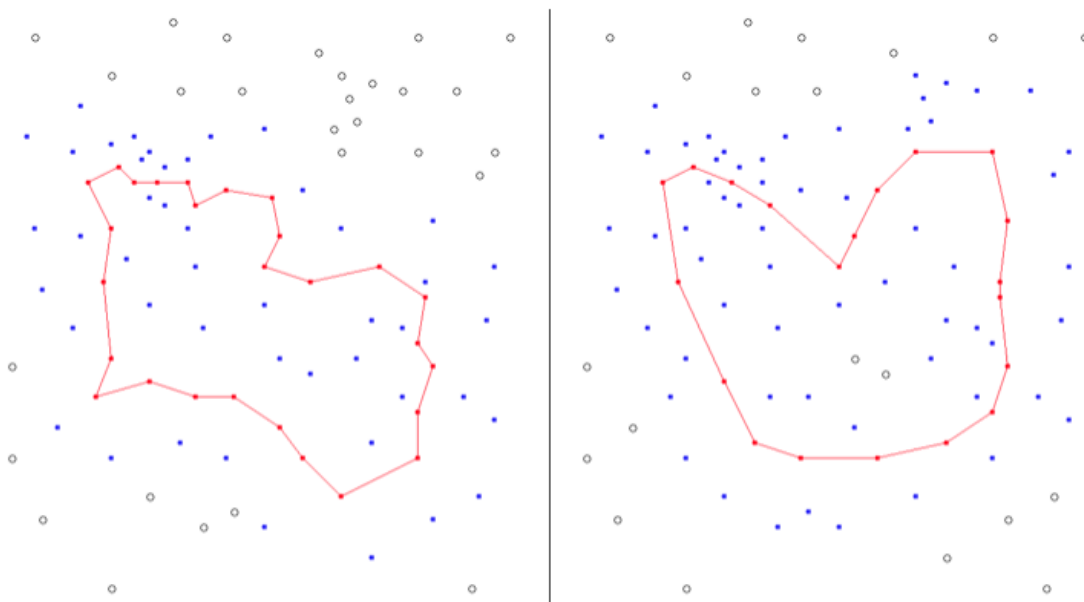


Figure 4.1: The optimal solutions of $p03$ with $L = L_1$, $r = 10$ for $\alpha = 0.5$ and $\alpha = 0.75$ respectively

Regarding the impact of tour length restriction on the optimal solution structure, we observed that a higher percentage of the total demand can be covered and the coverage diameter expands as L gets larger. In scattered graphs, more vertices are visited while fewer vertices are partially covered when L is increased. Moreover, changes in the percentage of demand that is covered on-tour and off-tour have a similar impact when compared to changes in the number of tour stops and partially covered vertices, respectively. An illustration of the impact of maximum tour length is given in Figure 4.3 for a scattered graph.

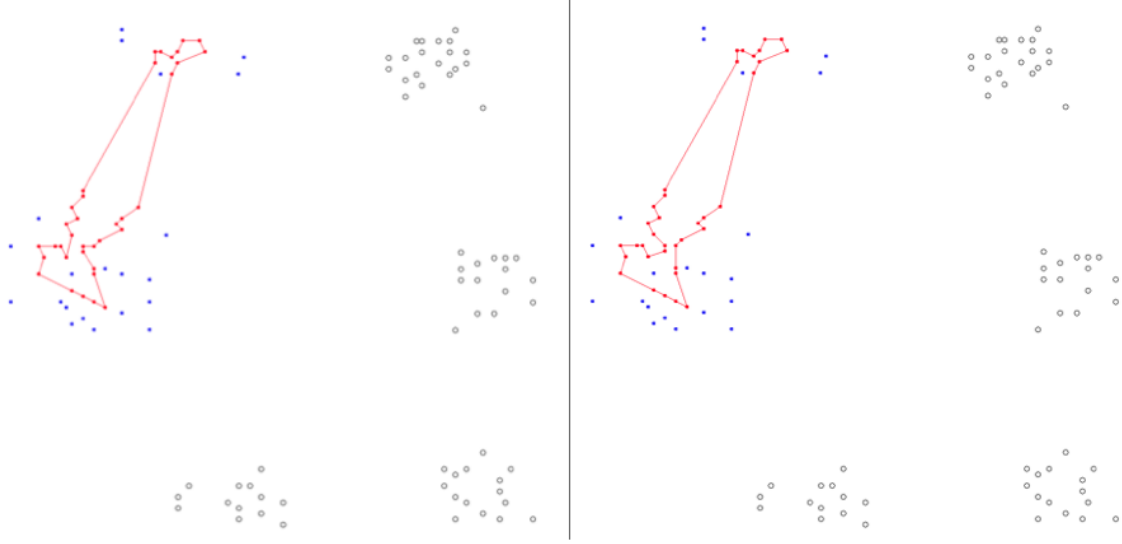


Figure 4.2: The optimal solutions of $p11$ with $L = L_1$, $r = 10$ for $\alpha = 0.5$ and $\alpha = 0.75$ respectively

On the other hand, in clustered graphs, the total demand covered is improved first by attempting to reach as many clusters as possible, while respecting the tour length constraint imposed by L , and then by visiting the maximum possible number of vertices in these clusters. Therefore, even though the number of tour stops does not necessarily get larger as L increases, the percentage of demand that is fully covered becomes higher according to our observations. Furthermore, if the increase in L is enough to reach some previously unvisited clusters; i.e., if the growth in the tour diameter is significant, then the percentage of partially covered demand becomes larger. When L increases further (but not enough to visit another cluster), more vertices in the visited clusters are included in the tour, resulting in a decrease in the percentage of partially covered demand. An example of this situation is illustrated in Figure 4.4.

Finally, we explored how the optimal solutions are affected by the changes in coverage distance. Raising the value of the coverage distance parameter increases the percentage of total demand covered and the coverage diameter in general. Note that this increase is more significant in scattered graphs, whereas the optimal solution

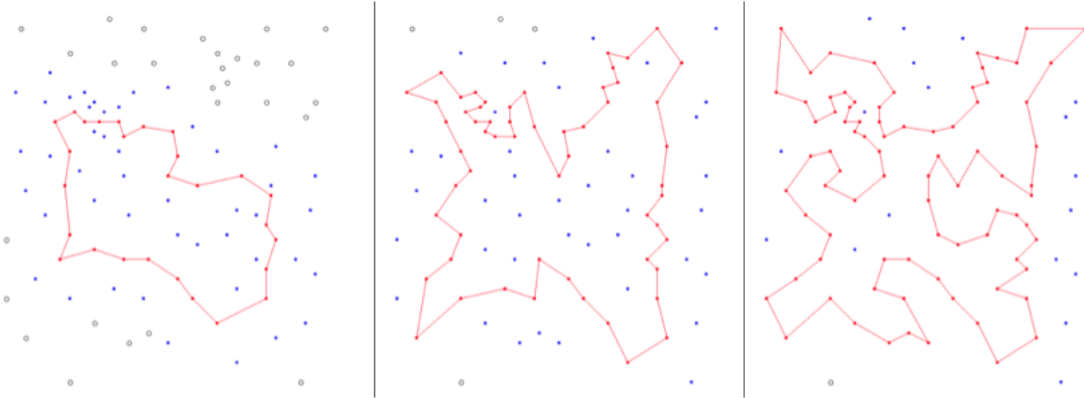


Figure 4.3: The optimal solutions of $p03$ with $\alpha = 0.5$, $r = 10$ for $L = L_1$, $L = L_2$ and $L = L_3$ respectively

may only slightly change or even remain unchanged in clustered graphs unless the increase in r is sufficient to cover some vertices without visiting the clusters they belong to. Based on our results, we observed that increasing r usually gives rise to a decrease in the tour diameter. This enables visiting more vertices while maintaining the same coverage level especially when the underlying graph is scattered and dense. Figure 4.5 exhibits how the optimal solution changes depending on the coverage distance for fixed α and L . However, regarding the clustered instances where r is increased enough to allow some vertices in a different cluster to be covered by the tour stop(s), we noticed that the tour diameter increases and fewer vertices are visited. Such an example is depicted in Figure 4.6.

Overall, we can conclude that the maximum tour length L is the most critical parameter that determines the optimal solutions based on the test instances used in our experiments. The impact of changing L can be remarkable especially when the underlying problem graph is clustered. Even though the parameters α and r affect the optimal solution in terms of which vertices should be designated as tour stops and which ones should be partially covered in order to ensure maximum coverage, the improvement achieved by increasing the values of these parameters cannot go beyond a certain limit (assuming that $\alpha < 1$) under the same tour length restriction.

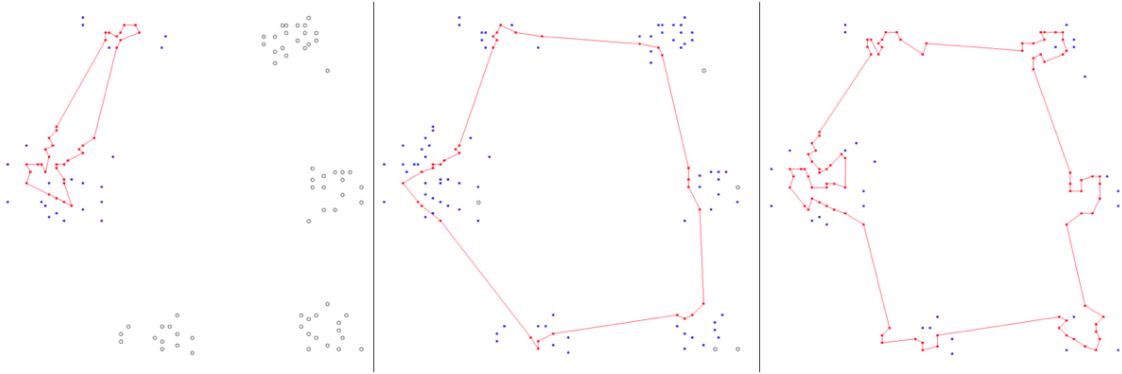


Figure 4.4: The optimal solutions of $p11$ with $\alpha = 0.5$, $r = 10$ for $L = L_1$, $L = L_2$ and $L = L_3$ respectively

This is actually an intuitive result since with sufficiently large L values, all vertices would become tour stops. However, in general we can remark that as α gets larger, the optimal solution tends to contain more partially covered vertices and fewer tour stops scattered through a slightly wider area (the tour diameter is larger). Conversely, increasing r often leads to a reduction in the tour diameter, which facilitates visiting more vertices, except for some clustered graphs. Finally, it is clear that the impact of changing the value of a parameter on the optimal solutions also depends on the underlying problem graph.

4.4 Conclusion

We have considered the TCMCSP in which the goal is to find a tour visiting a subset of the vertices that maximizes the amount of demand covered subject to an upper bound on the tour length. This problem is practically relevant in cases where it is not efficient to visit every demand point separately. Integrating the notion of coverage into a routing scheme; i.e., satisfying the demand of multiple customers through each customer on the route, may provide means to increase system efficiency by utilizing the available resources more effectively. This chapter presents an efficient solution

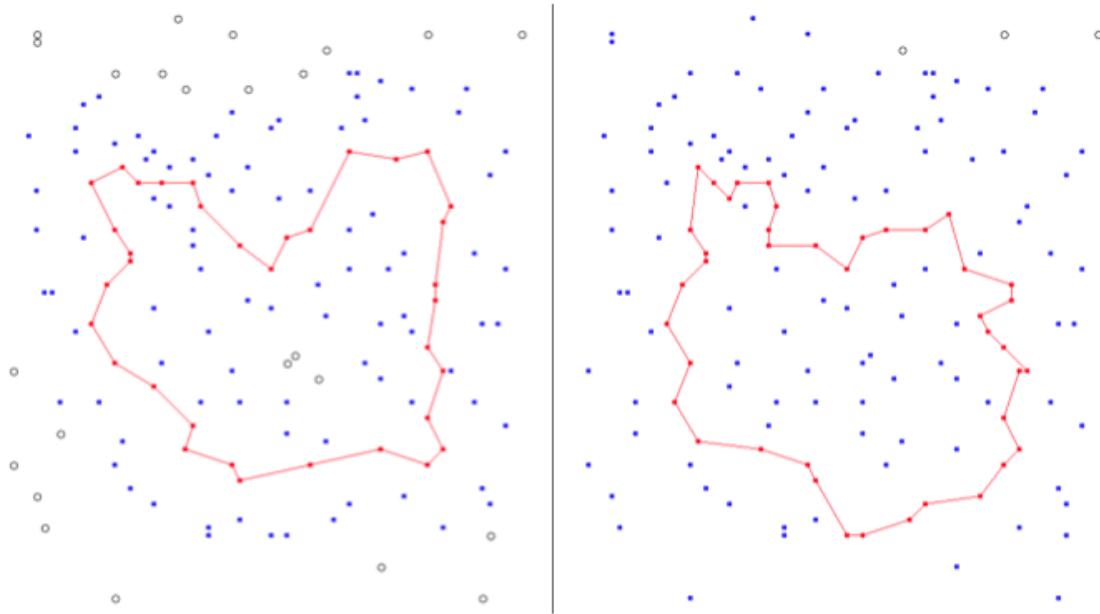


Figure 4.5: The optimal solutions of $p04$ with $\alpha = 0.5$, $L = L_1$ for $r = 10$ and $r = 20$ respectively

approach to a problem that unifies coverage and routing.

In the TCMCSP, we assumed that the demand of a vertex is fully covered if it is visited, partially covered if it is not visited but sufficiently close to some vertex on the tour, and not covered otherwise. We modeled the problem on an undirected network and since our formulation involves exponentially many connectivity constraints, we proposed branch-and-cut algorithms to solve the TCMCSP. We presented simple optimality cuts and two families of valid inequalities, namely the lifted connectivity inequalities (LCIs) and cover inequalities. Four branch-and-cut schemes were devised to evaluate the impact of LCIs and cover inequalities on solution times and LP relaxation bounds of our formulation.

We also adapted the flow formulation presented in [40] to model the problem. Computational experiments demonstrated the superiority of using a branch-and-cut solution approach over the flow formulation. Moreover, the results indicated the

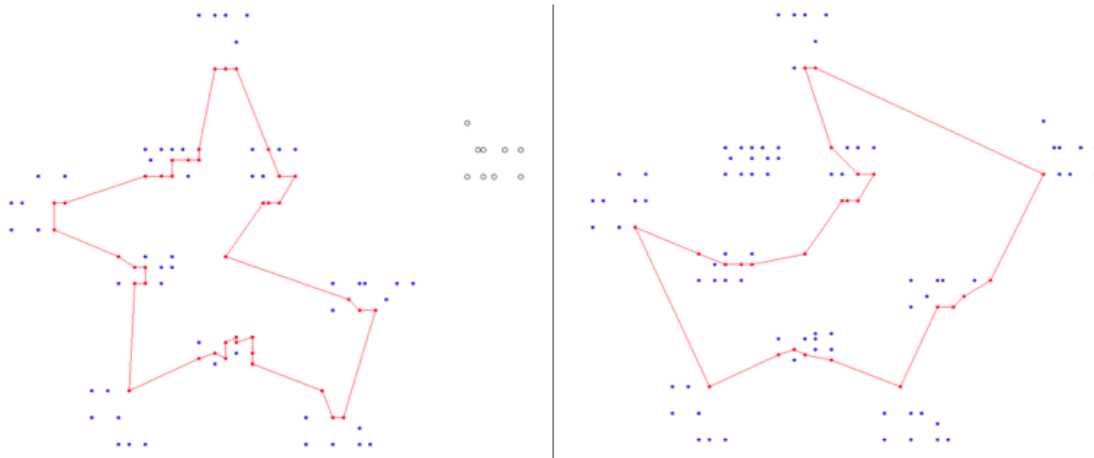


Figure 4.6: The optimal solutions of $p12$ with $\alpha = 0.75$, $L = L_2$ for $r = 10$ and $r = 20$ respectively

effectiveness of LCIs and cover inequalities, both of which were shown to be quite powerful in increasing the quality of linear relaxation bounds of our formulation and accelerating the solution procedure.

Finally, we investigated the effects of the partial coverage multiplier (α), maximum tour length (L) and coverage distance (r) parameters on the optimal solution structure. We conclude that L has the largest impact on the optimal solutions. Additionally, there is a tendency towards visiting fewer vertices as α increases, while extending r usually results in the reverse behavior.

Chapter 5

A Branch-and-Price Algorithm for the Vehicle Routing Problem with Roaming Delivery Locations

A recent survey revealed that in 2016, for the first time, online purchases have surpassed in-store purchases [118]. The business-to-consumer retail segment continues to grow year-over-year with an ever-increasing push towards online shopping. As an example, Amazon, one of the e-commerce giants received 398 orders per second and 34.4 million orders in total during its Prime Day event on July 15th, 2015 [119]. A year later, on Prime Day 2016, the number of orders increased by more than 60%, making it the biggest sales day in the history of the company [120]. Due to the sheer volume, the huge number of delivery locations, and the aggressive service levels promised, last-mile logistics is a huge challenge for Amazon. Amazon's first-quarter shipping costs in 2016 hit \$3.27 billion, a 42% increase from the same period in 2015 [121].

Not surprisingly, Amazon is seeking and exploring innovative ideas to improve

the efficiency of last-mile delivery operations. Among these innovative ideas is trunk delivery, where the orders of customers are delivered to the trunk of their cars, and introduced by Amazon in collaboration with Audi and DHL in certain areas ([122–124]). Volvo too is offering the required technology and has launched an in-car delivery service, initially limited to Stockholm, Sweden ([125, 126]). Recently, Smart has partnered with DHL to start providing trunk delivery service to customers in Germany who order merchandise from Amazon in September 2016 [127].

Motivated by the interest in trunk delivery, we study the variant of the VRP in which each customer has an itinerary specifying one or more locations with corresponding time windows where the customer’s order can be delivered to the trunk of his/her car (the car will be parked at these locations during the given time windows). This problem is introduced in [57] and referred to as the VRPRDL. This problem can be seen as a special case of the GVRPTW in which the sets of delivery locations for the customers form the clusters. However, the time windows exhibit a special structure, as the time windows of the locations in a cluster, i.e., the time windows of the delivery locations for a single customer, are non-overlapping. Our computational experiments reveal that it is this special structure that allows us to solve large instances. We note, however, that our solution approach does not explicitly exploit the time window structure and can, thus, also be used to solve instances of the GVRPTW.

We consider the static and deterministic version of the VRPRDL, i.e., we assume complete knowledge of the itinerary of each customer and deterministic travel times. It is not unrealistic, when planning delivery routes, to assume that knowledge of a customer’s itinerary is available, as many companies, e.g., Roadie roadie.com, are already using analytics to predict the travel patterns of people using the location data collected from the GPS tracking technology in their smart phones. However, given that the actual travel times and customer itineraries may differ when the planned delivery routes are executed, in practice, technology to dynamically adjust routes will also be needed.

The main contribution of this study is that it presents an effective branch-and-price algorithm for the VRPRDL. Branch-and-price [128] has established itself as an effective solution methodology for a variety of vehicle routing and scheduling problems, e.g., the VRPTW [61], the VRP with stochastic demands [129], the VRP with pickup and deliveries [130], and the VRP with split deliveries [131]. As far as we know, this is the first study in which an exact solution approach for the VRPRDL, or, more generally, for the GVRPTW, is developed. The novelty of our branch-and-price algorithm lies in the fact that it works with location clusters instead of locations. This requires modification of existing techniques for solving the pricing problem and modification of the implementation of certain branching rules. Furthermore, since the GVRPTW generalizes several variants of the VRP, our branch-and-price algorithm can be used to solve instances of these variants as well.

An extensive computational study shows that our branch-and-price algorithm is capable of solving instances of up to 120 customers. We also demonstrate that the algorithm can be used to solve instances of the variant in which a customer order can be delivered either to the customer’s home or to the customer’s car, although only instances of up to 60 customers. In [57], this problem variant is called the VRP with home and roaming delivery locations (VRPHRDL). Finally, we use our algorithm to analyze the cost savings that can be achieved when making optimal use of the option to deliver to the trunk of a car.

The remainder of this chapter is organized as follows. Section 5.1 presents a mixed-integer programming formulation as well as a set-partitioning model for the VRPRDL, describes the pricing problem and how to solve it. Section 5.2 discusses the techniques employed in our branch-and-price algorithm to increase its efficiency and gives some implementation details. Section 5.3 explains how the VRPHRDL can be solved with our algorithm. Section 5.4 provides the results of an extensive computational study, demonstrating the efficacy of the branch-and-price algorithm and analyzing the benefits of trunk delivery. Section 5.5 concludes with some final remarks.

5.1 Problem definition and formulations

The VRPRDL is formally defined as follows. Let $G = (N, A)$ with $N = \{0, 1, \dots, n\}$ be a complete directed graph in which node 0 corresponds to the depot and the remaining nodes correspond to the locations of interest. Each arc $(i, j) \in A$ has an associated travel time t_{ij} and cost w_{ij} both satisfying the triangle inequality. The set of customers that require a delivery during the planning period $[0, T]$ is represented by C . The delivery for a customer $c \in C$ is characterized by a demand quantity d_c and a geographic profile which specifies where and when a delivery can be made. Let $N_c \subseteq N$ denote the set of locations that customer c will visit during the planning horizon. By duplicating locations, we may assume $N_c \cap N_{c'} = \emptyset$ for different customers $c, c' \in C$. Note that we can express the set of nodes as $N = N_0 \cup \{i \in N_c \mid c \in C\}$, where $N_0 = \{0\}$. The locations $i \in N_c$ for $c \in C$ have non-overlapping time windows $[e_i, l_i]$ during which the delivery can take place and correspond to the customer's vehicle itinerary during the planning horizon. We use $c(i)$ to denote the customer associated with location i and we let $c(0) = 0$. A fleet of m homogeneous vehicles, each with capacity Q , is available to make deliveries; vehicles start and end their delivery routes at the depot. The goal is to find a set of delivery routes visiting each customer at one of the locations in the customer's itinerary, during the time that the customer is at that location, and such that the demand delivered on a route is no more than Q , the duration of a route does not exceed T , and the total cost is minimized.

The basic version of the VRPRDL is static and deterministic, i.e., it is assumed that all customer locations and the time spent at these locations are known with certainty for the entire planning horizon. Let x_{ij} be a binary variable indicating whether arc $(i, j) \in A$ is used or not. We write $x(A') = \sum_{(i,j) \in A'} x_{ij}$ for $A' \subseteq A$ and we use $\delta^-(i)$ and $\delta^+(i)$ to denote the sets of incoming and outgoing arcs of node i , respectively. The basic version of the VRPRDL can be formulated as follows:

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} w_{ij} x_{ij} \\ & x(\delta^-(i)) - x(\delta^+(i)) = 0 \quad i \in N, \end{aligned} \quad (5.1)$$

$$\sum_{i \in N_c} x(\delta^-(i)) = 1 \quad c \in C, \quad (5.2)$$

$$x(\delta^+(0)) \leq m, \quad (5.3)$$

$$s_j \geq s_i + t_{ij}x_{ij} + (e_j - l_i)(1 - x_{ij}) \quad (i, j) \in A, j \neq 0, \quad (5.4)$$

$$s_i + t_{i0}x_{i0} \leq \min\{l_i + t_{i0}, T\} \quad (i, 0) \in A, \quad (5.5)$$

$$e_i \leq s_i \leq l_i \quad i \in N, \quad (5.6)$$

$$y_j \geq y_i + d_{c(j)} - Q(1 - x_{ij}) \quad (i, j) \in A, j \neq 0, \quad (5.7)$$

$$d_{c(i)} \leq y_i \leq Q \quad i \in N, \quad (5.8)$$

$$x_{ij} \in \{0, 1\} \quad (i, j) \in A,$$

where s_i is the arrival time at location i and y_i is the cumulative quantity delivered by a vehicle making a delivery at location i when it is leaving location i . Note that the values of the y_i variables are relevant only for those locations that are actually visited. The objective is to minimize the total cost of the delivery routes. Vehicle flow conservation at every location is captured by Constraints (5.1). Constraints (5.2) guarantee that each customer receives a delivery at exactly one of the locations in his/her itinerary. Constraints (5.3) limit the number of vehicle departures from the depot to at most m . Constraints (5.4)–(5.6) determine the arrival times at each of the locations while ensuring that all vehicles return to the depot by time T and the time windows at the locations are respected. At the same time, these constraints prevent subtours from occurring. Constraints (5.7) and (5.8) ensure that the required quantities are delivered to the customers (in combination with Constraints (5.2)) and that the capacity of the vehicles is respected. Note that this formulation is slightly different from the one presented in [57].

The VRPRDL can also be formulated as a set partitioning problem by applying Dantzig-Wolfe decomposition to the formulation above. Let R denote the set of all feasible delivery routes (i.e., respecting capacity and time window constraints), let w_r be the cost of route $r \in R$, and let a_{ir} for every $i \in N$ and $r \in R$ indicate whether location i is visited on route r ($a_{ir} = 1$) or not ($a_{ir} = 0$). The set partitioning

formulation of the VRPRDL is as follows:

$$\min \sum_{r \in R} w_r z_r \quad (5.9)$$

$$\sum_{r \in R} \sum_{i \in N_c} a_{ir} z_r = 1 \quad c \in C, \quad (5.10)$$

$$\sum_{r \in R} z_r \leq m, \quad (5.11)$$

$$z_r \in \mathbb{Z}_+ \quad r \in R, \quad (5.12)$$

where z_r is the number of times route r is used. This formulation has exponentially many variables as the number of routes is exponential in the size of N . Therefore, we use column generation to solve its LP relaxation, which will be referred to as the master problem from now on. Note that since the arc costs satisfy the triangle inequality, we can replace $\sum_{r \in R} \sum_{i \in N_c} a_{ir} z_r = 1$, $c \in C$ with $\sum_{r \in R} \sum_{i \in N_c} a_{ir} z_r \geq 1$, $c \in C$ and still obtain a solution in which each customer is visited exactly once. This restricts the associated dual variable to be nonnegative, which typically leads to faster convergence of the column generation procedure.

5.1.1 Pricing problem

Let $\bar{R} \subset R$ be such that there exists a feasible solution to the master problem when $z_r = 0$ for all $r \in R \setminus \bar{R}$. A formulation involving only routes in \bar{R} is called a restricted master problem (RMP). Once RMP is solved to optimality, we check if there exists a column with negative reduced cost with respect to the original master problem. Such a column is a route r for which the following condition is satisfied:

$$w_r - \lambda_0^* - \sum_{c \in C} \sum_{i \in N_c} a_{ir} \lambda_c^* < 0. \quad (5.13)$$

Note that since $w_r = \sum_{(i,j) \in r} w_{ij}$ and $\lambda_0^* + \sum_{c \in C} \sum_{i \in N_c} a_{ir} \lambda_c^* = \sum_{i \in N} a_{ir} \lambda_{c(i)}^*$, we can rewrite the condition as:

$$\sum_{(i,j) \in r} w_{ij} - \sum_{i \in N} a_{ir} \lambda_{c(i)}^* < 0. \quad (5.14)$$

Thus, the pricing problem is an elementary shortest path problem with time window and capacity constraints (ESPPTWCC), where the cost of arc (i, j) is set to $w_{ij} - \lambda_{c(i)}^*$ for $i \in N_c$ and $j \in N \setminus N_c$, i.e., the goal is to find an elementary path of shortest length starting and ending at the depot and respecting capacity and time window constraints. The ESPPTWCC can be formulated as follows:

$$\min \sum_{(i,j) \in A} (w_{ij} - \lambda_{c(i)}^*) x_{ij}$$

$$x(\delta^-(0)) = 1, \tag{5.15}$$

$$x(\delta^+(0)) = 1, \tag{5.16}$$

$$x(\delta^-(i)) - x(\delta^+(i)) = 0 \quad i \in N \setminus \{0\}, \tag{5.17}$$

$$\sum_{i \in N_c} x(\delta^-(i)) \leq 1 \quad c \in C, \tag{5.18}$$

$$\sum_{c \in C} d_c \sum_{i \in N_c} x(\delta^+(i)) \leq Q, \tag{5.19}$$

$$s_j \geq s_i + t_{ij} x_{ij} + (e_j - l_i)(1 - x_{ij}) \quad (i, j) \in A, j \neq 0, \tag{5.20}$$

$$s_i + t_{i0} x_{i0} \leq \min\{l_i + t_{i0}, T\} \quad (i, 0) \in A, \tag{5.21}$$

$$e_i \leq s_i \leq l_i \quad i \in N, \tag{5.22}$$

$$x_{ij} \in \{0, 1\} \quad (i, j) \in A.$$

Constraints (5.15)–(5.17), together with subtour elimination constraints (5.20), define a path starting from and ending at the depot. Constraints (5.18) force this path to be elementary, i.e., they guarantee that each customer is visited at most once. Constraints (5.19)–(5.22) enforce the capacity and time window restrictions.

5.1.2 Solving the pricing problem

Solving a mixed integer programming problem using an off-the-shelf solver at every pricing iteration is computationally expensive. Hence, we adopt the iterative label setting algorithm proposed by [132] to solve the ESPPTWCC. Given a directed

graph with arbitrary arc lengths and a specified pair of source and sink nodes s and t , ESPPTWCC is the problem of finding the shortest resource-feasible path from s to t . As implied by its name, we have two different resources in ESPPTWCC arising from time window and capacity restrictions. Hence, a resource-feasible path in our context is one that respects the availability of time and capacity resources. Moreover, this path should also be elementary, i.e., free of cycles. A common way to ensure that an elementary resource-feasible path is obtained at the end of a label setting algorithm is to use node-visit resources, which were introduced by [133]. A node-visit resource is a resource with a capacity of one unit, which is consumed when the associated node is visited. Note, however, that the definition of an elementary path is slightly different in the case of the VRPRDL, because multiple locations are associated with a single customer. If nodes i and j are both locations associated with customer $c \in C$, then a path containing both i and j should not be considered elementary because it visits customer c at least twice. Therefore, instead of using node-visit resources, we maintain a customer-visit resource for each customer, which is consumed when any one of his associated locations is included in the path.

Label setting is a dynamic programming approach which constructs resource-feasible paths originating at a source node s and ending at a sink node t . Starting with the trivial path containing only the source node s , the label setting procedure extends unprocessed partial paths along all feasible arcs to create new (partial) paths. A state is associated with each partial path capturing the cost and the resource consumptions along the path. The extension of a partial path along an arc is infeasible if the resulting path would not be resource-feasible or if it cannot be augmented to reach the sink node within the available resource limits.

The efficiency of a label setting algorithm depends on the dominance relation that is used to eliminate partial paths. Let P_{si} and P'_{si} be two distinct paths from the source node s to node i with their respective states given by the label vectors \mathbf{U} and \mathbf{U}' . Suppose that the first element of the label vector represents the cost and the remaining elements represent the resource consumptions (in the same order for both label vectors) along the corresponding path. Path P_{si} is said to dominate path P'_{si} if $\mathbf{U} \neq \mathbf{U}'$ and $U_k \leq U'_k$ for $k = 1, \dots, K + 1$, where K is the number of resources.

To be able to eliminate additional partial paths using the above dominance relation, [133] introduce the notion of *unreachable nodes*. When creating a new partial path, the idea is to consume not only the node-visit resources for the nodes in the partial path itself, but also the node-visit resources of nodes that are not in the path, but cannot be reached by any feasible extension of the current partial path. Because we use customer-visit resources (rather than node-visit resources), we consider the unreachability of customers and say that customer c is unreachable if every node $i \in N_c$ is unreachable. Thus, the following resource consumption rule is adopted in our algorithm when consuming customer-visit resources. The resource corresponding to customer c in a partial path is consumed either when a node $i \in N_c$ is in the path, or else when none of the nodes from N_c can be contained in any feasible extension of the current partial path.

As the label associated with a partial path is a vector representing the state of the path, i.e., its cost and resource consumptions, the size of the state-space increases with the number of resources. In order to limit the size of the state-space and to accelerate the label-setting algorithm, [132] suggest to start the solution procedure with a state-space relaxation in which node-visit resources are initially not present. When the label setting procedure ends, the multiplicity of each node (number of times each node is visited) in the optimal path is computed and node-visit resources are introduced for some or all of the nodes with multiplicity greater than one. The label setting procedure is iteratively executed, each time starting with the original resources and the set of all node-resources introduced during the previous iterations, until the optimal path returned at the end of an iteration is elementary. The advantage of this state-space augmenting approach is that the number of node-visit resources introduced to obtain a least-cost elementary path is usually much smaller than the number of nodes.

To be able to provide details of the state-space augmenting approach of [132], we introduce some additional notation. Let S be the set of critical customers, i.e., the customers for which a customer-visit resource is maintained. We denote the multiplicity of customer c on path p , i.e., the number of times a location $i \in N_c$ is visited on path p , by $M_c(p)$. We start with $S = \emptyset$ and update S at the end of each

label setting iteration that did not return an elementary optimal path by adding the customer with the highest multiplicity. In case of ties, we add the customer with the smallest index. Now suppose that the state corresponding to a given path p is described by the label vector $\mathbf{U} = (U_0, \dots, U_{|S|+2})$. We assume that U_0 , U_1 , and U_2 specify the cost of p , and the consumption of the time and capacity resource along p , respectively. The remaining elements of \mathbf{U} indicate the consumption of customer-visit resources in the order that they were added to S . For example, if c_k is the k^{th} customer added to S , then U_{k+2} shows whether customer c_k is included in the path p or not.

To keep track of the resource consumptions, we define h_{ij}^r to be the amount of resource r consumed when arc (i, j) is used ($h_{ij}^1 = t_{ij}$ and $h_{ij}^2 = d_j$). When extending a label \mathbf{U}^i through arc (i, j) to create a new label \mathbf{U}^j , we set $U_1^j = \max\{U_1^i + t_{ij}, e_j\}$, because if the delivery vehicle arrives at location j before e_j , then it has to wait for customer $c(j)$, and $U_2^j = U_2^i + d_j$. We define R_i^r to be the limit of resource r at node i . The need to have a node index in our resource limit definition is the presence of time windows, since the closing time of a window is different for each node and we have to respect these limits while constructing resource-feasible paths. Even though the capacity resource limit is independent of the node and is equal to Q , we keep the node index for generality. Finally, let L_i denote the set of all labels associated with the efficient (i.e., non-dominated) partial paths ending at node i , and let the source s and the sink t correspond to the depot.

Algorithm 1: *State-space augmenting algorithm*

```

Set  $S = \emptyset$ 
do
     $\mathcal{P}^* = \text{labelSetting}(S)$ 
    Find  $p^* \in \mathcal{P}^*$  with shortest length
    Find the customer  $c$  having the highest multiplicity on path  $p^*$ 
    if  $M_c(p^*) > 1$  then
         $S \leftarrow S \cup \{c\}$ 
while  $p^*$  is not elementary
Return  $p^*$ 

```

Let L_i denote the set of all labels associated with the efficient (i.e., non-dominated) partial paths ending at node i , and let the source s and the sink t correspond to the depot.

Algorithm 2: *labelSetting(S)*

Data: Source node s , sink node t , resource limits R_i^r for $i \in N$ and $r \in \{1, 2\}$, resource consumptions h_{ij}^r for $(i, j) \in A$ and $r \in \{1, 2\}$, opening of time window e_i for every $i \in N$

Result: The set \mathcal{P}^* of all paths corresponding to non-dominated labels on the sink node t

Initialization

Let $L_0 = \{(0, 0, 0)\}$, $L_i = \emptyset$ for all $i \in N \setminus \{s\}$, and $L = \bigcup_{i \in N} L_i$.

Label selection and treatment

while $L \neq \emptyset$ **do**

Select $i \in N$ and $\mathbf{U}^i \in L_i$ so that \mathbf{U}^i is the lexicographically minimal label in L

$L \leftarrow L \setminus \{\mathbf{U}^i\}$

for $(i, j) \in A$ **do**

if *canExtendLabel*(i, \mathbf{U}^i, j) **then**

$\mathbf{U}^j = \text{extendLabel}(i, \mathbf{U}^i, j)$

performDominanceCheck(\mathbf{U}^j, j)

Return \mathcal{P}^*

Subroutine 3: *canExtendLabel*(i, \mathbf{U}^i, j)

Returns *false* if the extension of label \mathbf{U}^i along (i, j) is not resource-feasible. Note that the feasibility of extension along (i, t) is established before creating \mathbf{U}^i

```
if  $j \neq t$  then
  if  $c(j) \in S$  then
    Let  $k$  be the order of customer  $c(j)$  in  $S$ 
    if  $U_{k+2}^i = 1$  then
      Return false
    if  $\max\{U_1^i + h_{ij}^1, e_j\} > R_j^1$  or  $\max\{U_1^i + h_{ij}^1, e_j\} + h_{jt}^1 > R_t^1$  then
      Return false
    else if  $U_2^i + h_{ij}^2 > R_j^2$  then
      Return false
    else
      Return true
else
  Return true
```

Subroutine 4: $extendLabel(i, \mathbf{U}^i, j)$

Extends the label \mathbf{U}^i through arc (i, j) by updating the cost component and resource consumptions, and then returns the new label

$$U_0^j \leftarrow U_0^i + (w_{ij} - \lambda_{c(i)})$$

$$U_1^j \leftarrow \max\{U_1^i + h_{ij}^1, e_j\}$$

$$U_2^j \leftarrow U_2^i + h_{ij}^2$$

if $j \neq t$ **then**

if $c(j) \in S$ **then**

 Let k be the order of customer $c(j)$ in S

$$U_{k+2}^j \leftarrow 1$$

For strong dominance, consume the visitation resource for each customer in S that is not on the partial path extended to j , and yet is unreachable from node j

for $c \in S$ **do**

 Let l be the order of customer c in S

if $U_{l+2}^j = 0$ and $isReachable(j, \mathbf{U}^j, c) = false$ **then**

$$U_{l+2}^j \leftarrow 1$$

Return \mathbf{U}^j

Subroutine 5: $isReachable(i, \mathbf{U}^i, c)$

Returns true if any of the locations corresponding to customer c is reachable from node i by extending the partial path associated with the label \mathbf{U}^i

for $j \in N_c$ **do**

if $canExtendLabel(i, \mathbf{U}^i, j)$ **then**

 Return *true*

Return *false*

Subroutine 6: *performDominanceCheck*(\mathbf{U}^i, i)

If the label \mathbf{U}^i is dominated by an existing label, then this method returns false; otherwise it adds \mathbf{U}^i to L_i and L , removes the labels dominated by \mathbf{U}^i from L_i and L (if any such label is found), and returns true

```
for  $\mathbf{U} \in L_i$  do
  if  $\mathbf{U}^i \neq \mathbf{U}$  then
    if  $\mathbf{U}$  dominates  $\mathbf{U}^i$  then
      Return false
    else if  $\mathbf{U}^i$  dominates  $\mathbf{U}$  then
       $L_i \leftarrow L_i \setminus \{\mathbf{U}\}$ 
       $L \leftarrow L \setminus \{\mathbf{U}\}$ 
    else
      Continue
  else
    Break the loop
```

$L_i \leftarrow L_i \cup \{\mathbf{U}^i\}$

```
if  $i \neq t$  then
   $L \leftarrow L \cup \{\mathbf{U}^i\}$ 
```

Return true

Algorithm 1 represents a straightforward implementation of the state-space augmenting approach that solves the ESPPTWCC optimally. Despite being more efficient than using off-the-shelf software to solve the integer programming formulation of the pricing problem, employing a straightforward implementation of the state-space augmenting algorithm to solve the pricing problem may still be (too) time-consuming. However, to solve the master problem there is no need to identify a most negative reduced cost column at every pricing iteration; identifying any negative reduced cost column suffices. Finding a most negative reduced cost column is typically needed only towards the end of the column generation process, because few, if any, negative reduced cost columns exist at that time. Consequently, we invoke Algorithm 1 in

our implementation only if no negative reduced cost columns can be detected heuristically, and even in that case we terminate the algorithm as soon as a pre-specified number of negative reduced cost columns is constructed by the algorithm.

5.2 A branch-and-price algorithm

We develop a branch-and-price algorithm to solve the VRPRDL, i.e., a branch-and-bound algorithm in which at each node of the search tree the LP relaxation is solved using column generation. The most time consuming component of a branch-and-price algorithm is typically the solution of the pricing problem. Therefore, the efficient detection of negative reduced cost columns is critical to the performance of any branch-and-price algorithm. In the following, we present the techniques we employ to speed up the solution of the pricing problem, we describe the adopted branching scheme, and we discuss how we deal with the tailing-off effect.

5.2.1 Heuristic pricing

It is not necessary to return a most-negative reduced cost column in each pricing iteration, it suffices to return (at least one) negative reduced cost column, if one exists. Even though the change in the value of the solution to the restricted master problem, when adding any negative reduced cost column rather than a most-negative reduced cost column, may be smaller, and more pricing iterations may have to be performed to reach an optimal solution, the reduction in solution time in each pricing step typically reduces the overall computation time.

A simple application of the above idea is based on the observation that usually many elementary negative reduced cost paths are found during the first few label setting iterations. Therefore, in our implementation of the state-space augmentation algorithm, we collect elementary negative reduced cost paths found in each iteration

and will sometimes terminate the algorithm prematurely, i.e., before a most-negative reduced cost column has been found, and return all elementary negative reduced cost columns collected.

Another observation leads to a second useful idea: as the dual values are updated after each pricing iteration, some elementary paths with a non-negative reduced cost in one pricing iteration may have a negative reduced cost in a subsequent iteration. Therefore, at the end of a pricing iteration, non-dominated elementary paths with non-negative reduced costs found in the last label setting iteration are kept in a column pool. At the start of each pricing iteration, the columns in the pool are evaluated to see if they (now) have a negative reduced cost. To ensure that it does not become too costly to explore the column pool, we limit its size, i.e., we keep a maximum number of columns. At the end of a pricing iteration, if we add elementary non-negative reduced cost columns to the pool, we check its size, and, if the maximum pool size is exceeded, the oldest columns are removed until the desired pool size is reached.

Another strategy to detect negative reduced cost columns quickly is to make use of heuristics before invoking the exact pricing algorithm. To this end, we implement a truncated search version of the state-space augmenting approach. Instead of maintaining all non-dominated labels and their associated partial paths, we store only a small number of such labels at each node to speed up the search procedure. More specifically, we keep only a pre-specified number of efficient labels per node, and each time a new label is added to the list of efficient labels of a node, we discard the one with the largest cost if the number of labels exceeds the limit. In this way, fewer labels are treated at each label setting iteration which can facilitate faster detection of negative cost elementary paths. Again, excluding a part of the solution space may come at the expense of performing more iterations, i.e., there is a trade-off between the number of efficient labels maintained and the number of label setting iterations performed by the state-space augmenting approach.

Briefly, we try to identify negative reduced cost columns first by exploring the

column pool, then by invoking the truncated-search version of the state-space augmentation algorithm, and finally by invoking the full-search version of the state-space augmentation algorithm when the other approaches fail. To control the time spent in pricing iterations even further, we terminate any pricing iteration as soon as a predetermined number of elementary negative reduced cost columns has been found.

Finally, we keep track of the minimum cost γ of the elementary paths detected during the search, which gives an upper bound on the optimal value of the ESPPTWCC, and the cost η of the optimal path obtained at the end of each label setting iteration, which gives a lower bound on the optimal value of the ESPPTWCC. Once the gap between these bounds is closed, we can conclude that the pricing problem has been solved optimally.

As mentioned earlier, the state-space augmenting approach starts without any customer-visit resources. However, the customers added to the set S of critical customers during a pricing iteration are likely to be visited more than once in subsequent pricing iterations, and clearing S at the end of each pricing iteration may therefore result in an increase in the number of label setting iterations. Consequently, rather than initializing the algorithm with $S = \emptyset$ each time, we retain the set of critical customers throughout the column generation process at a node of the search tree. We set $S = \emptyset$ only at the start of processing a node in the search tree.

5.2.2 Bidirectional search

A common technique used to speed up the solution of the pricing problem is bidirectional search, in which paths, consuming around half of the resources available, are constructed from both the source and the sink and then merged to obtain complete paths. Even though our implementation of bidirectional search provided efficiency gains when solving the pricing problem exactly, the overall performance of the branch-and-price algorithm deteriorated. We speculate that the reason is that we infrequently solve the pricing problem exactly and that the columns returned by the

truncated search, which can be substantially different when deploying bidirectional search, are not as effective.

5.2.3 Branching

When the optimal solution to the master problem is fractional, we have to perform branching to find integer solutions. We branch on the arc variables, x_{ij} , in the original formulation. If an arc variable has a fractional value, we force the arc to be a part of the solution in one branch while prohibiting routes containing that arc in the other branch. Branching on variables in the original formulation has become a standard feature of many branch-and-price algorithms [134].

Each variable of the master problem corresponds to the number of times a particular route is used in the optimal delivery plan. Therefore, we compute the value of an arc (i, j) in an optimal solution to the master problem by summing the optimal values of the routes that include the arc (i, j) . Once a branching decision is enforced, we have to ensure that the columns in the restricted master problem and the ones that will be returned by the pricing subroutine are compatible with this decision. To guarantee compatibility at a child node, first we filter the columns coming from the restricted master problem associated with its parent node to exclude those that are in conflict with the branching decision and then update the pricing problem by forbidding the proper arc(s).

Ensuring compatibility is rather straightforward when the branching decision requires prohibiting a certain arc (i, j) , i.e., the columns containing this arc are removed from the restricted master problem and the arc (i, j) is ignored while solving the pricing problem. On the other hand, to ensure that (i, j) is included in the solution, we omit all columns (routes) containing any arc from the set $(\delta^+(i) \setminus (i, j)) \cup (\delta^-(j) \setminus (i, j)) \cup (\delta^-(l) : l \in (N_{c(i)} \setminus \{i\}) \cup (N_{c(j)} \setminus \{j\}))$ from the restricted master problem, and discard all the arcs in this set while solving the pricing problem (i.e., we do not extend partial paths through these arcs). Note that as

a result of column filtering, the master problem may become infeasible. Therefore, we always maintain a set of artificial columns with a high cost that guarantee the feasibility of the master problem. More specifically, we store the columns used to initialize the restricted master problem, and introduce these columns as “artificial” columns prior to starting column generation at a node of the search tree. By assigning the sum of the costs of these columns to each one of them individually, we ensure that these columns will not be in the optimal basis when column generation completes.

We have considered three strategies for selecting the arc to branch on. The first one is conventional branching, denoted by *CB*, where we select an arc with value closest to 0.5. The second one is to choose an arc that appears in the greatest number of routes in the solution to the master problem, denoted by *MF*. The last one is to choose an arc with a fractional value that occurs the earliest in time, denoted by *ED*. In particular, for an arc (i, j) , we determine the time at which a vehicle using this arc in its route departs from node i . There may be multiple routes containing (i, j) , in this case, we take the minimum of the departure times from node i over all such routes. In all of the arc selection strategies, we have the additional restrictions that the value of the chosen arc should be less than one and both of its endpoints should correspond to customer locations.

In *CB*, if there is more than one arc whose value is closest to 0.5, then we pick the first one encountered. In *MF*, we break ties by choosing either the arc encountered first during the search (*MF*) or the arc whose value is closest to 0.5 (*MFC*). Similarly, for *ED*, we either select the arc encountered first during the search (*ED*) or the arc whose value is closest to 0.5 (*EDC*).

5.2.4 Initial set of columns and feasible solutions

The column generation method starts by solving a restriction of the master problem. Therefore, we need to provide an initial set of columns that guarantees the feasibility

of the master problem, i.e., a feasible starting solution. The quality of this solution can have a significant impact on the performance of the branch-and-price algorithm, especially for large problem instances. In our experiments, we use the feasible solution found by the heuristic of [57]. The heuristic constructs a feasible solution within a few seconds for small and medium size instances and within a few minutes for large instances. The time spent by the heuristic on improving the initial feasible solution depends on the quality of that solution, but is small compared to the time spent by our branch-and-price algorithm.

The value of any feasible solution provides an upper bound on the optimal objective function value and can thus be used to fathom nodes by bound. Of course, the higher the quality of the feasible solution, the more effective the fathoming becomes. Therefore, we also embed the following commonly used heuristic for producing a, hopefully high-quality, feasible solution. After solving the master problem at the root node, we solve the restricted master problem, i.e., including initial as well as generated columns, as an integer program (simply handing it to an off-the-shelf software package). This heuristic has proven to be quite successful in a number of different applications, and initial experimentation in our setting revealed that the resulting integer programs could be solved quickly, and, thus, did not impede the overall solution process.

5.2.5 Handling the tailing-off effect

Many pricing iterations may have to be performed with little or no improvement in the objective function value towards the end of the column generation process at a node in the search tree. This situation is quite common in branch-and-price algorithms and is known as the tailing-off effect. Below, we discuss two approaches for dealing with the tailing-off effect.

5.2.5.1 Using Lagrangian dual bounds for early pruning

The occurrence of tailing-off at a node is especially unfortunate if the node is fathomed by bound once the master problem has been solved, because in that case much time may have been “wasted” by “unnecessarily” solving pricing problems. This situation can, possibly, be prevented if an alternative lower bound can be computed that can be used to fathom the node before the column generation process at the node completes. Such a bound can be computed using concepts from Lagrangian relaxation. Note, however, that this may also mean that fewer columns are added to the column pool, which can have a negative impact on solution efficiency.

Dualizing the covering constraints in the master problem, we obtain the Lagrangian relaxation

$$\begin{aligned} \min \quad & \sum_{r \in R} w_r z_r + \sum_{c \in C} \lambda_c \left(1 - \sum_{r \in R} \sum_{i \in N_c} a_{ir} z_r\right) \\ \text{s.t.} \quad & \sum_{r \in R} z_r \leq m, \end{aligned} \tag{5.23}$$

$$z_r \geq 0, \quad r \in R, \tag{5.24}$$

which provides a lower bound on the optimal value of the master problem for any nonnegative vector of multipliers $(\lambda_1, \dots, \lambda_{|C|})$. Rearranging gives

$$\sum_{c \in C} \lambda_c + \left\{ \min \sum_{r \in R} (w_r - \sum_{i \in N \setminus \{0\}} a_{ir} \lambda_{c(i)}) z_r \right. \\ \left. \text{s.t. (5.23), (5.24)} \right\},$$

which, when taking the values of the optimal dual solution to RMP, gives

$$\sum_{c \in C} \lambda_c + \left\{ \min \sum_{r \in R} (\bar{w}_r + \lambda_0) z_r \right. \\ \left. \text{s.t. (5.23), (5.24)} \right\},$$

where \bar{w}_r is the reduced cost of route r and λ_0 is the value of the dual variable corresponding to (5.23). This quantity can be bounded from below by

$$\sum_{c \in C} \lambda_c + m(\bar{w}_{min} + \lambda_0) = \sum_{c \in C} \lambda_c + m\lambda_0 + m\bar{w}_{min} = \theta_{RMP}^* + m\bar{w}_{min},$$

where \bar{w}_{min} is the minimum reduced cost and θ_{RMP}^* is the optimal value of RMP.

This lower bound can be computed easily at the end of a column generation iteration if the optimal value of the pricing problem is known. (Note that θ_{RMP}^* always provides an upper bound on the optimal value of the master problem, and that when $\bar{w}_{min} = 0$ it provides a lower bound as well, in which case the master problem is solved optimally.) Observe that it is possible to obtain a lower bound on the optimal value of the master problem as soon as a complete label setting iteration is performed, because the state-space augmenting algorithm yields a lower bound on the optimal value of the pricing problem at the end of each label setting iteration. Therefore, whenever the full-search version of the state-space augmentation algorithm is used in a pricing iteration, we compute a lower bound for the master problem at the end of each label setting iteration and see if it can be used to prune the node.

5.2.5.2 Early branching

Another strategy to deal with the tailing-off effect is to prematurely terminate the column generation procedure and perform branching early. To accomplish this, one can stop generating columns and apply branching, for example, when the improvement in the objective function value is less than ϵ in the last Δ iterations. Note that the values of ϵ and Δ need to be set carefully, because branching too aggressively may grow the search tree significantly and may be counterproductive.

5.2.6 Implementation details

There are many algorithmic choices and parameter settings that impact the computational performance of the branch-and-price algorithm. In the following, we first summarize our implementation of a straightforward branch-and-price algorithm and then describe the algorithmic choices and parameter settings for the enhanced version

that incorporates the ideas discussed above.

5.2.6.1 Straightforward branch-and-price

In order to evaluate the improvements achieved by our enhanced branch-and-price algorithm, we consider a straightforward approach, in which out-and-back routes from the depot to the first location of every customer are used to define the initial set of columns (these routes always correspond to a feasible solution because $m = |C|$ in VRPRDL instances), the state-space augmenting algorithm is initialized with $S = \emptyset$ at every pricing iteration, only a single column with the most negative reduced cost is added to the restricted master problem (i.e., the pricing problem is solved to optimality), and the conventional branching strategy (CB) is employed; there is no early pruning, no early branching, and the restricted master problem at the end of the column generation process at the root node is not used to seek a, possibly improved, feasible solution.

5.2.6.2 Enhanced branch-and-price

In our enhanced branch-and-price algorithm, the following parameter settings control the solution of the pricing problem: the column generation process terminates as soon as β elementary negative reduced cost columns are identified. The truncated-search version of the state-space augmentation algorithm restricts the number of non-dominated labels at each node to α . In the truncated-search version of the state-space augmentation algorithm, we also monitor the cost γ of the shortest elementary path detected so far, which yields an upper bound on the optimal ESPPTWCC value, and the cost η of the shortest path obtained at the end of each truncated label setting iteration, which provides a lower bound on the cost of the shortest elementary path that can be obtained by truncated-search, and terminate the truncated search when these bounds at the end of a label setting iteration are equal. The size of the column pool is 1000, i.e., at most 1000 columns are stored at any one time. Only

when neither the exploration of the column pool nor the truncated search produces any elementary negative reduced cost columns do we invoke exact pricing. The set S of critical customers used in the state-space augmentation algorithm is retained throughout the column generation process at each node in the search tree, and it is cleared right before column generation starts at another node.

In some of our computational experiments and for some values of α and β , we also consider forcing the truncated search to stop upon completing a single label setting iteration even when the number of negative reduced cost columns detected is less than β and the gap between γ and η is not closed at the end of this label setting iteration. We use T to denote that we adopt this termination criterion, i.e., stop at the end of the first label setting iteration, instead of iterating until γ and η become equal, which is denoted by F .

Furthermore, the solution obtained by the heuristic of [57] is used to initialize the algorithm, the restricted master problem is solved as an integer program upon completing column generation at the root node in the hope of finding an improved feasible solution, early pruning is active, and the nodes in the branch-and-price tree are evaluated in a depth-first order (best-bound and depth-first search produce similar results on small and medium size instances, the latter performs better for large size instances). Early branching is not used as computational experiments revealed that the benefits are negligible.

Finally, we want to point out that our branch-and-price algorithm can also solve instances where arc costs or travel times do not satisfy the triangle inequality. In the former case, one can simply use the partitioning constraints given by (5.10) when defining the master problem. In the latter case, one has to compute the shortest travel time between each pair of locations prior to executing the branch-and-price algorithm, because this information is used in certain steps of the state-space augmenting method when solving the pricing problems and in preprocessing the problem graph to reduce its size.

5.3 Incorporating a home delivery option

Trunk delivery was introduced in the hope that it would create cost-saving opportunities compared to making home deliveries. It is easy to construct examples where this is indeed the case. However, it is also easy to construct examples where this is not the case and the cost actually increases. Thus, companies that are considering trunk delivery will likely deploy a hybrid model in which deliveries can either be made at the home location of the customer (during the entire planning horizon) or to trunk of the customer's car at one of the locations in the car's itinerary, if this creates cost savings.

Fortunately, our branch-and-price algorithm can be used for solving this hybrid model by simply making slight changes in the instance data. In particular, we replace the time windows associated with the home location of each customer, which, in the instances of VRPRDL, are the first and last location of the car's itinerary by a single location with time window $[0, T]$, where T is the length of the planning horizon. Note that this means that the time windows of the locations visited by a customer are no longer non-overlapping. However, the branch-and-price algorithm has no components that exploit or rely on this non-overlapping property and thus it can be applied without any modification. Of course the performance may (and, as we will see, will) deteriorate.

5.4 Computational study

We conduct a computational study to (1) evaluate the performance of our branch-and-price algorithm, and to (2) assess the benefits of employing trunk delivery services.

5.4.1 Instances and preprocessing

In our computational experiments, we use two sets of VRPRDL instances.

The first set consists of slightly modified versions of the 40 random instances introduced in [57], in which the travel times have been adjusted to ensure that they satisfy the triangle inequality and, when necessary, the time windows at locations have been adjusted accordingly. These instances range in size from 15 to 120 customers, each with up to 5 roaming delivery locations. This set of instances is well-suited to assess the impact of the number of customers and roaming delivery locations on the performance of the branch-and-price algorithm.

The second set contains two variations of 10 medium-size instances generated in order to investigate the impact of the distance of the roaming delivery locations to the depot on the performance of the branch-and-price algorithm. The distance of the roaming delivery locations to the depot may impact the performance of the branch-and-price algorithm for several reasons. When the roaming delivery locations are closer to the depot, this typically implies that the customers spend more time traveling, which in turn implies that there is less time available for making deliveries, i.e., the time windows at the roaming delivery locations are narrower. Narrower time windows may lead to more effective preprocessing, i.e., elimination of more variables. On the other hand, when the roaming delivery locations are closer to the depot, this typically implies that the roaming delivery locations of different customers are closer together, which in turn implies that there are more feasible solutions. More feasible solutions not only implies possibly lower costs, but may also lead to less effective preprocessing, i.e., elimination of fewer variables. The first variation of each instance is created using the instance generator described in [57], which chooses the roaming delivery locations for a customer uniform randomly in a circle with radius $sT/2\rho$ and center at the customer's home location, where s is the (constant) vehicle speed and ρ is the maximum number of locations per customer. In the second variation of each instance, the home location and the fraction of time spent at every roaming delivery location are the same, but the roaming delivery locations themselves tend

to be closer to the depot (which implies that the associated time window widths are usually different). More precisely, in the second variation, the roaming locations are chosen uniform randomly in a circle with radius $sT/2\rho$ and center on the line connecting the home location and the depot, either at the midpoint of the line or at distance $sT/2\rho$ of the home location, whichever is closer to the home location. (As with the first set of instances, the travel times and the time windows are adjusted to ensure that the travel times satisfy the triangle inequality.)

In all the instances, the planning horizon is 12 hours, the vehicle capacity is 750, and the geographic profile of every customer consists of at least one and at most six locations (the first and last location always being the home location of the customer - in case there is only one location, it signals that the customer is at home all day).

Although there is no restriction on the fleet size in the original instances, we assume that m , the number of vehicles available for making the deliveries, is equal to the number of routes in the solution provided by the heuristic of [57] plus one when solving VRPRDL and VRPHRDL instances with our enhanced algorithm. We impose this restriction, because smaller values of m results in stronger lower bounds on the optimal value of the master problem, and preliminary experiments using small to medium sized instances revealed that the optimal costs remain unchanged even if we set $m = |C|$. Characteristics of the first and second sets of instances are provided in Tables 5.1 and 5.2, respectively. For each instance, we present the number of customers, the average number of locations per customer, and, considering all customers, the minimum, average, and maximum distance from home location to the depot, the minimum, average, and maximum fraction of time available for making a delivery, and the average width of time windows. Note that for the second set of instances, the number of customers is not specified in the table as all instances in this set contain 40 customers. Also in Table 5.2 is an additional column presenting the average distance from the roaming delivery locations to the depot. We observe that there are noticeable differences between instances, with some instances having, on average, only 2.47 locations per customer whereas others have, on average, 4.33 locations per customer, and some instances having, on average, only 49% of the planning horizon available to make deliveries whereas others have, on average, 90%

of the planning horizon available.

We compute the Euclidean distance between each pair of locations based on their coordinates and then round this distance to the nearest integer. In order to ensure that arc costs satisfy the triangle inequality, we assign the shortest distance between each pair of nodes to the cost of the arc connecting these two nodes. Furthermore, we reduce the size of the network by eliminating nodes and arcs that cannot be a part of any feasible solution. Our preprocessing steps are as follows:

1. Eliminate node i and all arcs incident to this node if $t_{0i} > l_i$ or $e_i + t_{i0} > T$; and
2. Eliminate arc (i, j) if $\max\{t_{0i}, e_i\} + t_{ij} > l_j$ or $\max\{t_{0i}, e_i\} + t_{ij} + t_{j0} > T$

Essentially, we eliminate a node i if an out-and-back route from the depot to the node, i.e., route $0 - i - 0$, leads to a time window violation. Similarly, we eliminate an arc (i, j) if route $0 - i - j - 0$ is not time-feasible. This simple preprocessing is very effective, it reduces the number of nodes and arcs substantially: the minimum, average, and maximum reduction in the number of arcs across the VRPRDL instances are 72.53%, 87.49% and 93.1%, respectively. The reduction is smaller for the VRPHRDL instances as the home locations now have wide time windows, and thus fewer nodes and arcs are eliminated.

The algorithm is implemented in Java using the branch-and-price framework of Java OR library (*jORLib*) and Java graph theory library (*JGraphT*), and CPLEX 12.6.3 is employed for solving the restricted master problems through Concert Technology. All experiments are performed on a 64-bit machine with Intel Xeon E5-2650 v3 processor at 2.30 GHz. The time limit is set to two hours for instances with up to 60 customers and to six hours for instances with 120 customers. In the computational results tables, we will indicate that a time limit was reached with TL. Note that the solution times reported in the tables do not include the time spent by the heuristic of [57] to find the initial solution.

Table 5.1: Characteristics of the instances in the first set

Instance	Number of customers	Avg number of customer locations	Avg width of time windows	Distance to depot (from home location)			Fraction of time available for delivery		
				min	avg	max	min	avg	max
1	15	4.07	102.93	3	67.07	166	0.05	0.58	1
2	15	3.73	125.25	28	83.73	148	0.31	0.65	1
3	15	3.40	139.61	7	72.33	167	0.08	0.66	1
4	15	3.27	130.96	3	70.80	159	0.08	0.59	1
5	15	3.40	142.96	13	101.07	174	0.14	0.68	1
6	20	3.25	148.23	2	76.45	152	0.16	0.67	1
7	20	3.35	138.49	15	76.25	174	0.08	0.64	1
8	20	3.95	102.09	6	87.10	161	0.04	0.56	1
9	20	3.75	94.65	3	73.35	171	0.01	0.49	1
10	20	3.10	154.32	0	80.55	169	0.14	0.66	1
11	30	3.40	130.90	2	85.43	176	0.03	0.62	1
12	30	3.73	103.13	20	78.03	174	0.01	0.53	1
13	30	3.90	99.88	3	71.27	171	0.04	0.54	1
14	30	3.53	124.70	5	73.07	171	0.12	0.61	1
15	30	4.10	94.59	3	81.70	176	0.04	0.54	1
16	30	3.93	107.95	2	72.77	160	0.03	0.59	1
17	30	4.30	83.51	2	85.23	165	0.08	0.50	1
18	30	3.50	120.30	17	89.70	154	0.01	0.58	1
19	30	3.23	139.63	12	82.47	179	0.08	0.63	1
20	30	2.47	223.78	12	75.10	172	0.05	0.77	1
21	60	3.73	115.26	4	78.88	173	0.04	0.60	1
22	60	3.53	117.18	1	80.07	170	0.01	0.58	1
23	60	3.90	100.76	2	89.03	179	0.04	0.55	1
24	60	3.80	118.83	8	93.12	175	0.03	0.63	1
25	60	3.88	108.87	3	79.60	165	0.06	0.59	1
26	60	3.73	108.36	1	89.77	178	0.01	0.56	1
27	60	3.45	131.47	8	90.27	176	0.03	0.63	1
28	60	3.63	111.70	1	75.07	180	0.04	0.56	1
29	60	3.75	112.79	2	91.95	179	0.02	0.59	1
30	60	3.95	108.27	4	93.97	177	0.01	0.59	1
31	120	3.83	112.43	0	76.23	178	0.01	0.60	1
32	120	3.67	116.42	0	83.98	174	0.04	0.59	1
33	120	3.92	104.71	0	69.69	179	0.02	0.57	1
34	120	3.78	112.51	2	79.45	176	0.01	0.59	1
35	120	3.75	107.41	1	77.35	174	0.07	0.56	1
36	120	3.51	127.94	0	89.57	178	0.05	0.62	1
37	120	3.88	102.49	3	83.14	177	0.02	0.55	1
38	120	3.51	126.62	0	86.11	177	0.03	0.62	1
39	120	3.56	119.54	0	89.30	178	0.03	0.59	1
40	120	3.84	103.44	0	79.73	171	0.07	0.55	1

Table 5.2: Characteristics of the instances in the second set

Instance	Avg number of customer locations	Avg width of time windows	Avg distance from roaming locations to depot	Distance to depot (from home location)			Fraction of time available for delivery		
				min	avg	max	min	avg	max
41_v1	3.98	159.60	101.71	16	24.57	176	0.75	0.88	1
41_v2		149.79	80.72	16	24.50	175	0.66	0.83	1
42_v1	3.93	161.66	91.10	19	23.60	167	0.72	0.88	1
42_v2		151.25	68.85	18	23.59	167	0.70	0.82	1
43_v1	4.05	158.07	88.51	4	21.43	171	0.76	0.89	1
43_v2		144.85	69.19	4	21.44	171	0.57	0.81	1
44_v1	3.53	181.38	93.58	2	24.65	174	0.68	0.89	1
44_v2		173.61	72.11	2	24.61	174	0.62	0.85	1
45_v1	3.60	180.12	67.59	4	22.67	177	0.79	0.90	1
45_v2		166.60	48.31	4	22.69	177	0.59	0.83	1
46_v1	4.33	143.42	98.54	1	20.57	176	0.77	0.86	1
46_v2		131.63	79.35	1	20.52	176	0.65	0.79	1
47_v1	4.18	145.56	96.31	11	22.85	174	0.70	0.84	1
47_v2		139.58	77.20	11	22.87	173	0.67	0.81	1
48_v1	3.83	167.82	97.96	9	25.11	178	0.72	0.89	1
48_v2		153.76	76.83	9	25.08	178	0.53	0.82	1
49_v1	3.80	165.56	100.17	12	26.97	175	0.74	0.87	1
49_v2		148.48	72.55	12	26.96	175	0.51	0.78	1
50_v1	4.03	155.06	78.11	2	20.42	176	0.75	0.87	1
50_v2		146.07	60.85	2	20.48	177	0.68	0.82	1

5.4.2 Evaluating the performance of the branch-and-price algorithm

To be able to evaluate the benefits of the various techniques introduced in the branch-and-price algorithm, we start by solving the first set of instances with the straightforward implementation described in Section 5.2.6.1. The results can be found in Table 5.3. For each instance, we report the name of the instance, the cost of the initial solution, the cost of the best solution, the integrality gap, the solution time (in seconds), the total number of pricing iterations performed during the execution of the algorithm, and the number of nodes evaluated during the search, respectively. When the algorithm terminates within the time limit, the best solution is optimal, otherwise it may or may not be optimal. The integrality gap is computed as the ratio $(\theta_{IP}^* - \theta_{LP}^*)/\theta_{LP}^*$, where θ_{LP}^* is the optimal value of the master problem at the root node of the search tree and θ_{IP}^* is the cost of the best solution found during the execution of the algorithm. When the master problem at the root node cannot be solved within the time limit, it is not possible to compute an integrality gap, which

is indicated with a dash. If an instance is solved to optimality (i.e., the algorithm terminates within the time limit), the integrality gap provides some additional insight into the relative difficulty of the instance. If, on the other hand, an instance cannot be solved within the time limit, then the integrality gap provides a quality guarantee of the best solution found.

Table 5.3: Results with the straightforward BAP

Instance	Initial solution	Best solution	Integrality gap (%)	Solution time (s)	Iterations	Nodes
1	2074	901	0	0.58	39	1
2	2316	1286	0	0.27	29	1
3	2234	991	0	0.43	30	1
4	1982	1062	0	0.27	30	1
5	3322	1832	0	0.04	26	1
6	3328	1294	1.73	2.30	245	31
7	3204	1155	1.29	30.47	773	70
8	3170	1455	0	0.18	39	1
9	2838	1260	1.86	3.32	307	32
10	3270	1684	0	0.55	33	1
11	4932	1922	0.31	14.28	283	27
12	4610	2324	2.52	120.38	16914	2707
13	4868	1747	0	23.33	101	1
14	4084	1273	0	30.83	100	1
15	4656	1694	0	22.78	97	1
16	4770	1938	0	57.13	82	1
17	4502	1965	0.10	5.75	117	3
18	5392	1827	0	2.10	83	1
19	5286	2083	2.46	93.55	4140	413
20	4236	1822	0	78.70	103	1
21	10374	3761	0	606.60	216	1
22	9316	2828	0	272.43	357	3
23	10326	4440	0.01	238.27	208	3
24	10536	3378	0	382.37	250	1
25	9784	9784	-	TL	1	0
26	10822	4536	0	78.61	158	1
27	10634	2865	0	296.23	242	1
28	9450	9450	-	TL	1	0
29	10988	3964	0.73	TL	82536	11593
30	11840	4107	0	45.22	177	1
31	18142	18142	-	TL	1	0
32	19514	19514	-	TL	1	0
33	18008	18008	-	TL	4	0
34	19880	19880	-	TL	2	0
35	19196	19196	-	TL	1	0
36	21772	21772	-	TL	3	0
37	20010	20010	-	TL	2	0
38	20032	20032	-	TL	2	0
39	20136	20136	-	TL	480	0
40	20042	20042	-	TL	1	0

We observe that the straightforward implementation is able to solve all instances

with 15, 20, and 30 customers, but fails to find an optimal solution or prove its optimality for three of the instances with 60 customers. In fact, the first pricing problem cannot be solved within two hours for Instances 25 and 28. None of the instances with 120 customers can be solved within six hours.

Next, we solve Instances 1–30 using the enhanced implementation introduced in Section 5.2.6.2 with different combinations of control parameters for the solution of the pricing problem and with different branching schemes. More specifically, six configurations for solving the pricing problem were evaluated: $(5, 5, F)$, $(5, 10, F)$, $(10, 10, F)$, $(15, 10, F)$, $(10, 10, T)$ and $(15, 10, T)$, where the first parameter specifies the number of labels kept in the truncated-search (α), the second parameter specifies the number of negative reduced cost columns required before terminating the solution of the pricing problem early (β), and the third parameter specifies whether or not the state-space augmenting algorithm is terminated after a single label setting iteration (T or F). Furthermore, all branching schemes were evaluated: CB , MF , MFC , ED , and EDC . Thus, in total, the 30 instances were solved 30 times.

We found that 19 of the instances can be solved optimally without branching by each of the tested configurations. The solution to the master problem at the root node is always integer in 18 cases. In the other case (for Instance 21), some settings required solving the restricted master problem at the end of the column generation process at the root node as an integer program in order to produce an integer solution with cost equal to the optimal objective value of the master problem. Furthermore, we found that the branching schemes ED and EDC performed poorly compared to the other branching schemes. Therefore, in Table 5.4, we present the results for six configurations and three branching schemes for 12 instances. We report the solution time (in seconds), the number of pricing iterations, and the number of nodes evaluated.

To analyze the results and to choose a default branching scheme and default control parameters for the solution of the pricing problem, we will focus on the solution times. First, we observe that both the MF and the MFC branching schemes outperform the CB branching scheme. Second, when comparing the different pricing

Table 5.4: Results with *CB*, *MF*, and *MFC* arc selection rules

(5, 5, F)			(5, 10, F)			(10, 10, F)			(15, 10, F)			(10, 10, T)			(15, 10, T)			
Time	Iters.	Nodes	Time	Iters.	Nodes	Time	Iters.	Nodes	Time	Iters.	Nodes	Time	Iters.	Nodes	Time	Iters.	Nodes	
Select the arc for branching using the <i>CB</i> rule																		
6	1.06	193	47	0.75	84	21	0.96	100	25	0.99	106	29	1.16	164	33	1.21	198	45
7	1.52	217	20	1.48	150	16	1.67	139	19	2.27	169	23	2.85	217	23	3.04	216	23
9	0.67	143	25	0.52	103	22	0.53	76	17	0.57	88	19	0.73	125	25	0.88	138	25
11	0.87	119	11	0.68	73	7	0.82	82	13	1.17	93	15	0.85	85	7	1.45	109	11
12	96.68	16833	4765	82.10	12567	3957	53.76	9881	3195	101.41	15435	4981	57.07	9877	2971	95.59	14331	4243
17	0.64	73	3	0.59	60	7	0.76	77	7	0.59	44	3	0.52	45	3	0.57	67	3
19	5.76	523	73	14.14	1464	293	8.58	586	105	7.47	399	79	8.51	557	89	11.78	712	111
23	1.06	72	1	2.11	54	1	2.60	58	1	3.98	73	3	5.25	95	2	2.10	53	1
25	883.02	12085	899	158.57	1810	171	852.88	8379	962	832.75	7832	1027	389.69	3544	333	857.31	7307	699
28	43.52	572	31	241.30	2022	251	30.89	161	5	89.42	441	35	194.78	1119	113	106.05	656	61
29	1,714.91	42775	8557	1,713.56	39998	9471	2,407.46	43979	11197	1,461.10	20813	5207	2,776.68	51304	10841	1,570.03	25831	5477
Select the arc for branching using the <i>MF</i> rule																		
6	1.08	193	47	0.82	84	21	0.98	100	25	0.98	106	29	1.10	164	33	1.13	198	45
7	2.87	391	57	1.91	222	25	2.23	203	27	4.16	360	81	2.94	222	29	3.62	251	29
9	0.52	125	21	0.50	98	21	0.63	98	23	0.68	102	25	0.72	134	23	0.85	143	27
11	1.13	151	25	0.94	98	17	0.85	82	13	1.14	93	15	1.22	122	19	1.34	109	11
12	14.61	2174	579	9.25	1277	325	6.44	762	221	12.00	1389	401	17.59	2261	607	14.06	1889	489
17	0.73	89	7	0.57	60	7	0.74	77	7	0.55	44	3	0.44	45	3	0.93	99	7
19	11.13	1216	201	16.19	1881	379	10.41	844	173	6.79	524	113	10.10	860	153	15.82	1180	209
23	1.10	72	1	2.13	54	1	2.61	58	1	4.12	73	3	5.35	95	2	2.13	53	1
25	643.79	8695	687	451.87	6236	761	1,394.33	15068	2063	2,277.97	22493	3319	1,282.84	15608	1642	2,340.25	21952	2516
28	43.90	572	31	116.01	1027	131	30.48	161	5	103.08	502	45	175.97	1030	101	103.48	619	52
29	38.25	789	155	330.35	6644	1689	367.58	6145	1635	174.20	2215	555	127.94	1825	371	115.17	1643	281
Select the arc for branching using the <i>MFC</i> rule																		
6	1.19	193	47	0.79	84	21	0.97	100	25	1.02	106	29	1.10	164	33	1.16	198	45
7	2.11	264	37	1.83	212	23	1.92	179	21	4.19	329	65	3.10	236	29	3.38	242	27
9	0.63	143	25	0.52	98	21	0.65	90	21	0.79	124	31	0.71	134	23	0.87	138	25
11	0.97	125	17	0.95	98	17	0.81	82	13	1.134	93	15	0.86	85	7	1.42	109	11
12	13.06	2022	523	9.21	1244	321	12.42	1479	419	11.089	1277	361	11.77	1644	461	14.11	1736	453
17	0.53	70	3	0.57	60	7	0.74	77	7	0.58	44	3	0.449	45	3	0.58	67	3
19	8.72	894	155	11.25	1172	229	11.53	900	177	8.44	641	147	9.15	817	145	14.96	1095	195
23	1.08	72	1	2.13	54	1	2.62	58	1	4.26	73	3	5.22	95	2	2.13	53	1
25	63.17	713	25	376.50	3901	471	119.80	830	66	503.35	4230	556	1,179.33	14473	1510	1,500.08	12833	1286
28	44.93	572	31	283.52	2221	289	31.46	161	5	88.14	441	35	172.70	1027	99	103.78	656	61
29	163.76	3744	818	145.53	2618	587	83.35	1127	261	189.05	2736	653	98.72	1644	318	174.85	2847	551

parameter configurations used in the experiments, we see that the pricing problem is solved to optimality more often when multiple label setting iterations are not allowed during truncated search. This is expected given the fact that the column generation procedure is initialized with $S = \emptyset$ at each branch-and-price tree node and no customers are added to S by truncated-search when it is terminated during or at the end of the first label setting iteration. Invoking the full-search provides a means to update S and to identify the negative reduced cost columns that would not otherwise be encountered during truncated-search. However, this usually increases the total number of column generation iterations as well as the number of nodes in the branch-and-price tree, and results in longer solution times. Hence, we conclude that the configurations (α, β, F) lead to better performance in general. Although the choice between the different configurations of control parameters for the solution of the pricing problem is not obvious, schemes $MF(5, 5, F)$, $MFC(5, 5, F)$, and $MFC(10, 10, F)$ appear to be most “robust”, in the sense that they lead to the minimum total solution time across the first set of instances (with up to 60 customers).

We also examined how often a pricing iteration completes after (1) exploring the column pool, (2) applying truncated-search, and (3) applying full-search. For all branching schemes and all parameter configurations, most pricing iterations complete after truncated-search, which implies that truncated-search returns at least one elementary negative reduced cost column. Although rare, pricing iterations sometimes complete after exploring the column pool, especially when many pricing iterations have to be performed, which is not surprising since the column pool fills up gradually and the more columns, the better the chance of having negative reduced cost columns in the column pool in subsequent iterations.

We use the three most promising settings, i.e., $MF(5, 5, F)$, $MFC(5, 5, F)$, and $MFC(10, 10, F)$ to solve the 120 customer instances. The results can be found in Table 5.5. For each instance, the first four columns correspond to the name of the instance, the cost of the initial solution, the cost of the solution obtained by solving the restricted master problem at the root node as an integer program, and the cost of the best solution found, respectively. The remaining columns provide the integrality gap, the solution time, the total number of pricing iterations, the average time per

pricing iteration, and finally, the number of nodes evaluated. The best solution for each instance is highlighted in bold.

We observe that the algorithm is able to find an optimal solution to three instances with the settings $MF(5, 5, F)$, $MFC(5, 5, F)$ and $MFC(10, 10, F)$. Considering the number of best solutions obtained with each of these settings, $MF(5, 5, F)$ is superior to the others as it produces the best solution for seven instances, whereas six best solutions are found by the other two schemes. Based on the solution times for the instances that are solved to optimality, $MFC(5, 5, F)$ results in the minimum total solution time. Hence, no setting clearly dominates the others.

We also observe that solving the restricted master problem at the root node as an integer program is quite effective. With the setting $MF(5, 5, F)$, the solution value improves, on average, by almost 5%, and for Instance 39 the improvement exceeds 10%. The integrality gap values reveal that high-quality solutions are produced; with the setting $MF(5, 5, F)$, the average integrality gap is only 1.31%. Finally, we observe that although the total number of pricing iterations significantly decreases for most instances with a larger value of β , the time per pricing iteration increases when the number of labels kept at a node during truncated-search is larger (compare the time per iterations for $MFC(5, 5, F)$ and $MFC(10, 10, F)$).

To evaluate the benefits of the various techniques introduced in the branch-and-price algorithm to improve its performance, we next compare the straightforward implementation with the enhanced implementation with the setting $MF(5, 5, F)$. The results can be found in Table 5.6. A dash in the column “Root IP solution” means that the master problem has an integer optimal solution.

We observe that the benefits of implementing the techniques described earlier are significant. Instances that can be solved to optimality are solved orders of magnitude faster, and for the instances that cannot be solved to optimality, solutions of much better quality are obtained. Specifically, for the instances that can be solved optimally by both approaches, the reduction in average solution time is 97%. Furthermore, when the straightforward implementation fails to find an optimal solution

Table 5.5: Results obtained with the selected settings on the large instances

Instance	Initial solution	Root IP solution	Best solution (5, 5, F)	Integrality gap (%) with MF branching rule	Solution time (s)	Iterations in total	Time per iteration (s)	Nodes
31	5186	4935	4935	0.01	1,629.82	1375	1.18	1
32	5685	5278	5278	2.82	TL	56702	0.38	5518
33	5156	5091	5091	3.06	TL	48765	0.44	3564
34	5486	5219	5218	0.19	8,547.16	5747	1.48	305
35	5685	5536	5530	1.70	TL	39837	0.54	3708
36	7088	6498	6498	0	168.13	599	0.28	1
37	4967	4845	4845	0.55	TL	48336	0.44	3305
38	5745	5610	5608	0.99	TL	112974	0.19	12000
39	6552	5878	5878	1.37	TL	114894	0.19	12733
40	5265	5056	5048	2.43	TL	57739	0.37	3677
Average	5681.5	5394.6	5392.9	1.31	3448.37	48696.8	0.55	4481.2
(5, 5, F) with MFC branching rule								
31	5186	4935	4935	0.01	1,645.31	1375	1.19	1
32	5685	5278	5278	2.82	TL	20252	1.06	1561
33	5156	5091	5083	2.90	TL	39177	0.55	2154
34	5486	5219	5218	0.19	4,618.77	3114	1.48	125
35	5685	5536	5528	1.67	TL	41013	0.52	2749
36	7088	6498	6498	0	183.85	599	0.31	1
37	4967	4845	4845	0.55	TL	51382	0.42	3966
38	5745	5610	5609	1.01	TL	122279	0.17	9513
39	6552	5878	5878	1.37	TL	88890	0.24	10250
40	5265	5056	5056	2.59	TL	52736	0.41	3217
Average	5681.5	5394.6	5392.8	1.31	2149.31	42081.7	0.64	3353.7
(10, 10, F) with MFC branching rule								
31	5186	4935	4935	0.01	1,513.15	622	2.43	1
32	5685	5278	5278	2.82	TL	23568	0.91	2479
33	5156	5093	5085	2.94	TL	40682	0.53	3111
34	5486	5221	5218	0.19	12,159.45	5605	2.16	501
35	5685	5548	5519	1.50	TL	21859	0.98	2234
36	7088	-	6498	0	213.30	364	0.58	1
37	4967	4854	4854	0.74	TL	34600	0.62	3641
38	5745	5616	5610	1.02	TL	66327	0.32	7344
39	6552	5882	5849	0.87	TL	51237	0.42	7256
40	5265	5050	5050	2.47	TL	28476	0.75	3448
Average	5681.5	5275.22	5389.6	1.26	4628.63	27334	0.97	3001.6

Table 5.6: Straightforward branch-and-price vs. the default branch-and-price with parameter configuration $(5, 5, F)$ and the MF branching rule

Instance	Straightforward		$MF(5, 5, F)$				
	Best solution	Solution time (s)	Initial solution	Root IP solution	Best solution	Integrality gap (%)	Solution time (s)
1	901	0.58	957	-	901	0	0.26
2	1286	0.27	1292	-	1286	0	0.04
3	991	0.43	1004	-	991	0	0.07
4	1062	0.27	1069	-	1062	0	0.04
5	1832	0.04	1832	-	1832	0	0.02
6	1294	2.30	1300	1300	1294	1.73	1.08
7	1155	30.47	1158	1158	1155	1.29	2.87
8	1455	0.18	1555	-	1455	0	0.07
9	1260	3.32	1272	1268	1260	1.86	0.52
10	1684	0.55	1733	-	1684	0	0.03
11	1922	14.28	1931	1922	1922	0.31	1.13
12	2324	120.38	2325	2325	2324	2.52	14.61
13	1747	23.33	1758	-	1747	0	0.68
14	1273	30.83	1281	-	1273	0	0.64
15	1694	22.78	1696	-	1694	0	0.50
16	1938	57.13	1941	-	1938	0	0.75
17	1965	5.75	1966	1965	1965	0.10	0.73
18	1827	2.10	1831	-	1827	0	0.23
19	2083	93.55	2121	2110	2083	2.46	11.13
20	1822	78.70	1889	-	1822	0	1.53
21	3761	606.60	3775	-	3761	0	4.13
22	2828	272.43	2877	-	2828	0	10.74
23	4440	238.27	4447	4440	4440	0.01	1.10
24	3378	382.37	3477	-	3378	0	11.62
25	9784	TL	3375	3169	3161	0.84	643.79
26	4536	78.61	4586	-	4536	0	1.87
27	2865	296.23	2877	-	2865	0	7.08
28	9450	TL	4220	4176	4173	0.08	43.90
29	3964	TL	4017	3979	3964	0.73	38.25
30	4107	45.22	4122	-	4107	0	1.80
31	18142	TL	5186	4935	4935	0.01	1629.82
32	19514	TL	5685	5278	5278	2.82	TL
33	18008	TL	5156	5091	5091	3.06	TL
34	19880	TL	5486	5219	5218	0.19	8547.16
35	19196	TL	5685	5536	5530	1.70	TL
36	21772	TL	7088	6498	6498	0.00	168.13
37	20010	TL	4967	4845	4845	0.55	TL
38	20032	TL	5745	5610	5608	0.99	TL
39	20136	TL	6552	5878	5878	1.37	TL
40	20042	TL	5265	5056	5048	2.43	TL

within the time limit, it is also unable to solve the root LP; this never happens with the enhanced implementation. Finally, we note that the average integrality gap, when using the enhanced implementation, is 1.85%, which demonstrates that high-quality solutions can be obtained for VRPRDL instances with up to 120 customers.

5.4.3 Impact of distance of roaming delivery locations to the depot on algorithm performance

In the previous experiments, we focused on identifying an effective parameter configuration for solving the pricing problems and an effective branching scheme. Next, we assess the performance of the branch-and-price algorithm (with configuration $MF(5, 5, F)$) on the second set of instances in order to evaluate whether the distance of the roaming delivery locations to the depot affects its performance. The results for both variations are reported in Table 5.7.

Table 5.7: Results for Instances 41–50 obtained by enhanced branch-and-price with $MF(5, 5, F)$

Instance	Initial solution	Root IP solution	Best solution	Integrality gap (%)	Solution time (s)	Iterations	Nodes
41_v1	3278	3208	3203	1.88	1249.35	33672	6147
41_v2	2147	2139	2133	2.33	854.47	7528	805
42_v1	2842	-	2799	0	3.00	58	1
42_v2	2032	2012	1946	4.12	1005.36	6472	743
43_v1	2614	2607	2607	3.68	TL	111845	14854
43_v2	1995	1966	1966	1.09	270.72	1644	135
44_v1	2344	2273	2261	2.03	98.52	2342	251
44_v2	1749	-	1610	0	41.59	222	1
45_v1	3220	-	3217	0	1.63	34	1
45_v2	2479	-	2478	0	9.76	80	1
46_v1	2806	2805	2805	0.91	3.81	126	5
46_v2	2504	2469	2469	1.11	27.37	302	39
47_v1	3463	3385	3339	3.38	3710.35	89246	21169
47_v2	1992	1947	1946	1.22	68.96	556	37
48_v1	3331	-	3325	0	1.15	48	1
48_v2	2418	2386	2380	1.65	477.83	8384	1493
49_v1	3598	3538	3534	1.82	104.26	3084	595
49_v2	2605	2507	2492	0.55	13.62	207	9
50_v1	2779	-	2752	0	8.74	114	1
50_v2	2551	2449	2443	0.38	164.371	1038	119

We observe that for each instance the cost of the solution to the second variation is smaller than the cost of the solution to the first variation, which is to be expected as it should be possible to better exploit the roaming delivery locations when they are closer to the depot. Furthermore, we see that all second variation instances are solved optimally, with a maximum solution time of 1005.36s, whereas instance 42_v1 cannot be solved within the time limit of two hours. Moreover, the total time spent by the algorithm in solving all second variation instances is considerably less than the total time required to solve all first variation instances. However, the algorithm does not perform uniformly better on second variation instances. The fact that there are likely fewer feasible solutions in the first variation instances may explain why more first variation instances can be solved at the root node of the search tree. That there are likely more feasible solutions in the second variation instances is reflected by the fact that the problem graph after preprocessing for the second variation is denser for all instances.

5.4.4 Assessing the benefits of employing trunk delivery services

To be able to comprehensively assess the benefits of trunk delivery services, we need to be able to solve instances of the VRPHRDL as well. Therefore, we start by investigating how well our branch-and-price algorithm performs on VRPHRDL instances. We experimented with the three settings that proved most effective for the VRPRDL instances and found that although no setting clearly dominated, most best-known solutions are found with the setting $MFC(10, 10, F)$. Therefore, we report the results obtained with this setting in Table 5.8, where, for completeness sake, we include also the value of the best solution found by any of the three settings (highlighting in bold the values of the solutions that have a smaller cost than the value of the best solution obtained with the setting $MFC(10, 10, F)$).

For the VRPHRDL instances with up to 60 customers, 24 of the 30 instances can be solved to optimality within two hours and optimal solutions to most of these

Table 5.8: Results for the VRPHRDL instances with parameter configuration $(10, 10, F)$ and the *MFC* branching rule

Instance	Initial solution	Root IP solution	Best solution	Best known solution	Integrality gap (%)	Solution time (s)	Iterations	Nodes
1	777	-	773	773	0	0.62	23	1
2	1111	-	1065	1065	0	0.08	9	1
3	991	-	988	988	0	0.11	15	1
4	916	-	914	914	0	0.17	16	1
5	1710	-	1710	1710	0	0.04	10	1
6	1099	1099	1099	1099	2.04	2.84	147	23
7	1020	1010	996	996	0.67	11.02	290	36
8	1457	-	1346	1346	0	0.33	24	1
9	998	-	997	997	0	0.56	22	1
10	1167	-	1166	1166	0	0.18	26	1
11	1607	-	1587	1587	0	8.54	80	1
12	1834	-	1808	1808	0	4.70	53	1
13	1563	-	1563	1563	0	3.38	38	1
14	1058	-	1058	1058	0	3.26	45	1
15	1363	1355	1347	1347	2.06	155.93	1232	175
16	1574	1564	1517	1517	2.09	TL	139823	30731
17	1446	1445	1445	1445	0	2.14	58	1
18	1679	1627	1627	1627	0.95	26.67	271	35
19	1468	-	1461	1461	0	1.59	38	1
20	1730	-	1715	1715	0	2.09	49	1
21	2860	-	2580	2580	0	396.47	495	1
22	2213	2213	2213	2213	4.98	TL	7654	591
23	3393	3373	3363	3363	0.18	194.98	372	23
24	2587	2574	2574	2569	4.70	TL	8400	596
25	2429	2414	2400	2400	5.38	TL	8898	622
26	2881	2847	2846	2845	0.80	TL	19931	2414
27	2521	-	2518	2518	0	33.85	114	1
28	2830	2758	2758	2758	0.04	3,392.94	578	5
29	2917	2913	2913	2892	6.93	TL	22862	3460
30	2711	-	2691	2691	0	41.77	106	1
31	3991	3984	3984	3984	-	TL	1876	0
32	4012	3958	3958	3958	-	TL	1011	0
33	3828	3645	3645	3630	-	TL	2155	0
34	4065	3958	3958	3891	-	TL	1598	0
35	3290	3255	3255	3255	-	TL	1905	0
36	4607	4533	4533	4525	-	TL	1102	0
37	3585	3395	3395	3395	-	TL	2063	0
38	4131	3980	3980	3976	-	TL	1343	0
39	4686	4316	4316	4316	-	TL	1519	0
40	3980	3680	3680	3680	-	TL	1876	0

instances are found at the root node. Note that, as in previous experiments, the initial solutions are obtained by the heuristic of [57].

For the VRPHRDL instances with 120 customers, the algorithm does not finish solving the root node LP within six hours, demonstrating that VRPHRDL instances are harder to solve than VRPRDL instances. Even when the time limit is reached before solving the root node LP, a large number of columns has been generated, which enables us to find an improved solution by solving the restricted master problem as an integer program when the time limit is reached. We observe that for these difficult instances, the branch-and-price algorithm finds solutions that, on average, improve the initial solution by approximately 4%. As mentioned at the start of this section, the primary reason that VRPHRDL instances are more difficult than VRPRDL instances is that time windows at the home locations are much wider, which reduces the effectiveness of the preprocessing, and, as a consequence, leads to denser graphs and more arc variables.

Having alternative locations to deliver a customer order, e.g., to the trunk of the customer’s car when it is parked somewhere other than at home, provides additional flexibility to a delivery company. However, it is easy to construct examples where allowing deliveries only to the trunk of a customer’s car may not lead to cost savings, but may, in fact, lead to an increase in cost, compared to being able to only deliver at the customer’s home. This is the reason that companies are more likely to deploy a hybrid model in which deliveries can either be made to the customer’s home or to the customer’s car.

To assess the benefits of trunk delivery, we compute, for the same instance, a VRP solution, in which deliveries can only be made at the customer’s home location, a VRPRDL solution, in which deliveries can only be made to the trunk of the customer’s car, and a VRPHRDL solution, in which deliveries can be made either to the customer’s home or to the trunk of the customer’s car. The results can be found in Table 5.9, where the VRP solution is also computed with our branch-and-price algorithm. We report the cost, the fraction of deliveries made at a customer’s home, and the number of delivery routes. We use a star to indicate that the best solution

is not necessarily optimal, i.e., the algorithm reached the time limit.

We observe that having the flexibility to deliver either to a customer’s home or to the trunk of the customer’s car has huge advantages. The average percentage of cost-savings is over 18% (with a minimum cost savings of 4.93% and a maximum cost savings of 36.29%). This is, in a large part, because fewer delivery routes are required; on average 8.88 for the VRP solutions and on average 7.65 for the VRPHRDL solutions. Interestingly, the average number of delivery routes in the VRPRDL solutions is 11.10. Even though there are more delivery locations to choose from, the time during which deliveries can be made is less, because deliveries cannot be made while the car is driving. Also interesting to note is that the fraction of deliveries made at a customer’s home location in VRPHRDL solutions is quite high, on average 0.76.

The difference in VRP, VRPRDL, and VRPHRDL solutions can be seen quite well in Figure 5.1, where we show the optimal VRP, VRPRDL, and VRPHRDL solutions for Instance 28. The empty rectangles represent the home locations while the filled circles represent the other potential delivery locations. The optimal VRP solution has 10 delivery routes and a cost of 3423, whereas the number of delivery routes in the optimal VRPRDL solution is 14 with cost 4173 (55% of the deliveries are made at home locations), and the number of delivery routes in the optimal VRPHRDL solutions is only 8 with cost 2758 (83% of the deliveries are made at home locations).

We repeated the last experiment for the instances of the second set and found similar results as shown in Table 5.10.

5.5 Concluding remarks

Trunk delivery is one of the innovative ideas being explored to reduce delivery cost in the business-to-consumer retail market sector. Our computational study shows that a hybrid model in which a delivery can be made either at a customer’s home or

Table 5.9: Comparison of the VRP, the VRPRDL, and the VRPHRDL solutions

Instance	VRP			VRPRDL			VRPHRDL			Savings wrt best VRP solution (%)
	Routes	Best solution	Home/total	Routes	Best solution	Home/total	Routes	Best solution		
1	3	864	0.47	4	901	0.60	3	773	10.53	
2	4	1187	0.47	5	1286	0.73	4	1065	10.28	
3	5	1305	0.60	4	991	0.67	3	988	24.29	
4	3	974	0.53	5	1062	0.87	3	914	6.16	
5	7	2171	0.53	6	1832	0.67	6	1710	21.23	
6	4	1246	0.65	5	1294	0.75	4	1099	11.80	
7	4	1156	0.55	4	1155	0.85	3	996	13.84	
8	5	1636	0.60	6	1455	0.70	5	1346	17.73	
9	4	1215	0.35	5	1260	0.60	4	997	17.94	
10	5	1444	0.60	8	1684	0.85	4	1166	19.25	
11	8	2491	0.50	7	1922	0.70	5	1587	36.29	
12	6	2001	0.50	8	2324	0.67	6	1808	9.65	
13	6	1774	0.60	6	1747	0.73	6	1563	11.89	
14	5	1648	0.50	6	1273	0.80	4	1058	35.80	
15	6	1935	0.47	6	1694	0.57	5	1347	30.39	
16	6	1890	0.47	7	1938	0.80	5	1517*	19.74	
17	7	2199	0.33	8	1965	0.57	5	1445	34.29	
18	6	1836	0.43	7	1827	0.73	5	1627	11.38	
19	6	1889	0.70	7	2083	0.83	5	1461	22.66	
20	6	1823	0.83	6	1822	0.83	6	1715	5.92	
21	9	3046	0.47	13	3761	0.80	8	2580	15.30	
22	7	2452*	0.57	10	2828	0.78	7	2213*	9.75	
23	12	3949	0.48	16	4440	0.87	10	3363	14.84	
24	9	2924	0.43	11	3378	0.80	8	2569*	12.14	
25	8	2648*	0.47	11	3161	0.78	8	2400*	9.37	
26	11	3663	0.47	16	4536	0.73	9	2845*	22.33	
27	11	3438	0.60	10	2865	0.72	8	2518	26.76	
28	10	3423	0.55	14	4173	0.83	8	2758	19.43	
29	12	4010*	0.48	14	3964	0.75	9	2892*	27.88	
30	12	3924	0.38	14	4107	0.77	8	2691	31.42	
31	16	4919	0.53	18	4935	0.82	14	3984*	19.01	
32	15	4593*	0.46	19	5278*	0.80	13	3958*	13.83	
33	14	4296*	0.53	18	5083*	0.73	13	3630*	15.50	
34	14	4613*	0.52	17	5218	0.83	13	3891*	15.65	
35	11	3725*	0.48	20	5519*	0.86	11	3255*	12.62	
36	18	5745*	0.51	22	6498	0.83	15	4525*	21.24	
37	14	4804*	0.48	17	4845*	0.74	11	3395*	29.33	
38	14	4395*	0.55	21	5608*	0.79	14	3976*	9.53	
39	19	6028*	0.45	24	5849*	0.84	15	4316*	28.40	
40	13	3871*	0.46	19	5048*	0.85	13	3680*	4.93	
Average	8.88	2828.75	0.51	11.10	3065.23	0.76	7.65	2290.53	18.26	

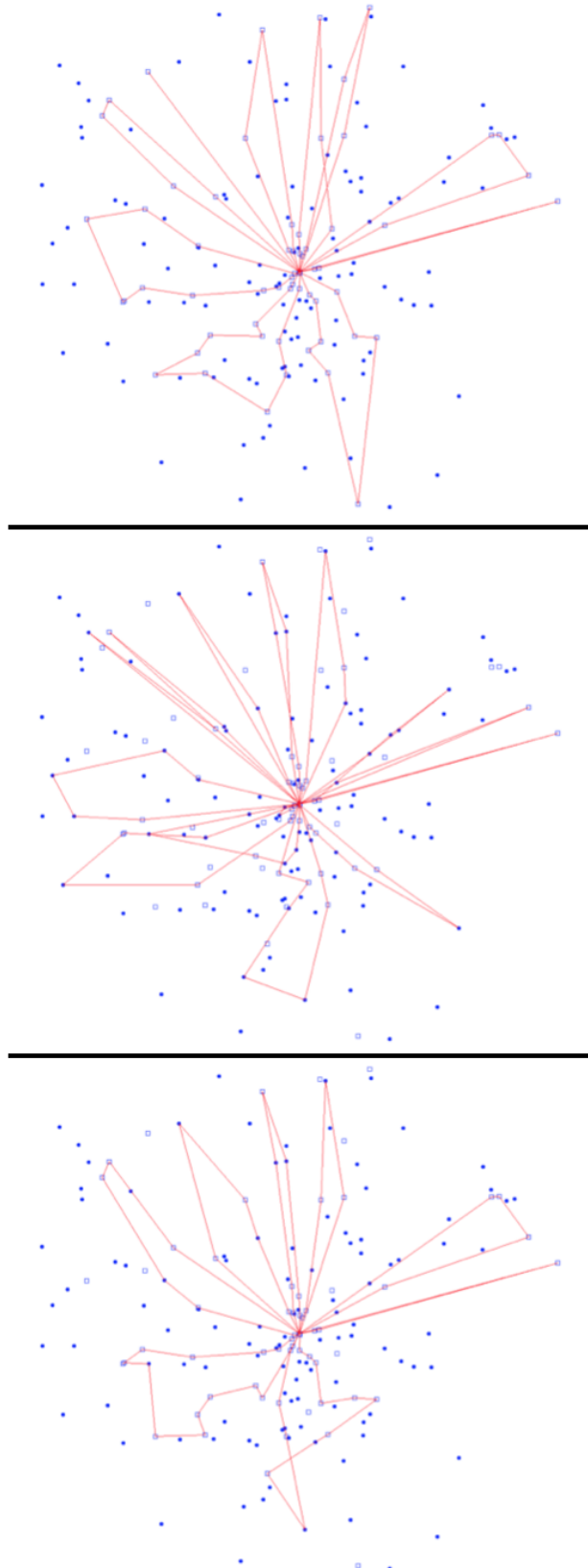


Figure 5.1: The optimal VRP, VRPRDL and VRPHRDL solutions for Instance 28
from top to bottom, respectively

Table 5.10: Comparison of the VRP, the VRPRDL, and the VRPHRDL solutions for instances in the second set.

Instance	VRP		VRPRDL			VRPHRDL			Savings wrt best VRP solution (%)
	Routes	Best solution	Home/total	Routes	Best solution	Home/total	Routes	Best solution	
41.v1	8	2745	0.55	10	3203	0.80	8	2662	3.02
41.v2	8	2738	0.53	7	2133	0.65	6	1998	27.03
42.v1	9	2805	0.58	9	2799	0.80	8	2610	6.95
42.v2	9	2806	0.58	7	1946	0.58	6	1946*	30.65
43.v1	8	2536*	0.45	8	2607*	0.70	7	2260	10.88
43.v2	8	2536*	0.43	8	1966	0.50	6	1830	27.84
44.v1	7	2318	0.65	7	2261	0.75	7	2147	7.38
44.v2	7	2315	0.60	6	1610	0.70	5	1478	36.16
45.v1	10	3191	0.80	10	3217	0.90	10	3172	0.60
45.v2	10	3192	0.70	8	2478	0.73	8	2466	22.74
46.v1	8	2718	0.53	9	2805	0.75	8	2616	3.75
46.v2	8	2717	0.50	8	2469	0.58	8	2388	12.11
47.v1	9	3069	0.53	10	3339	0.70	9	3011*	1.89
47.v2	9	3060	0.40	7	1946	0.50	6	1848	39.61
48.v1	10	3358	0.50	10	3325	0.65	10	3278	2.38
48.v2	10	3359	0.40	8	2380	0.60	7	2264	32.60
49.v1	11	3615	0.55	11	3534	0.65	11	3514*	2.79
49.v2	11	3615	0.45	8	2492	0.55	8	2457	32.03
50.v1	9	2928	0.50	10	2752	0.55	10	2727	6.86
50.v2	9	2930	0.55	8	2443	0.73	7	2302	21.43
Average	8.90	2927.55	0.54	8.45	2585.25	0.67	7.75	2448.70	16.44

to the trunk of the customer’s car has huge potential. On the instances in our test suite, the cost savings were in the order of 20%.

We have assumed deterministic travel times and complete knowledge of the itinerary of a customer (more specifically the itinerary of the customer’s car). These are strong assumptions, but they are, in our opinion, not completely unreasonable. There are well-funded and highly-regarded start-up companies, e.g., Roadie (www.roadie.com), with a business proposition that is entirely based on the assumption that in the future, there will be reliable and predictable information on the travel patterns of people (based on 24-hour monitoring of the location of their smart phone and predictive analytics). Starting to explore new business models assuming this information is available is important, and we are among the first to do so.

However, we recognize that an important next step is to investigate operational strategies and related optimization models that can dynamically handle deviations from the planned customer itineraries. Therefore, in the next chapter, we focus on a dynamic variant of the VRPRDL and propose an iterative re-optimization approach to solve it.

Chapter 6

An Iterative Re-optimization Framework for the Dynamic Vehicle Routing Problem with Roaming Delivery Locations

In the previous chapter, we assumed that the itinerary of each customer is static and known with certainty when determining the delivery routes. However, in reality, a customer's itinerary can easily change during the day, and in some cases, the planned delivery routes may no longer be able to accommodate that customer. For example, suppose that a customer was planning to go home right after a meeting scheduled for 4pm, which got cancelled at the last minute, and he left work earlier than expected. If the order of this customer were going to be delivered at his work location some time during the meeting, then the planned delivery schedule will become infeasible. It may, if not in all cases, be still possible to find a set of feasible delivery routes by allowing the vehicles to serve customers that are not originally included in their

routes and by allowing the deliveries of customers to be made at locations that are different from the previously determined ones. Note that we assume that there is a single type of product demanded by every customer to ensure that a vehicle can deliver to customers that were previously assigned to other vehicles' routes as long as it has sufficient amount of products. Although this is a strong assumption, dedicating a set of vehicles to serve the customers demanding the same type of product is not completely unrealistic, because such a strategy enables utilizing the vehicles interchangeably when a disruption, as in the example above, occurs during the planning horizon. In the presence of multiple product types, the problem is applicable in a setting where vehicles collect the items that customers wish to return instead of making deliveries.

In this chapter, we introduce a dynamic variant of the vehicle routing problem with roaming delivery locations (D-VRPRDL), where there may be deviations from the original customer itineraries which can render the planned delivery schedule infeasible or suboptimal. To the best of our knowledge, ours is the first study focusing on a dynamic variant of the generalized vehicle routing problem and considering deviations in time windows as the source of dynamism.

The D-VRPRDL can be classified as a *dynamic and deterministic* vehicle routing problem. It is dynamic since part of the input (in our case, customer itineraries) changes over the planning horizon, and deterministic since we do not have any stochastic information about the input that will be revealed in the future. We propose an iterative framework using the branch-and-price algorithm developed for the VRPRDL to re-optimize the vehicle routes every time the itinerary of a customer is updated. The existing solution methods on dynamic and deterministic vehicle routing problems are mostly based on periodic re-optimization either by dividing the planning horizon into fixed decision epochs (also known as time slices) or as soon as a certain number of changes occurs in the input data. Our solution approach falls into the latter category.

Re-optimizing the previously planned vehicle routes requires solving a static problem with the currently available data. In order to put the updated vehicle routes

into effect as soon as possible, it is critical to solve each static problem quickly. To this end, most approaches in the literature rely on heuristics when solving static problems. However, by making effective use of the information collected during the solution of the previous static problems, it may be possible to obtain optimal or near-optimal solutions in a reasonably short amount of time using an exact approach as well. With this motivation, our purpose in this study is to investigate whether we can generate columns efficiently in a dynamic setting, in particular, whether the columns generated during the previous executions of the branch-and-price algorithm can be utilized towards a more efficient solution process for the re-optimization problems compared to solving them from scratch.

To the best of our knowledge, the only study employing a column generation based approach for solving a dynamic vehicle routing problem is due to [81]. The problem and the methodology presented in [81] are different from ours in the following respects:

- They focus on dynamically arriving pick up orders, while in our case, the source of dynamism is due to deviations in time windows,
- Although every pick up order has a hard time window, the authors do not allow rejection of service (i.e., each customer has to be served within the time window specified by the customer). Therefore, they assume that all orders arrive early enough to guarantee that they can always be picked up by dispatching a new vehicle from the depot. In the D-VRPRDL, we allow rejection of service since we do not make any assumptions about the latest time at which an itinerary update can be revealed.
- The D-VRPRDL is a dynamic variant of a generalized VRPTW. Hence, the problem solved at each decision point in our iterative re-optimization approach is a more general compared to the one in their case, which is a VRPTW.
- Their solution approach is based on column generation, but they generate new columns heuristically using the existing columns, and then solve a restricted

set partitioning model –involving the columns generated up to that time– by calling the MIP solver of CPLEX at each decision point.

- We re-optimize the previously planned vehicle routes after every update in the input data whereas they use fixed time slices.

The rest of this chapter is organized as follows. In Section 6.1, we formally define the D-VRPRDL and state our assumptions. As mentioned earlier, the iterative re-optimization scheme developed for the D-VRPRDL solves a series of static problems during the planning horizon in order to dynamically handle deviations from the original customer itineraries. Section 6.1 also gives a set covering based formulation for each static problem and the associated pricing problems. In Section 6.2, we present the details of our solution approach. Section 6.3 explains the scheme we use for generating itinerary updates and reports the results of a preliminary computational study conducted to test the efficiency of the proposed solution approach and to explore the value of information collected during the previous executions of the branch-and-price algorithm when solving a re-optimization problem. Finally, we provide some concluding remarks and future research directions in Section 6.4.

6.1 Problem definition and formulations

In this section, we provide a formal description of the D-VRPRDL in which for each customer, an itinerary, specifying when and where the customer’s order can be delivered, is available at the planning stage, but may change during the operation. As a consequence, the delivery routes should be revised based on the updated customer itineraries. For the sake of consistency, we use the notation defined in the previous chapter.

The goal is to find a set of delivery routes visiting each customer at one of the locations in the customer’s itinerary, during the time that the customer is at that location, and such that the demand delivered on a route is no more than Q , the

duration of a route does not exceed T , and the total transportation cost is minimized. As deviations from the original customer itineraries can render the planned delivery schedule infeasible or suboptimal, the vehicle routes are re-optimized after each update in an iterative fashion.

A feasible set of delivery routes shows the planned delivery location for every customer, i.e., the location where the customer's order will be delivered. Furthermore, it provides information on the earliest time that a vehicle can arrive at each customer location after serving the previous customers assigned to its route (assuming that it was dispatched from the depot at time 0) as well as the latest time that the vehicle should depart from each of these locations to serve the subsequent customers in its route within their respective time windows and return to the depot by time T . For planned delivery location i , we use ea_i and ld_i to represent the earliest arrival and the latest departure times of the assigned vehicle to and from i , respectively. We say that a given route is feasible, if for every node i in the route, the node's time window $[e_i, l_i]$ coincides with the delivery vehicle's time window $[ea_i, ld_i]$, that is, if $e_i \leq ld_i$ and $l_i \geq ea_i$.

For simplicity, we assume that itinerary updates are revealed one at a time, i.e., the delivery planner does not receive information about the itinerary change of two or more customers simultaneously. For a customer $c \in C$, we restrict ourselves to updates caused by early departure from or late arrival to a location $i \in N_c$. Since the travel times are static and deterministic in our problem setting, the resulting deviation is absorbed by the previous or subsequent locations in the customer's itinerary depending on the type of update. We also assume that every customer will visit all locations in his original itinerary. Therefore, if a customer leaves a location earlier/later than expected, time windows associated with some locations in his itinerary will get narrower while some will get wider. Note that when absorbing the deviation caused by an update, we do not allow any time window to shrink more than 75% of its original length in our update generation scheme which will be described later.

Another assumption we make is that the delivery vehicles can be diverted from

their planned routes to serve customers that are not originally assigned to them although we do not allow en route diversions. Suppose that an itinerary update is revealed at time t . A vehicle that is on its way to a customer location i at time t can be re-routed only after arriving at i , whereas if the vehicle is waiting at this location at time t and $e_i > t$, it can be re-routed immediately. Redirecting a vehicle to serve a customer while it was in transit to another customer may provide additional flexibility and yield considerable savings compared to adopting a “no en route diversion” strategy. Nonetheless, it requires responding to changes in the input data almost instantly in order to use the available diversion opportunities effectively, while it may be possible to identify solutions with better overall quality by investing more time in re-optimization instead, as pointed out in [99]. Hence, en route diversion strategy has received limited attention in the literature (see, for example, [135–140]).

Depending on the time of an update as well as the type and the amount of deviation incurred, it may/may not be possible to deliver the remaining orders –without violating the time window restrictions– using the vehicles that are assigned to the previously planned routes. Therefore, we assume that a sufficient number of vehicles is available at the depot and additional vehicles can be dispatched if needed. Clearly, even deploying more vehicles does not guarantee the feasibility of the re-optimization problem especially when an update is revealed towards the end of the planning horizon. Hence, rejection of service is allowed in our problem setting.

To solve the D-VRPRDL, we develop an iterative re-optimization scheme that solves a series of static problems using the branch-and-price algorithm described in the previous chapter. The first problem we solve is simply the VRPRDL defined over the entire planning horizon $[0, T]$ and the original customer itineraries. All subsequent problems can be considered as a restricted version of the VRPRDL since they involve a set of additional constraints specifying the origin for each vehicle dispatched from the depot before t , which is the time of the latest update. In the following subsection, we describe the static problem that needs to be solved at decision point t , denoted by SP hereafter, and provide a set covering based formulation.

6.1.1 Static problem formulation

Let O_t be the set of origin nodes for the static problem defined over the horizon $[t, T]$ with $t > 0$. This set contains the depot node, and one node for each vehicle assigned to a previously planned delivery route, indicating the location where the vehicle will be available for re-routing. In particular, if a vehicle is in transit at time t , then the corresponding origin node is the vehicle's current destination; otherwise, it is simply the current location of the vehicle. Every origin node $o \in O_t$ is associated with a time st_o and a quantity qr_o specifying the earliest time at which the vehicle located at origin o can be re-routed and remaining capacity of this vehicle, respectively. We take $st_o = t$ for the vehicles that are at the depot or at a customer location. For a vehicle that is en route, we take st_o to be the time that the vehicle will reach its destination. We compute qr_o considering the deliveries made up to time t . Hence, we assume that $qr_o = Q$. We use $C_t \subseteq C$ to represent the set of customers that require a delivery during $[t, T]$ ($C_0 = C$), and $N_c^t \subseteq N_c$ is the set of locations in the itinerary of customer $c \in C_t$ for which the time windows did not close earlier than t ($N_c^0 = N_c$). We define $N^t = \bigcup_{c \in C_t} N_c^t$.

We denote by R_o^t the set of all feasible delivery routes (i.e., respecting capacity and time window constraints) originating from $o \in O_t$ with an earliest start time of st_o and a delivery capacity of qr_o units. We define $R^t = \bigcup_{o \in O_t} R_o^t$. Let w_r be the cost of route $r \in R^t$, and let a_{ir} for every $i \in N^t$ and $r \in R^t$ indicate whether location i is visited on route r ($a_{ir} = 1$) or not ($a_{ir} = 0$). Then, the static problem SP can be formulated as follows:

$$\min \sum_{r \in R^t} w_r z_r \tag{6.1}$$

$$\sum_{r \in R^t} \sum_{i \in N_c^t} a_{ir} z_r \geq 1 \quad c \in C_t, \tag{6.2}$$

$$\sum_{r \in R_o^t} z_r = 1 \quad o \in O_t \setminus \{0\}, \tag{6.3}$$

$$z_r \in \mathbb{Z}_+ \quad r \in R^t, \tag{6.4}$$

where z_r is the number of times route r is used. This formulation is quite similar

to that of the static VRPRDL given in (5.9)–(5.12). Here, we have an additional set of constraints (6.3) specifying the location where each vehicle dispatched before t should originate from in the revised routing plan. Note that we do not restrict the number of vehicles originating from node 0, since we assume that a sufficiently large number of vehicles is available at the depot and can be dispatched whenever necessary. It is also worth mentioning that we do not penalize the use of additional vehicles, since the number of vehicle routes in the re-optimized delivery schedule will most likely be kept at minimum due to the fact that the arc costs satisfy the triangle inequality.

The formulation above has exponentially many variables as the number of routes is exponential in the size of N^t . Therefore, we use the branch-and-price algorithm described in the previous chapter to solve it. However, that algorithm is originally developed for the VRPRDL in which constraints (6.3) are not present. Each of these constraints gives rise to a separate pricing problem for the vehicle with the corresponding origin, whereas the pricing subroutine of the algorithm is configured to solve a single pricing problem to generate columns. Instead of modifying the algorithm, we make a manipulation on the problem graph which enables us to solve the pricing problems as a single problem, which will be discussed later in Section 6.2.1. Next, we derive the pricing problems, one for every $o \in O_t$, associated with SP .

6.1.2 Pricing problems

Consider the LP relaxation of (6.1)–(6.4), which will be referred to as the master problem from now on. Let $\bar{R}^t \subset R^t$ be such that there exists a feasible solution to the master problem when $z_r = 0$ for all $r \in R^t \setminus \bar{R}^t$. A formulation involving only routes in \bar{R}^t is called a restricted master problem (RMP). The dual of the RMP is:

$$\max \sum_{c \in C_t} \lambda_c + \sum_{o \in O_t \setminus \{0\}} \mu_o \quad (6.5)$$

$$\sum_{c \in C_t} \sum_{i \in N_c^t} a_{ir} \lambda_c + \mu_o \leq w_r \quad o \in O_t, r \in \bar{R}_o^t, \quad (6.6)$$

$$\lambda_c \geq 0 \quad c \in C_t, \quad (6.7)$$

$$\mu_o \in \mathbb{R} \quad o \in O_t \setminus \{0\}, \quad (6.8)$$

where $\lambda = (\lambda_1, \dots, \lambda_{|C_t|})$ and $\mu = (\mu_1, \dots, \mu_{|O_t|-1})$ are dual variable vectors associated with the constraints (6.2) and (6.3), and $\mu_0 = 0$. Denote the optimal solution to this dual problem by (λ^*, μ^*) . In the pricing problem for each origin, the goal is to identify a column with negative reduced cost with respect to the original master problem. In other words, for $o \in O_t$, we aim to find a column for which the associated dual constraint is violated. Such a column is a route r for which the following condition is satisfied:

$$w_r - \mu_o^* - \sum_{c \in C_k} \sum_{i \in N_c^k} a_{ir} \lambda_c^* < 0.$$

Note that since $w_r = \sum_{(i,j) \in r} w_{ij}$ and $\sum_{c \in C_t} \sum_{i \in N_c^t} a_{ir} \lambda_c^* = \sum_{i \in N^t} a_{ir} \lambda_{c(i)}^*$, we can rewrite the condition as:

$$\sum_{(i,j) \in r} w_{ij} - \mu_o^* - \sum_{i \in N^t} a_{ir} \lambda_{c(i)}^* < 0$$

for every $o \in O_t$. Thus, the pricing problem for a particular origin o is an elementary shortest path problem with time window and capacity constraints (ESPPTWCC), where the cost of arc (i, j) is set to

$$\bar{w}_{ij} = \begin{cases} w_{ij} - \mu_o^*, & \text{if } i = o \\ w_{ij} - \lambda_{c(i)}^*, & \text{if } i \in N^t \end{cases}$$

for $j \in N^t \cup \{0\}$. In the ESPPTWCC, the goal is to find an elementary path of shortest length starting at origin node $o \in O_t$ and ending at the depot while respecting capacity and time window constraints. It is important to note here that the nodes in the set $O_t \setminus \{0\}$ are copies of the actual customer locations. More specifically, if a vehicle is at/en route to customer location i , then we add i' to O_t , where i' is a copy of node i . This is necessary to distinguish between the cases where the vehicle actually makes a delivery at location i and where the vehicle visits

location i but leaves without delivering the order of customer $c(i)$. We model these situations by adding arcs from node i' to every other node in the problem graph –except those that are in $O_t \setminus \{0\}$. If the vehicle’s re-optimized route uses arc (i', i) , then we say that the delivery of customer $c(i)$ is made by this vehicle at location i .

6.2 An iterative re-optimization framework

To solve the D-VRPRDL, we propose an iterative approach which dynamically handles deviations from the original customer itineraries. It starts by solving the VRPRDL with the initially provided itineraries. The delivery schedule obtained at this planning stage is implemented as long as the input data remains unchanged. However, once an itinerary update is revealed, the planned vehicle routes may become infeasible or suboptimal, and thus, they are re-optimized considering the remaining deliveries and the positions of the vehicles that have been dispatched from the depot at the beginning. The revised delivery plan is executed until the next update after which another re-optimization problem needs to be solved. Briefly, our iterative solution scheme produces an updated set of delivery routes whenever there is a change in the available input data. As mentioned earlier, each re-optimization problem is a static VRPRDL with an additional set of constraints, and solved through the branch-and-price algorithm developed for the VRPRDL. Further details on our solution approach are presented in the following subsections.

6.2.1 Constructing and preprocessing the graph of SP

When there is a change in the itinerary of a customer, the problem graph should be updated accordingly since some nodes and arcs in the existing graph can no longer be a part of any feasible solution and conversely, some nodes and arcs that were eliminated during the last preprocessing stage may now be used in a feasible route. For simplicity, instead of modifying the existing one, we construct a new, complete

graph over the node set $N^t \cup \{0\}$. We should note here that in our implementation, we make a copy of the depot node (denote it by $0'$) to differentiate between the depot as an origin and as a destination. Therefore, we define the arc set as $\{(i, j) : i \in N^t \cup \{0\}, j \in (N^t \setminus \{i\}) \cup \{0'\}\} \setminus \{(0, 0')\}$, that is, all outgoing arcs of the depot are connected the original depot node, while all its incoming arcs are connected to the copy node. After constructing a graph as explained above, we add to it the nodes in $O^t \setminus \{0\}$ along with the arcs $\{(o, j) : o \in O_t \setminus \{0\}, j \in N^t \cup \{0'\}\}$, and apply the following preprocessing steps to obtain the problem graph of SP :

1. For every $i \in N^k$, if $st_o + t_{o,i} > l_i$ or $\max\{st_o + t_{o,i}, e_i\} + t_{i,0'} > T$ or $qr_o < d_{c(i)}$, then remove the arc (o, i) from the graph.
2. Remove every $i \in N^t$ for which $\delta^-(i) = \emptyset$ from the graph.
3. For every pair of nodes $i, j \in N^k$, compute $\text{earliest_arrival_to_j_from_i} = \max\{\min_{o \in O_k} \{st_o + t_{o,i}\}, e_i\} + t_{i,j}$. If $\text{earliest_arrival_to_j_from_i} > l_j$ or $\max\{\text{earliest_arrival_to_j_from_i}, e_j\} + t_{j,0'} > T$, remove the arc (i, j) from the graph.

6.2.2 Solving the pricing problems

As mentioned earlier, the dual of the restricted master problem associated with SP yields multiple pricing problems, one for each origin in O_t . The pricing problem corresponding to a particular origin o is an ESPPTWCC defined over the subgraph induced by the node set $N^t \cup \{o, 0'\}$, where o and $0'$ are the source and the sink nodes, respectively. The pricing subroutine embedded in our branch-and-price algorithm is designed to solve a single pricing problem. Therefore, instead of modifying the algorithm to solve multiple pricing problems, we extend the problem graph of SP and solve all the pricing problems at once as a single problem. To achieve this, we add an artificial source node s^* and artificial arcs (s^*, o) for all $o \in O_t$ to the problem graph. We take the cost of each artificial arc to be zero. The time and capacity resource consumptions for arc (s^*, o) are set to st_o and $Q - qr_o$, respectively. We

designate s^* as the source node and $0'$ as the sink node, and solve the ESPPTWCC on the extended graph. Observe that in this case, any path should use exactly one of the artificial arcs and contain exactly one of the origins since there are no arcs connecting any two origin nodes. Moreover, the amount of time consumed and the quantity delivered on a feasible path p using arc (s^*, o) , i.e., a feasible path originating from o , cannot exceed $T - st_o$ and qr_o , respectively. Hence, the path obtained by joining together $p \setminus \{(s^*, o)\}$ and the path from the depot to node o corresponds to a feasible vehicle route over the planning horizon $[0, T]$.

6.2.3 Initial set of columns for SP

The master problem corresponding to the LP relaxation of a static problem is solved by means of column generation, which starts by solving a restriction of the master problem. Therefore, we need to provide an initial set of columns that guarantees the feasibility of the master problem. The quality of this solution can have a significant impact on the performance of the branch-and-price algorithm, especially for large problem instances. For initializing the first static problem, we use the feasible solution found by the heuristic algorithm of [57]. For the subsequent problems, we can derive a set of starting columns using the current solution, i.e., the set of vehicle routes obtained by solving the previous static problem. Obviously, if the current solution remains feasible after an update in the available input data, then we can simply take the unexecuted segments of the columns in the current solution and use them to initialize the next static problem. Otherwise, one of these columns will become infeasible (since itinerary updates are revealed one at a time and the customer causing the update is only in one of the routes), and the feasibility of the current solution may be restored by manipulating this infeasible column.

Suppose that c is the critical customer, i.e., the customer whose itinerary change is revealed at time t , i is the planned delivery location for c , and $[e'_i, l'_i]$ is the updated time window of this customer at location i . If the customer's time window $[e'_i, l'_i]$ and the delivery vehicle's time window $[ea_i, ld_i]$ overlap, then the planned delivery route

containing i remains feasible. In this case, we omit the executed part of each route r for which $z_r = 1$ in the previous static problem, and use the resulting set of columns to initialize the restricted master problem associated with SP . Otherwise, we try to recover a feasible starting solution by applying the procedure described below.

Let \bar{r} with $z_{\bar{r}} = 1$ be the route containing node i , \bar{r}_f be the unexecuted segment of \bar{r} , Ψ be the set of nodes in \bar{r}_f corresponding to customer locations, and $o \in O_t$ be the origin for the vehicle assigned to \bar{r} . We know that this route has become infeasible as a result of the latest update, but it may be possible to identify a feasible route by changing the customer sequence or the delivery locations for some customers in \bar{r}_f . To this end, we take the subgraph \bar{G} of the problem graph induced by the node set $\bar{N} = \bigcup_{j \in \Psi} N_{c(j)}^t \cup \{o, o'\}$, and attempt to solve a generalized traveling salesman problem with time windows (GTSPTW) to obtain an alternative route, starting from o and ending at the depot, in which all customers in the set $\bigcup_{j \in \Psi} c(j)$ are served. However, the GTSPTW defined over the graph \bar{G} may be infeasible. In that case, we invoke the route splitting subroutine in section 6.2.3.1, which tries to split \bar{r}_f in a way to produce two feasible routes. If no splitting opportunities exist, the node removal subroutine in 6.2.3.2 is invoked.

It is important to note here that in recovering a feasible solution, we do not alter the planned delivery routes that remain feasible after the update at time t . Instead, we try to achieve this by modifying the unexecuted segment of \bar{r} and considering the option to dispatch additional vehicles from the depot. However, our heuristic may fail to produce a feasible set of routes since it is not always possible to restore feasibility without changing the routes of other vehicles (e.g. a vehicle dispatched from the depot at time t may not be able to reach any $j \in N_c^t$ on time, while one of the vehicles out for delivery can). In such cases, we define an artificial out-and-back route from the depot for customer c with a very high cost, and remove i from \bar{r}_f . If this artificial column appears in the final solution of SP , it means that customer c will not be served unless another itinerary change is revealed making delivery to this customer possible again. Details of our recovery heuristic is given in Algorithm 7, where $pred(j)$ and $succ(j)$ for $j \in \bar{r}_f$ correspond to the immediate predecessor and the immediate successor of j .

6.2.3.1 Route splitting

Suppose that we have $l'_i < l_i$, that is, customer c has left location i early. Due to triangle inequality, we always know that the vehicle assigned to \bar{r} can return to the depot by time T after visiting the location that immediately precedes i in \bar{r}_f , for otherwise \bar{r} would not be feasible. Furthermore, since \bar{r} is infeasible, we also know that $l'_i < ea_i$, and thus $l'_i < ld_i$. This relation implies that even if a vehicle departs from location i at time l'_i , it can serve the subsequent customers in \bar{r}_f within their respective time windows and return to the depot on time. Hence, if a vehicle dispatched from the depot at time t can reach location i by time l'_i , it means that we can split \bar{r}_f and define two feasible routes. An example of route splitting is illustrated in Figure 6.1. Of course, there may exist alternative splitting opportunities regarding \bar{r}_f . If that is the case, we select the one that incurs the least cost among all. Similar reasoning applies when $e'_i > e_i$, that is, when customer c arrives at location i later than expected.

6.2.3.2 Node removal

If splitting fails as well, our final attempt towards restoring feasibility is to use the node removal subroutine in which we define an out-and-back route from the depot for some customers and remove their associated delivery locations from \bar{r}_f . More specifically, we first check whether there exists $j \in N_c^t$ for which $0 - j - 0'$ is feasible. If we can find such a location j in the itinerary of customer c , then the recovery procedure returns the out-and-back route $0 - j - 0'$ and the route resulting from the removal of node i from \bar{r}_f , which are then used when initializing the restricted master problem. Otherwise, starting with its immediate predecessor (in early departure case), we iterate over the nodes of \bar{r}_f that were planned to be visited prior to location i and apply the following steps. If a feasible out-and-back route does not exist for the customer associated with the current node, we move to the previous node and continue. Else, we remove the current node from \bar{r}_f , and check whether the resulting route is feasible. We stop if feasibility is achieved, and

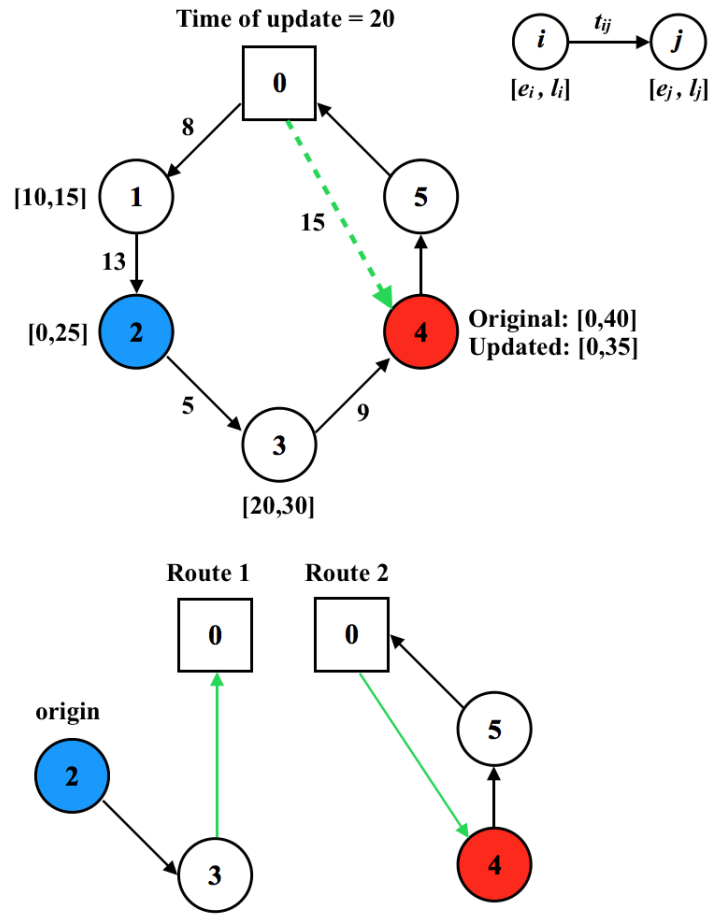


Figure 6.1: Two feasible routes obtained by route splitting

the recovery procedure returns the resulting route as well as a collection of out-and-back routes defined for the nodes removed from \bar{r}_f . Otherwise, i.e., if we still have an infeasible route, we move to the previous node and continue.

When there are no nodes left for removal, but a feasible route could not be identified, the recovery procedure returns an artificial out-and-back route for customer c and the route obtained by removing node i from the original \bar{r}_f (the one containing all nodes in $\Psi \setminus \{i\}$). Figure 6.2 depicts an example of node removal. The recovery in case of late arrival of customer to location i is handled in a similar fashion. The only difference is that we iterate over the nodes of \bar{r}_f that were planned to be visited after location i starting with the immediate successor of i if no feasible out-and-back route for customer c can be found.

Algorithm 7: *restoreFeasibility*(\bar{r}_f, i, t)

Data: Time t at which the latest update occurs, the route segment \bar{r}_f with origin $o \in O_t$, and the node i in \bar{r}_f whose time window update is revealed at time t

Result: A set of routes to be used when initializing the restricted master problem for SP

Define a GTSPTW over the subgraph of the problem graph of SP induced by $\bar{N} = \bigcup_{j \in \Psi} N_{c(j)}^t \cup \{o, o'\}$ where Ψ is the set of nodes in \bar{r}_f corresponding to customer locations

if *GTSPWTW is feasible* **then**

 Find the optimal GTSPTW tour r^*
 Return r^*

else

if $l'_i < l_i$ **then**

 Return *earlyDepartureRecovery*(\bar{r}_f, i, t)

else if $e'_i > e_i$ **then**

 Return *lateArrivalRecovery*(\bar{r}_f, i, t)

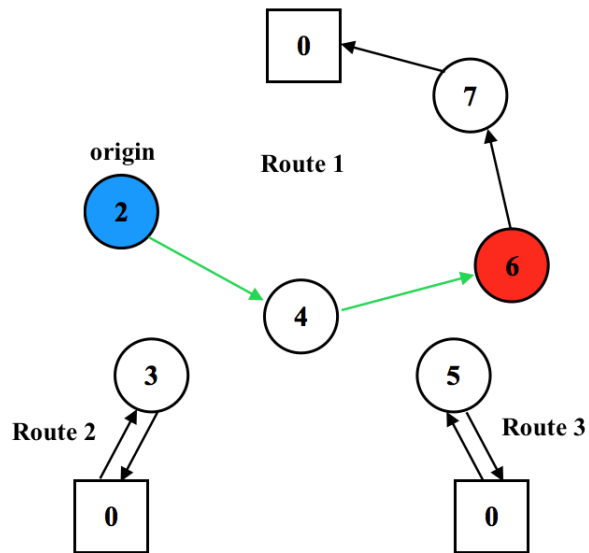
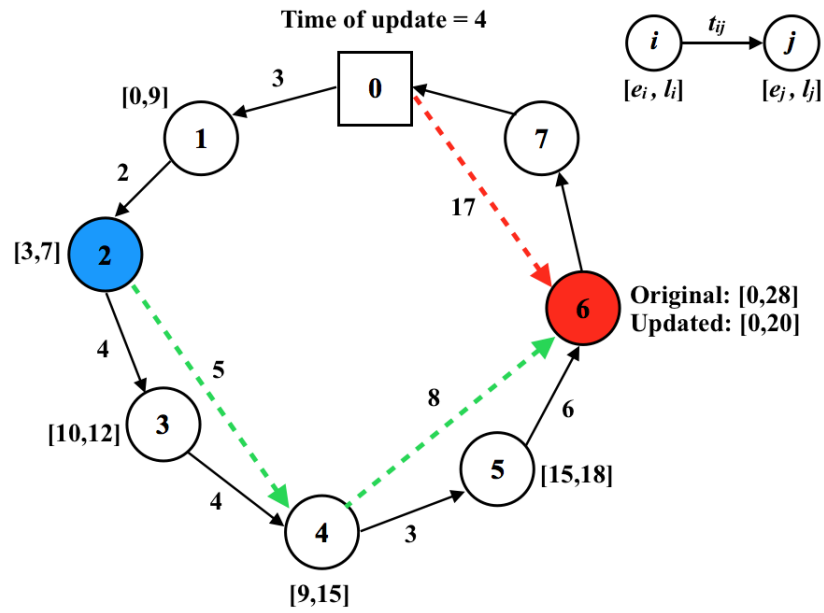


Figure 6.2: Feasible routes obtained by removing nodes 3 and 5 from the route

6.2.4 Generating more initial columns

Usually, sufficient time is available to solve the initial problem, and thus, it is possible to find optimal solutions or at least solutions of good quality during the planning stage. On the other hand, it may be necessary to solve many re-optimization problems during the execution stage, which requires producing solutions to these problems quickly. Although the computation time allocated to solve SP can be quite limited, usually a large number of pricing problems has already been solved and many columns have been generated during the previous executions of the algorithm. Transferring certain parts of this knowledge and using them when solving the current static problem, optimal or sufficiently good solutions may still be identified reasonably quickly. To this end, we propose a heuristic method that enables the iterative re-optimization scheme to pass a set of additional columns to the initial restricted master problem for SP . These are extracted both from the columns generated when solving the previous static problem and from those that were among its starting columns.

Let Ω be a set containing the columns used for initializing the restricted master problem associated with the previous static problem, and the columns generated throughout its solution procedure. Note that we exclude the columns in its final solution from Ω , since we have already described how they are processed to define an initial solution to SP . Suppose that for each route $r \in \Omega$, we keep the portion of r such that at least one vehicle can make a delivery to its first node, considering the earliest departure time of the vehicles from their respective origins (st_o for $o \in O_t$) as well as their remaining capacities (qr_o for $o \in O_t$). This is equivalent to identifying the first node in r that has not been eliminated from the problem graph of SP during the preprocessing phase, and taking the part of r from that node on. Observe that the route segment r_f extracted from r may still be infeasible due to several reasons. First, it may contain nodes and/or arcs that are not in the current problem graph. Second, it may involve a node whose time window has changed as a result of the latest update, and the amount of deviation from the original time window may be enough to render the sequence of nodes in the route segment infeasible. Finally, even

if there is a vehicle which can serve the first customer in r_f , it is not guaranteed that the vehicle will also be able to serve the subsequent customers and return to the depot without violating the time window and capacity restrictions, because the feasibility of the entire node sequence depends on the vehicle's remaining delivery capacity and the earliest departure time from the location corresponding to its origin.

A simple way to obtain a set of additional starting columns for SP would be to derive, if possible, a route segment from every $r \in \Omega$ based on the above rule and eliminate the infeasible ones. However, in that way, we may be discarding some node sequences that could yield useful columns for the branch-and-price algorithm when solving SP . Hence, for each route segment r_f extracted from Ω , we apply the five step procedure below to construct a feasible column.

1. Identify the nodes in r_f that are not in the problem graph of SP , and remove them from r_f .
2. Identify the arcs in r_f that are not in the problem graph of SP , and define new route segments r_f^1, \dots, r_f^ϕ resulting from the removal of arcs from r_f , where ϕ is the number of arcs removed plus one.
3. For every $k = 1, \dots, \phi$, assign an origin to r_f^k . Among all origins, we pick the one which yields the earliest arrival time to the first node of r_f^k . The selected origin, say o_k , is added to the beginning of r_f^k . Also, append the node $0'$ to the end of r_f^k for $k = 1, \dots, \phi - 1$.
4. For every $k = 1, \dots, \phi$, check the feasibility of r_f^k with respect to time window constraints and fix the violations, if any, by removing nodes from r_f^k .
5. For every $k = 1, \dots, \phi$, check if the total demand of the customers in r_f^k exceeds the remaining delivery capacity of the vehicle located at the origin assigned to this route. If it does, fix the capacity constraint violation by removing nodes from r_f^k . Else, the route r_f^k is returned as a feasible column to the initial restricted master problem for SP .

Further details of the last two steps are provided in the following along with a pseudocode in Algorithm 13.

6.2.4.1 Fixing time window violations

Given a route r with origin o , we consider two cases when fixing time window violations.

Case (i): r contains $j \in N_c^t$, i.e., a location in the itinerary of the critical customer, and the arc (o, j) is in the problem graph of SP . (If (o, j) is not in the graph, we set $r \leftarrow r \setminus \{j\}$ and treat the resulting route as in Case (ii)).

In this case, first, we attempt to solve a GTSP_{TW} as in the recovery heuristic given by Algorithm 7. If the GTSP_{TW} returns a solution, it means that we have found a feasible route with respect to time window restrictions. Otherwise, let $\Gamma_p = o, \dots, pred(j)$ and $\Gamma_s = succ(j), \dots, 0'$ be node sequences consisting of the nodes in r that precede and succeed j , respectively. That is, joining Γ_p , j , and Γ_s together gives the original route r . We construct a new route $r_{new} = o, j, 0'$, which is feasible by definition since the arc (o, j) is in the problem graph of SP . Starting with $i = succ(o)$, we iterate over the customer nodes in Γ_p . If the route obtained by inserting i immediately before j into r_{new} is feasible, then we update r_{new} accordingly. Else, it remains unchanged. We set $i \leftarrow succ(i)$ and insert all possible nodes of Γ_p to r_{new} .

Replacing the last node of r_{new} with the entire sequence Γ_s does not guarantee that the resulting route will satisfy the time window restrictions. Hence, we calculate edt , the earliest departure time from location j for a vehicle starting its route at time st_o and serving all the customers in r_{new} . Then, we find the first customer node in $i \in \Gamma_s$ for which $edt + t_{ji} \leq \min\{ld_i, l_i\}$. If such a customer node exists, then the vehicle leaving location j at time edt can serve $c(i)$ and the subsequent customers in Γ_s within their respective time windows and return to the depot by time T . In that case, we replace the last node of r_{new} with the subsequence of Γ_s starting with i .

Case (ii): r does not involve the critical customer c .

Starting with $\text{succ}(o)$, we iterate over the nodes in r until finding the first customer node j for which $ea_i > l_i$ or $\max\{ea_i, e_i\} + t_{i0'} > T$. Then, we let $edt = \max\{e_{\text{pred}(i), ea_{\text{pred}(i)}}\}$ and determine the first customer node j that comes after i in the route such that $edt + t_{\text{pred}(i), j} \leq \min\{l_j, ld_j\}$. Removing the sequence of nodes $i, \dots, \text{pred}(j)$ from r produces a route satisfying the time window requirements.

6.2.4.2 Fixing capacity constraint violations

Suppose that we are given a route r with origin o , which is feasible with respect to time window restrictions. We calculate totalDemand , the total demand of the customers in r to check whether the remaining delivery capacity of the vehicle originating from o will be sufficient to serve all the customers in r . If not, we set $r \leftarrow r \setminus \{\text{pred}(0')\}$ and $\text{totalDemand} \leftarrow \text{totalDemand} - d_{c(\text{pred}(0'))}$ until $qr_o \geq \text{totalDemand}$.

6.2.4.3 Storing & processing the set of columns

For implementation efficiency, we store the columns in Ω in a tree structure, which helps us process multiple columns starting with a common sequence of nodes at once. Only after the third step of the procedure described in section 6.2.4, we extract the columns from the tree as a set, and iterate over that set to fix time window and capacity violations. An example of a column tree is depicted in Figure 6.3. Every path from s^* to a leaf node $0'$ corresponds to a column.

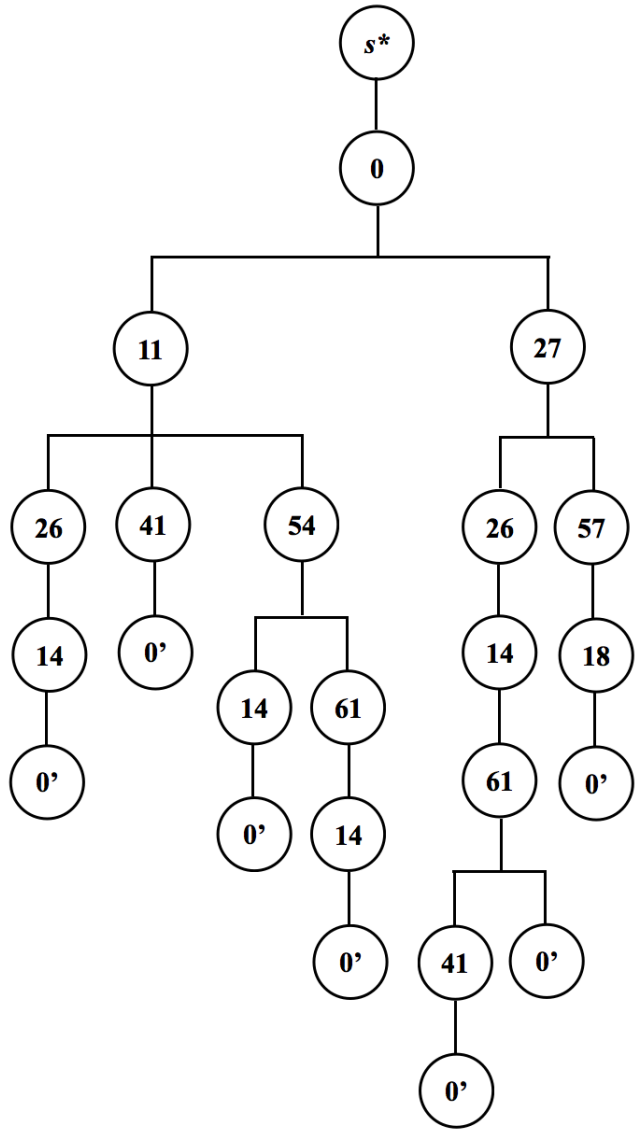


Figure 6.3: An example tree of columns

6.3 Computational study

We conduct preliminary experiments (1) to investigate the potential benefits of using the columns coming from the previous branch-and-price executions, and (2) to test the efficiency of our solution approach.

6.3.1 Test instances & update generation scheme

We perform our computations on the first set of instances used in the previous chapter. Let K be the maximum number of itinerary updates that will be revealed during the planning horizon $[0, T]$. We divide $[0, T]$ into K intervals of equal length and attempt to generate one update per each interval via the following procedure. For $k = 1, \dots, K$:

1. Randomly pick a point t in time interval $[(k - 1)T/K, kT/K]$ at which the delivery planner will be notified about the next itinerary change.
2. Randomly select a route until finding one for which the assigned vehicle has still at least one delivery to make. Stop if no such route can be found. Else, go to Step 3 with the selected route r .
3. Randomly select a node in r until finding one for which multiple locations exist in the associated customer's itinerary. Stop if no such nodes can be found. Else, go to Step 4 with the selected node i .
4. Compute the time window $[ea_i, ld_i]$ of the vehicle assigned to route r at location i , i.e., the earliest and the latest times that the vehicle can arrive at and should leave location i .
5. Decide on the type of itinerary update for customer $c(i)$. If $l_i - ea_i \leq ld_i - e_i$, we choose early departure; otherwise, we choose late arrival.

6. If the type of update is early departure, starting with i , we shrink the time windows of the nodes $j \in \{N_{c(i)} : l_j \leq l_i\}$ one at a time, until $e'_i < ea_i$ or shrinking is applied to all nodes in the sequence, whichever happens first. When we shrink $[e_j, l_j]$, the time windows associated with the succeeding nodes of j are shifted backwards to preserve the travel times between consecutive locations in $N_{c(i)}$. We assume that the total deviation resulting from the shrinkage of time windows is absorbed by the last node in $N_{c(i)}$, which corresponds to the customer's home location. Hence, for that node, we shift only the opening of the time window backwards yielding a wider time window at the home location. In late arrival case, time windows associated with the nodes $j \in \{N_{c(i)} : e_j \geq e_i\}$ are shrunk in a similar fashion. After shrinking $[e_j, l_j]$, the time windows of the nodes in $N_{c(i)}$ that precede j are shifted forward. As in the early departure case, we shift only the closing of the time window for the first location in $N_{c(i)}$. Note that when we shrink the time window for a node j , we decrease its length by $\lfloor 0.75(l_j - e_j) \rfloor$.

Notice that the above scheme may not always produce an update. For example, all vehicles may have performed their routes by time t , or the node selected in Step 3 may be the only location in the associated customer's itinerary. This is the reason why we defined K as the *maximum* number of updates earlier.

6.3.2 Implementation and experimental setup

The iterative re-optimization scheme is implemented in Java using the branch-and-price framework of Java OR library (*jORLib*) and Java graph theory library (*JGraphT*). After setting up each static problem, we invoke the enhanced branch-and-price algorithm in the previous chapter with pricing parameter configuration $(5, 5, F)$ (at most five columns are returned to the restricted master at every pricing iteration, five non-dominated labels stored at each node and multiple iterations are allowed in truncated label setting), and branching rule *MFC* (select the arc that appears most frequently in a given fractional solution, and in case of ties, the one

with value closest to 0.5).

Restricted master problems are solved with CPLEX 12.6 through Concert Technology. All experiments are performed on a single thread of a 64-bit machine with Intel Xeon E5-2630 v2 processor at 2.60 GHz. The time limit is set to two hours for each static problem. Of course, it is critical to solve these problems very quickly, but in our preliminary experiments, we wanted to observe how much time it usually takes to solve a re-optimization problem before deciding on a reasonable time limit.

6.3.3 Preliminary results

First, we carry out an experiment on small and medium sized instances by generating a single update to evaluate the performance of the branch-and-price algorithm in solving a re-optimization problem when the corresponding restricted master problem is initialized with

- (i) the columns derived from the previously planned vehicle routes,
- (ii) the columns in (i) and the columns extracted from the column tree (after eliminating the infeasible ones),
- (iii) the columns in (i) and the columns extracted from the column tree (after recovering the infeasible ones).

We will use `Soln_Cols`, `All_Feas_Cols` and `All_Cols` to denote the re-optimization strategies involving the columns in (i), (ii), and (iii), respectively.

The purpose of this experiment is twofold: to assess the potential benefits of using previously generated columns in addition to the ones in the latest delivery routes and to test the usefulness of the recovery procedures outlined in Section 6.2.4. For each of the strategies above, we present the numbers of pricing iterations, columns generated,

and nodes in the branch-and-price tree as well as the time spent for solving the re-optimization problem in Table 6.1.

The results indicate that utilizing the columns extracted from the column tree can significantly decrease the number of pricing iterations and the number of columns generated compared to the case where only the columns in (i) are present in the initial restricted master problem. However, the choice between `All_Feas_Cols` and `All_Cols` is not obvious. Although recovering the infeasible columns extracted from the column tree and using them in initialization is helpful in some cases, the strategy `All_Cols` may sometimes lead to a larger number of pricing iterations, columns, or nodes compared to `All_Feas_Cols`.

Our update generation scheme yields an update that makes the planned delivery schedule infeasible for 26 of the instances 1–30. We observe that the planned vehicle routes can be quickly re-optimized following the update. In eight of these instances, the final solution of the re-optimization problem involves an artificial column, implying that the customer with the itinerary update cannot be served by a new vehicle dispatched from the depot or by the vehicles that are already out for delivery. For the remaining 18 instances, the route involving the customer whose itinerary has changed becomes infeasible, but re-optimization produces a delivery schedule in which all customers can still be served within the planning horizon. We also note that solving the re-optimization problem requires fewer pricing iterations to be performed and a smaller number of columns to be generated, when the initial restricted master problem has a feasible solution.

For small and medium sized instances, solution times are quite short and close to one another with all three strategies `Soln_Cols`, `All_Feas_Cols` and `All_Cols`. However, the time spent by the branch-and-price algorithm tend to increase with the number of pricing iterations performed since it is the most time consuming part of the algorithm. To be able to make a more accurate comparison between the strategies `All_Feas_Cols` and `All_Cols`, we test them on large instances as well. The results are reported in Table 6.2. Note that our update generation scheme did not yield an update for Instance 39, thus, we did not include it in the table. Times spent for

re-optimization are still quite similar across the instances, but based on the average number of pricing iterations and the fact that it provides a better final solution at the end of two hours for Instance 40, we conclude that `All_Cols` may be a slightly better choice for further exploration.

It is also worth to mention that when we are able to identify an optimal or a near-optimal delivery schedule based on the original customer itineraries, we observe that solving the re-optimization problem is relatively easier. This is expected since we already have a set of very good vehicle routes to start with even if one of them becomes infeasible as a result of the update. Otherwise, the branch-and-price algorithm may spend a substantial amount of time (as in the case of Instance 40) for re-optimization depending on how far the vehicle routes determined at the beginning of the planning horizon is from the optimum.

In our final experiment, we test the efficiency of our iterative solution framework on Instances 1–30 using `All_Cols`. We allow at most $0.20|C|$ itinerary updates for each of these instances. In Table 6.3, we report the total number of updates generated by the scheme described in Section 6.3.1 and the averages regarding the number of pricing iterations performed, the number of nodes evaluated and the time spent per re-optimization problem. The results demonstrate that deviations from original itineraries can mostly be handled efficiently for small and medium sized instances. In particular, the re-optimization problems can be solved within less than five seconds on average. We should also note that although vehicles are allowed to be diverted from their planned routes and new vehicles can be dispatched from the depot whenever needed, we see that the changes in planned delivery routes are usually kept at minimum by examining the solutions of consecutive static problems.

For large instances, it is not always possible solve a re-optimization problem sufficiently fast. However, once the column generation procedure at the root node terminates, solving the resulting restricted master problem as an integer program may produce a near-optimal solution to the associated re-optimization problem in a reasonable amount of time. Moreover, constructing the column tree selectively, for

example by storing the columns generated only at the root node instead of maintaining all columns generated throughout the entire branch-and-price tree, may enhance the performance of the proposed iterative re-optimization framework.

6.4 Concluding remarks

In this study, we consider the VRPRDL in a dynamic setting, referred to as the D-VRPRDL, and develop an iterative approach which starts by solving the VRPRDL with the initially provided customer itineraries and re-optimizes the planned vehicle routes whenever an itinerary update is revealed. The D-VRPRDL is a quite challenging problem since it is critical to respond to the changes in the available input data quickly.

In our iterative solution scheme, we employ methods that facilitate the use of information collected when solving the previous re-optimization problems instead of starting from scratch. Preliminary experiments with our solution approach yield promising results in terms of efficiently handling deviations from original customer itineraries. However, there is still room for improvement especially for large problem instances. Exploring which parts of the previously collected information would be more useful when re-planning the vehicle routes can lead to further performance improvements.

Subroutine 8: *earlyDepartureRecovery*(\bar{r}_f, i, t)

```

Let  $F = \emptyset$ 
if  $t_k + t_{0i} \leq l'_i$  then
   $splitNode \leftarrow i$ 
   $minCostOfSplit = w_{pred(i),0'} + w_{0,i} - w_{pred(i),i}$ 
  Let  $u_1 = pred(i)$ ,  $u_2 = i$ , and  $LAT = l'_i$ 
  while  $u_1 \neq o$  do
    if  $t + t_{0,u_1} \leq l_{u_1}$  and  $max\{e_{u_1}, t + t_{0,u_1}\} + t_{u_1,u_2} \leq LAT$  then
      if  $w_{pred(u_1),0'} + w_{0,u_1} - w_{pred(u_1),u_1} < minCostOfSplit$  then
         $splitNode \leftarrow u_1$ 
         $minCostOfSplit \leftarrow w_{pred(u_1),0'} + w_{0,u_1} - w_{pred(u_1),u_1}$ 
         $LAT \leftarrow LAT - t_{u_1,u_2}$ 
         $u_2 \leftarrow u_1$ ,  $u_1 \leftarrow pred(u_1)$ 
      else
        Break the loop
     $F \leftarrow F \cup \{r_1, r_2\}$  where  $r_1 = (o, \dots, pred(splitNode), 0')$  and  $r_2 = (0, splitNode, \dots, 0')$ 
else
  Let  $u_0 = findNodeForOutAndBackRoute(c(i))$ 
  if  $u_0 \neq -1$  or  $pred(i) = o$  or  $st_o + t_{oi} > l'_i$  then
    Define  $r_1 = \bar{r}_f \setminus \{i\}$  and  $r_2 = (0, j, 0')$ , where  $j = u_0$  if  $u_0 \neq -1$ ; otherwise,  $j = i$  and  $r_2$  is an
    artificial column
     $F \leftarrow F \cup \{r_1, r_2\}$ 
  else
    Let  $u_1 = pred(i)$ ,  $u_2 = i$ ,  $LAT = l'_i$ , and  $H_0, H_1 = \emptyset$ 
    while  $u_1 \neq o$  do
       $u_0 \leftarrow findNodeForOutAndBackRoute(c(u_1))$ 
      if  $u_0 \neq -1$  then
         $H_0 \leftarrow H_0 \cup \{u_0\}$  and  $H_1 \leftarrow H_1 \cup \{u_1\}$ 
        if  $pred(u_1) = o$  or  $max\{e_{a_{pred(u_1)}}, e_{pred(u_1)}\} + t_{pred(u_1),u_2} \leq LAT$  then
          for  $j \in H_0$  do
             $F \leftarrow F \cup \{(0, j, 0')\}$ 
           $F \leftarrow F \cup \{\bar{r}_f \setminus H_1\}$ 
          Break the loop
        else
           $u_1 \leftarrow pred(u_1)$ 
      else if  $pred(u_1) \neq o$  and  $max\{st_o + t_{o,u_1}, e_{u_1}\} + t_{u_1,u_2} \leq LAT$  then
         $LAT \leftarrow LAT - t_{u_1,u_2}$ 
         $u_2 \leftarrow u_1$ ,  $u_1 \leftarrow pred(u_1)$ 
      else
        Define  $r_1 = \bar{r}_f \setminus \{i\}$  and  $r_2 = (0, i, 0')$ , where  $r_2$  is an artificial column
         $F \leftarrow F \cup \{r_1, r_2\}$ 
        Break the loop
     $F \leftarrow F \cup \{r_1, r_2\}$ 
  Return  $F$ 

```

Subroutine 9: *lateArrivalRecovery*(\bar{r}_f, i, t)

```

Let  $F = \emptyset$ 
if  $\text{succ}(i) = 0'$  or  $e'_i + t_{i0'} > T$  then
  Let  $u_0 = \text{findNodeForOutAndBackRoute}(c(i))$ 
  Define  $r_1 = \bar{r}_f \setminus \{i\}$  and  $r_2 = (0, j, 0')$ , where  $j = u_0$  if  $u_0 \neq -1$ ; otherwise,  $j = i$  and  $r_2$  is an artificial
  column
   $F \leftarrow F \cup \{r_1, r_2\}$ 
else
  Let  $\text{splitNode} = -1$ ,  $\text{minCostOfSplit} = +\infty$ ,  $u_1 = i$ ,  $u_2 = \text{succ}(i)$  and  $\text{EDT} = e'_i$ 
  while  $u_2 \neq 0'$  do
    if  $t + t_{0,u_2} \leq \min\{l_{u_2}, ld_{u_2}\}$  and  $w_{u_1,0'} + w_{0,u_2} - w_{u_1,u_2} < \text{minCostOfSplit}$  then
       $\text{splitNode} \leftarrow u_2$ 
       $\text{minCostOfSplit} \leftarrow w_{u_1,0'} + w_{0,u_2} - w_{u_1,u_2}$ 
    if  $\min\{l_{u_2}, T - t_{u_2,0'}\} - t_{u_1,u_2} \geq \text{EDT}$  then
       $\text{EDT} \leftarrow \text{EDT} + t_{u_1,u_2}$ 
       $u_1 \leftarrow u_2$ ,  $u_2 \leftarrow \text{succ}(u_2)$ 
    else
      Break the loop
  if  $\text{splitNode} \neq -1$  then
     $F \leftarrow F \cup \{r_1, r_2\}$  where  $r_1 = (0, \dots, \text{pred}(\text{splitNode}), 0')$  and  $r_2 = (0, \text{splitNode}, \dots, 0')$ 
  else
    Let  $u_0 = \text{findNodeForOutAndBackRoute}(c(i))$ 
    if  $u_0 \neq -1$  then
       $F \leftarrow F \cup \{\bar{r}_f \setminus \{i\}, (0, u_0, 0')\}$ 
    else
       $u_1 \leftarrow i$ ,  $u_2 \leftarrow \text{succ}(i)$  and  $\text{EDT} \leftarrow e'_i$ 
      Let  $H_0, H_1 = \emptyset$ 
      while  $u_2 \neq 0'$  do
         $u_0 \leftarrow \text{findNodeForOutAndBackRoute}(c(u_2))$ 
        if  $u_0 \neq -1$  then
           $H_0 \leftarrow H_0 \cup \{u_0\}$  and  $H_1 \leftarrow H_1 \cup \{u_2\}$ 
          if  $\text{succ}(u_2) = 0'$  or  $\text{EDT} + t_{u_1, \text{succ}(u_2)} \leq \min\{l_{\text{succ}(u_2)}, ld_{\text{succ}(u_2)}\}$  then
            for  $j \in H_0$  do
               $F \leftarrow F \cup \{(0, j, 0)\}$ 
             $F \leftarrow F \cup \{\bar{r}_f \setminus \{i\}\}$ 
            Break the loop
          else
             $u_2 \leftarrow \text{succ}(u_2)$ 
          else if  $\text{succ}(u_2) \neq 0'$  and  $\text{EDT} + t_{u_1, u_2} \leq \min\{l_{u_2}, T - t_{u_2, 0'}\}$  then
             $\text{EDT} \leftarrow \text{EDT} + t_{u_1, u_2}$ 
             $u_1 \leftarrow u_2$ ,  $u_2 \leftarrow \text{succ}(u_2)$ 
          else
            Define  $r_1 = \bar{r}_f \setminus \{i\}$  and  $r_2 = (0, i, 0')$ , where  $r_2$  is an artificial column
             $F \leftarrow F \cup \{r_1, r_2\}$ 
            Break the loop
  Return  $F$ 

```

Subroutine 10: $findNodeForOutAndBackRoute(c, t)$

Let $node = -1$ and $minCost = +\infty$

for $j \in N_c^t$ **do**

if $t + t_{0j} \leq l_j$ and $\max\{t + t_{0j}, e_j\} + t_{j0'} \leq T$ and $w_{0j} < minCost$ **then**
 $node \leftarrow j$
 $minCost \leftarrow w_{0,j}$

Return $node$

Subroutine 11: $fixTWViolationsWithCriticalCustomer(r, criticalNode)$

if r is infeasible wrt time window restrictions **then**

 Let $i = criticalNode$

 Define a GTSP_{TW} over the subgraph of the problem graph of SP induced by $\bar{N} = \bigcup_{j \in \Psi} N_{c(j)}^t \cup \{o, 0'\}$

 where Ψ is the set of nodes in \bar{r} corresponding to customer locations

if GTSP_{TW} is feasible **then**

 Find the optimal GTSP_{TW} tour r^*

$r \leftarrow r^*$

else

 Let $u_1 = o$, $u_2 = succ(o)$, and $edt = st_o$

while $u_2 \neq criticalNode$ **do**

 Let $eat = edt + t_{u_1, u_2}$

if $eat \leq l_{u_2}$ and $\max\{eat, e_{u_2}\} + t_{u_2, i} \leq \min\{l_i, T - t_{i, 0'}\}$ **then**

$edt \leftarrow \max\{eat, e_{u_2}\}$

$u_1 \leftarrow u_2$, $u_2 \leftarrow succ(u_2)$

else

$r \leftarrow r \setminus \{u_2\}$

$u_2 \leftarrow succ(u_2)$

$edt \leftarrow \max\{e_i, edt + t_{u_1, i}\}$ and $u_2 \leftarrow succ(u_2)$

while $u_2 \neq 0'$ **do**

if $edt + t_{i, u_2} > \min\{l_{u_2}, ld_{u_2}\}$ **then**

$r \leftarrow r \setminus \{u_2\}$

$u_2 \leftarrow succ(u_2)$

else

 Break the loop

Return r

Subroutine 12: *fixTWViolationsWithoutCriticalCustomer(r)*

```
if  $r$  is infeasible wrt time window restrictions then
  Let  $u_1 = o$ ,  $u_2 = \text{succ}(o)$ , and  $\text{edt} = st_o$ 
  outer_loop
  while  $u_2 \neq 0'$  do
    Let  $\text{eat} = \text{edt} + t_{u_1, u_2}$ 
    if  $\text{eat} \leq \min\{l_{u_2}, T - t_{u_2, 0'}\}$  then
       $\text{edt} \leftarrow \max\{\text{eat}, e_{u_2}\}$ 
       $u_1 \leftarrow u_2$ ,  $u_2 \leftarrow \text{succ}(u_2)$ 
    else
       $r \leftarrow r \setminus \{u_2\}$  and  $u_2 \leftarrow \text{succ}(u_2)$ 
      while  $u_2 \neq 0'$  do
        if  $\text{edt} + t_{u_1, u_2} > \min\{l_{u_2}, ld_{u_2}\}$  then
           $r \leftarrow r \setminus \{u_2\}$ 
           $u_2 \leftarrow \text{succ}(u_2)$ 
        else
          Break outer_loop
  Return  $r$ 
```

Algorithm 13: *fixTWAndCapacityViolations(r, c)*

```
Let  $\text{criticalNode} = -1$ 
for  $j \in r$  do
  if  $c(j) = c$  then
     $\text{criticalNode} \leftarrow j$ 
    Break the loop

if  $\text{criticalNode} \neq -1$  and  $(o, \text{criticalNode}) \notin A^k$  then
   $r \leftarrow r \setminus \{\text{criticalNode}\}$ 
   $\text{criticalNode} \leftarrow -1$ 

if  $\text{criticalNode} \neq -1$  then
   $r \leftarrow \text{fixTWViolationsWithCriticalCustomer}(r_f, \text{criticalNode})$ 
else
   $r \leftarrow \text{fixTWViolationsWithoutCriticalCustomer}(r_f)$ 

Let  $\text{totalDemand} = 0$ ,  $u_1 = \text{succ}(o)$ 
for  $j \in r_f$  do
   $\text{totalDemand} \leftarrow \text{totalDemand} + d_{c(j)}$ 

while  $\text{totalDemand} > qr_o$  and  $\text{pred}(0') \neq o$  do
   $\text{totalDemand} \leftarrow \text{totalDemand} - d_{c(\text{pred}(0'))}$ 
   $r \leftarrow r \setminus \{\text{pred}(0')\}$ 

Return  $r$ 
```

Table 6.1: Results with Soln_Cols, All_Feas_Cols, and All_Cols strategies on small and medium instances

Instance	Pricing iterations		Columns generated		Nodes evaluated		Time (s)	
	Soln_Cols	All_Feas_Cols	Soln_Cols	All_Feas_Cols	Soln_Cols	All_Feas_Cols	Soln_Cols	All_Feas_Cols
1	6	3	15	6	5	1	0.041	0.029
2	3	2	3	1	0	1	0.004	0.002
3	6	6	16	20	20	1	0.011	0.01
4	10	6	40	25	25	1	0.042	0.022
5	11	7	33	18	18	1	0.012	0.013
6	5	5	17	13	4	1	0.089	0.052
7	1	1	0	0	0	1	0.002	0.001
8	4	4	14	13	13	1	0.005	0.003
9	15	8	60	24	18	1	0.064	0.021
10	9	5	24	13	16	1	0.034	0.062
11	5	3	18	7	6	1	0.006	0.004
12	3	2	3	2	2	1	0.003	0.002
13	84	43	273	117	104	11	0.269	0.252
14	8	7	27	30	21	1	0.012	0.014
15	10	11	35	38	35	1	0.022	0.024
16	4	6	8	19	24	1	0.008	0.015
17	41	17	170	58	64	3	0.295	0.139
18	3	1	9	0	0	1	0.004	0.002
19	12	3	23	10	10	1	0.036	0.017
20	40	7	169	26	43	1	0.34	0.118
21	9	6	38	23	23	1	0.029	0.019
22	180	71	850	326	470	1	10.512	4.751
23	14	6	58	17	13	1	0.08	0.035
24	232	127	1054	549	462	3	8.668	6.307
25	25	15	50	51	27	1	0.118	0.091
26	15	16	67	58	42	1	0.059	0.039
27	123	71	484	332	297	1	3.613	3
28	17	4	72	15	12	1	0.103	0.036
29	1066	498	3363	1246	1093	165	33.158	34.771
30	82	51	375	218	212	1	2.498	2.01
Average	68.10	33.73	245.60	109.17	102.63	6.93	2.00	1.73

Table 6.2: Results with All_Feas_Cols and All_Cols on large instances

Instance	Pricing iterations		Columns generated		Nodes evaluated		Time (s)	
	All_Feas_Cols	All_Cols	All_Feas_Cols	All_Cols	All_Feas_Cols	All_Cols	All_Feas_Cols	All_Cols
31	73	60	312	257	1	1	14.469	13.853
32	24	31	102	118	1	1	0.482	0.788
33	117	112	384	315	11	13	5.698	5.334
34	40	40	136	156	1	1	21.006	33.018
35	54	38	250	173	1	1	6.78	4.038
36	46	36	181	170	1	1	2.575	1.955
37	3	3	3	3	1	1	0.01	0.013
38	5	4	12	10	1	1	0.015	0.013
40	9514	8777	35038	32257	518	573	7200	7200
Average	1097.33	1011.22	4046.44	3717.67	59.56	65.89	805.67	806.56

Table 6.3: Results obtained by iterative framework with All_Cols on small and medium instances

Instance	Updates	Avg iters	Avg nodes	Avg time (s)
1	2	3	1	0.154
2	2	2.5	1	0.043
3	3	5	1	0.106
4	3	3	1	0.017
5	2	6	1	0.017
6	2	4	1	0.098
7	3	4.33	1	0.369
8	3	4.33	1	0.032
9	2	3.5	1	0.039
10	2	8	1	0.066
11	5	3.2	1	0.035
12	5	2.6	1	0.058
13	6	4.33	1	0.03
14	5	4.6	1	0.046
15	6	13	1	0.327
16	5	6	1	0.075
17	4	8.5	2	0.2
18	6	10.17	1.33	0.15
19	5	25.2	7.8	0.811
20	5	5.8	1	0.066
21	8	10.63	1	0.557
22	12	16.83	1	1.019
23	11	11.55	1.18	0.414
24	10	20.3	1.2	1.69
25	10	8.3	1	0.491
26	12	7.67	1	0.182
27	12	20.42	1.83	0.808
28	11	5.18	1	2.497
29	11	46.82	11.36	3.488
30	10	13.4	1	0.581

Chapter 7

Conclusion

This thesis focuses on designing efficient optimization algorithms for different variants of the VRP, which was introduced in [1] and has attracted the attention of many researchers for the last six decades. Even in its simplest form, the VRP is an NP-hard problem. The problems encountered in practice usually involve complicating constraints and they are quite large in size. Nevertheless, the largest instance solved optimally contains no more than a few hundred customers for most VRP variants despite the day by day increasing computing power and the enhanced performance of the state-of-the-art MIP solvers. Hence, majority of the solution approaches proposed for the VRPs are based on heuristics, and the literature on exact algorithms is relatively scarce.

The first problem considered in this thesis is the SDVRP for which we present an arc flow formulation using vehicle-indexed variables, and derive a relaxation (R-SDVRP) by aggregating these variables over all vehicle indices. This relaxation may have solutions in which two or more vehicles exchange loads at some customers. We propose a polynomial time procedure to detect such customers, and devise two novel exact algorithms, namely patching and node-split algorithms, both of which work by extending the R-SDVRP iteratively, i.e., by adding variables and constraints

associated with the customer nodes violating the feasibility of the solution, until the solution of the resulting relaxation is feasible to the SDVRP. We also introduce two extensions of the SDVRP. The first one is the SDVRP with at most r splits. Although allowing the demand of each customer to be split and served by multiple vehicles may yield a less expensive set of vehicle routes compared to the case without the split delivery option, it may not be desirable to have many vehicles serving the demand of a customer from a practical point of view. Therefore, we consider the SDVRP in a setting where the number of deliveries made by separate vehicles is restricted from above by r . The second extension is the SDVRP with open routes in which the vehicles are not required to return to the depot after completing their service. Both extensions can be solved by slightly modifying the R-SDVRP and our algorithms.

The results of our computational study demonstrate that the most time consuming part of our algorithms is the solution of the R-SDVRP. If the R-SDVRP can be solved quickly, then our algorithms can effectively iterate until finding an optimal solution of the SDVRP (or the extension under consideration). Nevertheless, the R-SDVRP is a mixed integer program and solving it is a difficult task, especially for large sized problem instances. A useful direction for future research would be to focus on how to handle this relaxation more efficiently. For instance, new classes of valid inequalities for the SDVRP can be investigated to strengthen the lower bounds of the R-SDVRP, which may make it possible to tackle larger instances. In addition, our algorithms provide means to solve a large and symmetric formulation by solving a series of relaxations involving only a subset of the variables and constraints that cause symmetry in the original formulation. Therefore, both the patching and the node-split approaches can be adopted when solving symmetric problems other than the SDVRP. Exploring the iterative extension idea further on different problems can also be a worthwhile contribution.

Secondly, we study the TCMCSP which aims to determine a tour over a subset of the demand points in a given network so as to maximize the demand covered when there is an upper bound on the tour length. We present flow and cut based formulations for the problem as well as valid inequalities. Since the cut formulation involves

an exponential number of connectivity constraints, we develop different branch-and-cut schemes in which the violated connectivity constraints and valid inequalities are separated throughout the search tree when solving the cut formulation. The results of our computational study prove the effectiveness of the proposed valid inequalities in strengthening the linear relaxation bounds and in speeding up the solution procedure. We also discuss the relation between the optimal solution structure and the problem parameters based on the results of our computations.

The TCMCSP is a single vehicle routing problem. Thus, a possible direction for future research could be to study the multiple vehicle extension of the problem. It can be challenging to solve this extension, because in the presence of multiple vehicles, imposing a restriction on the length of each tour requires additional variables and constraints. Another direction for future research is to consider demand uncertainty. In the TCMCSP, we assume that we have deterministic information about the demand points (and their demands). Relaxing this assumption may provide useful insights into the structure of optimal tours when the demands are uncertain. Note here that our branch-and-cut algorithms, with a slight modification, are capable of solving a robust version of the TCMCSP where the demand of each demand point come from an interval uncertainty set.

The third problem considered in this thesis is the VRPRDL, in which a customer's order has to be delivered to the trunk of the customer's car during the time that the car is parked at one of the locations in the customer's (known) travel itinerary. We formulate the VRPRDL as a set-partitioning problem and present an efficient branch-and-price algorithm to solve it. We use this algorithm also for solving a more general version of the problem where a hybrid delivery strategy is adopted, allowing the delivery planner to make optimal use of the trunk delivery option. In particular, the hybrid strategy allows a delivery to either a customer's home any time during the day or to the trunk of the customer's car when it is parked at one of the locations in the customer's itinerary. We evaluate the effectiveness of our branch-and-price algorithm by conducting an extensive computational study. Furthermore, we analyze the benefits of using trunk delivery option against a pure home delivery strategy and conclude that approximately 20% cost savings can be achieved with

the hybrid delivery strategy. It is important to mention that since the VRPRDL is a GVRPTW in which the sets of delivery locations for the customers form the clusters, and the GVRPTW generalizes several variants of the VRP, our branch-and-price algorithm can be used to solve instances of these variants as well.

Finally, we study the VRPRDL in a dynamic setting, where there may be changes in the initially provided customer itineraries during the execution stage, and as a result, the planned delivery schedule may become infeasible or suboptimal. We develop an iterative framework which starts by solving the VRPRDL with the original itineraries to obtain an initial set of vehicle routes, and re-optimizes the vehicle routes whenever an itinerary update is revealed. This is achieved by solving a series of static problems over the planning horizon. Every re-optimization problem is a VRPRDL with an additional set of constraints specifying where the vehicles out for delivery should originate from in the revised routing plan.

Our iterative re-optimization scheme executes the branch-and-price approach developed for the VRPRDL to solve each static problem. Usually, sufficient time is available to solve the first problem, so optimal solutions or at least solutions of good quality can be identified during the planning stage. However, it may be necessary to re-optimize the vehicle routes frequently during the execution stage, implying that the computation time allocated to solve a re-optimization problem can be quite limited. To save some time when solving a re-optimization problem, we propose heuristic methods that enable us to utilize the columns generated throughout the previous branch-and-price executions. We conduct a preliminary computational analysis and report the results.

The VRPRDL may be yielding vehicle routes that are highly sensitive to changes in customer itineraries. It would be interesting to evaluate the robustness of the VRPRDL solutions, and explore whether we end up with a better overall delivery schedule for the dynamic VRPRDL, if we take into account the uncertainties regarding customer itineraries when planning the vehicle routes. For example, given the time window $[e_i, l_i]$ for every node i , the vehicle routes determined based on $[e_i + \zeta_i, l_i - \zeta_i]$ for some $\zeta_i > 0$ are guaranteed to remain feasible as long as the

deviation in $[e_i, l_i]$ does not exceed ζ_i (provided that i is in one of the routes). Even if the deviation is larger than ζ_i , it may be easier to recover the infeasibility of the planned vehicle routes. Obviously, robustness of these routes comes at the expense of higher transportation costs. However, in a dynamic setting, it may actually yield a less expensive overall delivery schedule.

Bibliography

- [1] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” *Management science*, vol. 6, no. 1, pp. 80–91, 1959.
- [2] M. R. Garey and D. S. Johnson, “A guide to the theory of np-completeness,” *WH Freeman, New York*, vol. 70, 1979.
- [3] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*. SIAM, 2014.
- [4] M. Dror and P. Trudeau, “Savings by split delivery routing,” *Transportation Science*, vol. 23, no. 2, pp. 141–145, 1989.
- [5] M. Dror and P. Trudeau, “Split delivery routing,” *Naval Research Logistics*, vol. 37, no. 3, pp. 383–402, 1990.
- [6] C. Archetti, M. Speranza, and A. Hertz, “A tabu search algorithm for the split delivery vehicle routing problem,” *Transportation Science*, vol. 40, no. 1, pp. 64–73, 2006.
- [7] M. Boudia, C. Prins, and M. Reghioui, “An effective memetic algorithm with population management for the split delivery vehicle routing problem,” in *Hybrid Metaheuristics*, pp. 16–30, Springer, 2007.
- [8] E. Mota, V. Campos, and Á. Corberán, “A new metaheuristic for the vehicle routing problem with split demands,” in *Evolutionary Computation in Combinatorial Optimization*, pp. 121–129, Springer, 2007.

- [9] S. Chen, B. Golden, and E. Wasil, “The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results,” *Networks*, vol. 49, no. 4, pp. 318–329, 2007.
- [10] M. Jin, K. Liu, and B. Eksioglu, “A column generation approach for the split delivery vehicle routing problem,” *Operations Research Letters*, vol. 36, no. 2, pp. 265–270, 2008.
- [11] A. Khmelev and Y. Kochetov, “A hybrid local search for the split delivery vehicle routing problem,” *International Journal of Artificial Intelligence*, vol. 13, no. 1, pp. 147–164, 2015.
- [12] R. Aleman, X. Zhang, and R. Hill, “An adaptive memory algorithm for the split delivery vehicle routing problem,” *Journal of Heuristics*, vol. 16, no. 3, pp. 441–473, 2010.
- [13] R. Aleman and R. Hill, “A tabu search with vocabulary building approach for the vehicle routing problem with split demands,” *International Journal of Metaheuristics*, vol. 1, no. 1, pp. 55–80, 2010.
- [14] J. Wilck-IV and T. Cavalier, “A genetic algorithm for the split delivery vehicle routing problem,” *American Journal of Operations Research*, vol. 2, pp. 207–216, 2012.
- [15] J. Wilck-IV and T. Cavalier, “A construction heuristic for the split delivery vehicle routing problem,” *American Journal of Operations Research*, vol. 2, pp. 153–162, 2012.
- [16] M. Silva, A. Subramanian, and L. Ochi, “An iterated local search heuristic for the split delivery vehicle routing problem,” *Computers & Operations Research*, vol. 53, pp. 234–249, 2015.
- [17] L. Berbotto, S. García, and F. Nogales, “A randomized granular tabu search heuristic for the split delivery vehicle routing problem,” *Annals of Operations Research*, vol. 222, no. 1, pp. 153–173, 2014.

- [18] C. Archetti and M. Speranza, “Vehicle routing problems with split deliveries,” *International Transactions in Operational Research*, vol. 19, no. 1-2, pp. 3–22, 2012.
- [19] W. Yi and A. Kumar, “Ant colony optimization for disaster relief operations,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 43, no. 6, pp. 660 – 672, 2007.
- [20] D. Gulczynski, B. Golden, and E. Wasil, “The split delivery vehicle routing problem with minimum delivery amounts,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 46, no. 5, pp. 612 – 626, 2010.
- [21] L. Ozdamar and O. Demir, “A hierarchical clustering and routing procedure for large scale disaster relief logistics planning,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 3, pp. 591 – 602, 2012.
- [22] M. Sahin, G. Cavuslar, T. Oncan, G. Sahin, and D. T. Aksu, “An efficient heuristic for the multi-vehicle one-to-one pickup and delivery problem with split loads,” *Transportation Research Part C: Emerging Technologies*, vol. 27, pp. 169 – 188, 2013.
- [23] Q. Chen, K. Li, and Z. Liu, “Model and algorithm for an unpaired pickup and delivery vehicle routing problem with split loads,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 69, pp. 218 – 235, 2014.
- [24] H. Wang, L. Du, and S. Ma, “Multi-objective open location-routing model with split delivery for optimized relief distribution in post-earthquake,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 69, pp. 160–179, 2014.
- [25] M. Dror, G. Laporte, and P. Trudeau, “Vehicle routing with split deliveries,” *Discrete Applied Mathematics*, vol. 50, no. 3, pp. 239 – 254, 1994.
- [26] G. Sierksma and G. Tijssen, “Routing helicopters for crew exchanges on off-shore locations,” *Annals of Operations Research*, vol. 76, pp. 261–286, 1998.

- [27] J. Belenguer, M. Martinez, and E. Mota, “A lower bound for the split delivery vehicle routing problem,” *Operations Research*, vol. 48, no. 5, pp. 801–810, 2000.
- [28] C. Lee, M. Epelman, C. White, and Y. Bozer, “A shortest path approach to the multiple-vehicle routing problem with split pick-ups,” *Transportation Research Part B: Methodological*, vol. 40, no. 4, pp. 265–284, 2006.
- [29] M. Jin, K. Liu, and R. Bowden, “A two-stage algorithm with valid inequalities for the split delivery vehicle routing problem,” *International Journal of Production Economics*, vol. 105, no. 1, pp. 228–242, 2007.
- [30] A. Ceselli, G. Righini, and M. Salani, “Column generation for the split delivery vehicle routing problem,” tech. rep., Technical report, University of Milan-DTI-Note del Polo, 2009.
- [31] L. Moreno, M. Aragão, and E. Uchoa, “Improved lower bounds for the split delivery vehicle routing problem,” *Operations Research Letters*, vol. 38, no. 4, pp. 302–306, 2010.
- [32] C. Archetti, N. Bianchessi, and M. Speranza, “A column generation approach for the split delivery vehicle routing problem,” *Networks*, vol. 58, no. 4, pp. 241–254, 2011.
- [33] G. Desaulniers, “Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows,” *Operations Research*, vol. 58, no. 1, pp. 179–192, 2010.
- [34] C. Archetti, N. Bianchessi, and M. Speranza, “Branch-and-cut algorithms for the split delivery vehicle routing problem,” *European Journal of Operational Research*, vol. 238, no. 3, pp. 685–698, 2014.
- [35] J. R. Current and D. A. Schilling, “The covering salesman problem,” *Transportation Science*, vol. 23, no. 3, pp. 208–213, 1989.

- [36] E. M. Arkin and R. Hassin, “Approximation algorithms for the geometric covering salesman problem,” *Discrete Applied Mathematics*, vol. 55, no. 3, pp. 197–218, 1994.
- [37] J. R. Current and D. A. Schilling, “The median tour and maximal covering tour problems: Formulations and heuristics,” *European Journal of Operational Research*, vol. 73, no. 1, pp. 114–126, 1994.
- [38] B. L. Golden, Z. Naji-Azimi, S. Raghavan, M. Salari, and P. Toth, “The generalized covering salesman problem,” *INFORMS Journal on Computing*, vol. 24, no. 4, pp. 534–553, 2012.
- [39] M. H. Shaelaie, M. Salari, and Z. Naji-Azimi, “The generalized covering traveling salesman problem,” *Applied Soft Computing*, vol. 24, pp. 867–878, 2014.
- [40] Z. Naji-Azimi and M. Salari, “The time constrained maximal covering salesman problem,” *Applied Mathematical Modelling*, vol. 38, no. 15, pp. 3945–3957, 2014.
- [41] M. Gendreau, G. Laporte, and F. Semet, “The covering tour problem,” *Operations Research*, vol. 45, no. 4, pp. 568–576, 1997.
- [42] M. J. Hodgson, G. Laporte, and F. Semet, “A covering tour model for planning mobile health care facilities in Suhum District, Ghana,” *Journal of Regional Science*, vol. 38, no. 4, pp. 621–638, 1998.
- [43] L. Motta, L. S. Ochi, and C. Martinhon, “Grasp metaheuristics for the generalized covering tour problem,” in *MIC2001-4th Metaheuristics International Conference Porto, Portugal*, Citeseer, 2001.
- [44] R. Baldacci, M. A. Boschetti, V. Maniezzo, and M. Zamboni, “Scatter search methods for the covering tour problem,” in *Metaheuristic Optimization via Memory and Evolution*, pp. 59–91, Springer, 2005.
- [45] P. Kubik, *Heuristic solution approaches for the covering tour problem*. PhD thesis, Universität Wien, 2007.

- [46] M. Hachicha, M. J. Hodgson, G. Laporte, and F. Semet, “Heuristics for the multi-vehicle covering tour problem,” *Computers & Operations Research*, vol. 27, no. 1, pp. 29–42, 2000.
- [47] Z. Naji-Azimi, J. Renaud, A. Ruiz, and M. Salari, “A covering tour approach to the location of satellite distribution centers to supply humanitarian aid,” *European Journal of Operational Research*, vol. 222, no. 3, pp. 596–605, 2012.
- [48] W. A. Oliveira, M. P. Mello, A. C. Moretti, and E. F. Reis, “The multi-vehicle covering tour problem: building routes for urban patrolling,” *arXiv:1309.5502*, 2013.
- [49] N. Jozefowicz, F. Semet, and E. G. Talbi, “The bi-objective covering tour problem,” *Computers & Operations Research*, vol. 34, no. 7, pp. 1929–1942, 2007.
- [50] P. C. Nolz, K. F. Doerner, W. J. Gutjahr, and R. F. Hartl, “A bi-objective metaheuristic for disaster relief operation planning,” in *Advances in Multi-objective Nature Inspired Computing*, pp. 167–187, Springer, 2010.
- [51] F. Tricoire, A. Graf, and W. J. Gutjahr, “The bi-objective stochastic covering tour problem,” *Computers & Operations Research*, vol. 39, no. 7, pp. 1582–1592, 2012.
- [52] D. Feillet, P. Dejax, and M. Gendreau, “Traveling salesman problems with profits,” *Transportation Science*, vol. 39, no. 2, pp. 188–205, 2005.
- [53] B. L. Golden, L. Levy, and R. Vohra, “The orienteering problem,” *Naval Research Logistics*, vol. 34, no. 3, pp. 307–318, 1987.
- [54] G. Laporte and S. Martello, “The selective travelling salesman problem,” *Discrete Applied Mathematics*, vol. 26, no. 2-3, pp. 193–207, 1990.
- [55] S. Kataoka and S. Morito, “An algorithm for single constraint maximum collection problem,” *Journal of the Operations Research Society of Japan*, vol. 31, no. 4, pp. 515–530, 1988.

- [56] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, “The orienteering problem: A survey,” *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.
- [57] D. Reyes, M. Savelsbergh, and A. Toriello, “Vehicle routing with roaming delivery locations,” *Transportation Research Part C: Emerging Technologies*, vol. 80, pp. 71–91, 2017.
- [58] M. Savelsbergh, “Local search in routing problems with time windows,” *Annals of Operations research*, vol. 4, no. 1, pp. 285–305, 1985.
- [59] M. M. Solomon, “Algorithms for the vehicle routing and scheduling problems with time window constraints,” *Operations Research*, vol. 35, no. 2, pp. 254–265, 1987.
- [60] M. Desrochers, J. K. Lenstra, M. W. Savelsbergh, and F. Soumis, “Vehicle routing with time windows: optimization and approximation,” *Vehicle routing: Methods and studies*, vol. 16, pp. 65–84, 1988.
- [61] M. Desrochers, J. Desrosiers, and M. Solomon, “A new optimization algorithm for the vehicle routing problem with time windows,” *Operations Research*, vol. 40, no. 2, pp. 342–354, 1992.
- [62] S. Dabia, S. Ropke, T. Van Woensel, and T. De Kok, “Branch and price for the time-dependent vehicle routing problem with time windows,” *Transportation Science*, vol. 47, no. 3, pp. 380–396, 2013.
- [63] A. Agra, M. Christiansen, R. Figueiredo, L. M. Hvattum, M. Poss, and C. Requejo, “The robust vehicle routing problem with time windows,” *Computers & Operations Research*, vol. 40, no. 3, pp. 856–866, 2013.
- [64] M. Schneider, A. Stenger, and D. Goeke, “The electric vehicle-routing problem with time windows and recharging stations,” *Transportation Science*, vol. 48, no. 4, pp. 500–520, 2014.

- [65] D. Taş, M. Gendreau, N. Dellaert, T. Van Woensel, and A. De Kok, “Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach,” *European Journal of Operational Research*, vol. 236, no. 3, pp. 789–799, 2014.
- [66] Ç. Koç, T. Bektaş, O. Jabali, and G. Laporte, “A hybrid evolutionary algorithm for heterogeneous fleet vehicle routing problems with time windows,” *Computers & Operations Research*, vol. 64, pp. 11–27, 2015.
- [67] G. Ghiani and G. Improta, “An efficient transformation of the generalized vehicle routing problem,” *European Journal of Operational Research*, vol. 122, no. 1, pp. 11–17, 2000.
- [68] R. Baldacci, E. Bartolini, and G. Laporte, “Some applications of the generalized vehicle routing problem,” *Journal of the Operational Research Society*, vol. 61, no. 7, pp. 1072–1077, 2010.
- [69] T. Bektas, G. Erdogan, and S. Røpke, “Formulations and branch-and-cut algorithms for the generalized vehicle routing problem,” *Transportation Science*, vol. 45, no. 3, pp. 299–316, 2011.
- [70] A. A. Kovacs, B. L. Golden, R. F. Hartl, and S. N. Parragh, “The generalized consistent vehicle routing problem,” *Transportation Science*, vol. 49, no. 4, pp. 796–816, 2014.
- [71] H. M. Afsar, C. Prins, and A. C. Santos, “Exact and heuristic algorithms for solving the generalized vehicle routing problem with flexible fleet size,” *International Transactions in Operational Research*, vol. 21, no. 1, pp. 153–175, 2014.
- [72] N.-H. Quttineh, T. Larsson, J. Van den Bergh, and J. Beliën, “A time-indexed generalized vehicle routing model and stabilized column generation for military aircraft mission planning,” in *Optimization, Control, and Applications in the Information Age*, pp. 299–314, Springer, 2015.

- [73] B. Biesinger, B. Hu, and G. Raidl, “An integer l-shaped method for the generalized vehicle routing problem with stochastic demands,” *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 245–252, 2016.
- [74] A. Louati *et al.*, “Modeling municipal solid waste collection: A generalized vehicle routing model with multiple transfer stations, gather sites and inhomogeneous vehicles in time windows,” *Waste Management*, vol. 52, pp. 34–49, 2016.
- [75] L. Moccia, J.-F. Cordeau, and G. Laporte, “An incremental tabu search heuristic for the generalized vehicle routing problem with time windows,” *Journal of the Operational Research Society*, vol. 63, no. 2, pp. 232–244, 2012.
- [76] N. H. Wilson and N. J. Colvin, *Computer control of the Rochester dial-a-ride system*. Massachusetts Institute of Technology, Center for Transportation Studies, 1977.
- [77] H. N. Psaraftis, “A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem,” *Transportation Science*, vol. 14, no. 2, pp. 130–154, 1980.
- [78] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte, “Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem,” *Parallel Computing*, vol. 30, no. 3, pp. 377–387, 2004.
- [79] A. M. Campbell and M. W. Savelsbergh, “Decision support for consumer direct grocery initiatives,” *Transportation Science*, vol. 39, no. 3, pp. 313–327, 2005.
- [80] S. Mitrović-Minić and G. Laporte, “Waiting strategies for the dynamic pickup and delivery problem with time windows,” *Transportation Research Part B: Methodological*, vol. 38, no. 7, pp. 635–655, 2004.
- [81] Z.-L. Chen and H. Xu, “Dynamic column generation for dynamic vehicle routing with time windows,” *Transportation Science*, vol. 40, no. 1, pp. 74–88, 2006.

- [82] S. Ichoua, M. Gendreau, and J.-Y. Potvin, “Exploiting knowledge about future demands for real-time vehicle dispatching,” *Transportation Science*, vol. 40, no. 2, pp. 211–225, 2006.
- [83] M. Mes, M. Van Der Heijden, and A. Van Harten, “Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems,” *European Journal of Operational Research*, vol. 181, no. 1, pp. 59–75, 2007.
- [84] A. Goel and V. Gruhn, “A general vehicle routing problem,” *European Journal of Operational Research*, vol. 191, no. 3, pp. 650–660, 2008.
- [85] B. Fleischmann, S. Gnutzmann, and E. Sandvoß, “Dynamic vehicle routing based on online traffic information,” *Transportation science*, vol. 38, no. 4, pp. 420–433, 2004.
- [86] E. Taniguchi and H. Shimamoto, “Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times,” *Transportation Research Part C: Emerging Technologies*, vol. 12, no. 3, pp. 235–250, 2004.
- [87] H.-K. Chen, C.-F. Hsueh, and M.-S. Chang, “The real-time time-dependent vehicle routing problem,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 42, no. 5, pp. 383–408, 2006.
- [88] J. Barceló, H. Grzybowska, and S. Pardo, “Vehicle routing and scheduling models, simulation and city logistics,” in *Dynamic Fleet Management*, pp. 163–195, Springer, 2007.
- [89] M. Tagmouti, M. Gendreau, and J.-Y. Potvin, “A dynamic capacitated arc routing problem with time-dependent service costs,” *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 1, pp. 20–28, 2011.
- [90] J.-Q. Li, P. B. Mirchandani, and D. Borenstein, “A lagrangian heuristic for the real-time vehicle rescheduling problem,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, no. 3, pp. 419–433, 2009.

- [91] J.-Q. Li, P. B. Mirchandani, and D. Borenstein, “Real-time vehicle rerouting problems with time windows,” *European Journal of Operational Research*, vol. 194, no. 3, pp. 711–727, 2009.
- [92] Q. Mu, Z. Fu, J. Lysgaard, and R. Eglese, “Disruption management of the vehicle routing problem with vehicle breakdown,” *Journal of the Operational Research Society*, vol. 62, no. 4, pp. 742–749, 2011.
- [93] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard, “Parallel tabu search for real-time vehicle routing and dispatching,” *Transportation science*, vol. 33, no. 4, pp. 381–390, 1999.
- [94] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, “Ant colony system for a dynamic vehicle routing problem,” *Journal of Combinatorial Optimization*, vol. 10, no. 4, pp. 327–343, 2005.
- [95] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin, “Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries,” *Transportation Research Part C: Emerging Technologies*, vol. 14, no. 3, pp. 157–174, 2006.
- [96] N. Azi, M. Gendreau, and J.-Y. Potvin, “A dynamic vehicle routing problem with multiple delivery routes,” *Annals of Operations Research*, vol. 199, no. 1, pp. 103–112, 2012.
- [97] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, “A review of dynamic vehicle routing problems,” *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.
- [98] H. N. Psaraftis, M. Wen, and C. A. Kontovas, “Dynamic vehicle routing problems: Three decades and counting,” *Networks*, vol. 67, no. 1, pp. 3–31, 2016.
- [99] T. Bektas, P. P. Repoussis, and C. D. Tarantilis, “Dynamic vehicle routing problems,” *Vehicle Routing: Problems, Methods, and Applications*, vol. 18, p. 299, 2014.

- [100] L. Gouveia, “A result on projection for the vehicle routing problem,” *European Journal of Operational Research*, vol. 85, no. 3, pp. 610–624, 1995.
- [101] A. Letchford and J. Salazar-González, “Projection results for vehicle routing,” *Mathematical Programming*, vol. 105, no. 2-3, pp. 251–274, 2006.
- [102] D. Naddef and G. Rinaldi, *Branch-and-cut algorithms for the capacitated VRP*, ch. in *The Vehicle Routing Problem*, P. Toth and D. Vigo (editors), pp. 53–81. SIAM Monographs on Discrete Mathematics and Applications Philadelphia, PA, 2002.
- [103] A. Atamtürk, “On capacitated network design cut–set polyhedra,” *Mathematical Programming*, vol. 92, no. 3, pp. 425–437, 2002.
- [104] C. Archetti, M. Speranza, and M. Savelsbergh, “An optimization-based heuristic for the split delivery vehicle routing problem,” *Transportation Science*, vol. 42, no. 1, pp. 22–31, 2008.
- [105] T. Ralphs, L. Kopman, W. Pulleyblank, and L. Trotter, “On the capacitated vehicle routing problem,” *Mathematical Programming*, vol. 94, no. 2-3, pp. 343–359, 2003.
- [106] L. Schrage, “Formulation and structure of more complex/realistic routing and scheduling problems,” *Networks*, vol. 11, no. 2, pp. 229–232, 1981.
- [107] D. Sariklis and S. Powell, “A heuristic method for the open vehicle routing problem,” *Journal of the Operational Research Society*, vol. 51, no. 5, pp. 564–573, 2000.
- [108] C. Tarantilis and C. Kiranoudis, “Distribution of fresh meat,” *Journal of Food Engineering*, vol. 51, no. 1, pp. 85–91, 2002.
- [109] J. Brandão, “A tabu search algorithm for the open vehicle routing problem,” *European Journal of Operational Research*, vol. 157, no. 3, pp. 552–564, 2004.
- [110] Z. Fu, R. Eglese, and L. Li, “A new tabu search heuristic for the open vehicle routing problem,” *Journal of the Operational Research Society*, vol. 56, no. 3, pp. 267–274, 2005.

- [111] A. Letchford, J. Lysgaard, and R. Eglese, “A branch-and-cut algorithm for the capacitated open vehicle routing problem,” *Journal of the Operational Research Society*, vol. 58, no. 12, pp. 1642–1651, 2007.
- [112] F. Li, B. Golden, and E. Wasil, “The open vehicle routing problem: Algorithms, large-scale test problems, and computational results,” *Computers & Operations Research*, vol. 34, no. 10, pp. 2918–2930, 2007.
- [113] A. Ceselli, G. Righini, and M. Salani, “A column generation algorithm for a rich vehicle-routing problem,” *Transportation Science*, vol. 43, no. 1, pp. 56–69, 2009.
- [114] Q. Song and L. Liu, “The application of tabu search algorithm on split delivery open vehicle routing problem,” *BioTechnology: An Indian Journal*, vol. 8, no. 8, pp. 1088–1094, 2013.
- [115] P. Fouilhoux, O. E. Karasan, A. R. Mahjoub, O. Özkök, and H. Yaman, “Survivability in hierarchical telecommunications networks,” *Networks*, vol. 59, no. 1, pp. 37–58, 2012.
- [116] M. Stoer and F. Wagner, “A simple min-cut algorithm,” *Journal of the Association for Computing Machinery*, vol. 44, no. 4, pp. 585–591, 1997.
- [117] J. Edmonds and R. M. Karp, “Theoretical improvements in algorithmic efficiency for network flow problems,” *Journal of the Association for Computing Machinery*, vol. 19, no. 2, pp. 248–264, 1972.
- [118] M. Farber, “Consumers are now doing most of their shopping online.” <http://fortune.com/2016/06/08/online-shopping-increases/>, 2016. Accessed: 2016-07-24.
- [119] A. Garcia, “Amazon prime day shattered global sales records.” <http://money.cnn.com/2015/07/15/news/amazon-walmart-sales/>, 2015. Accessed: 2015-09-22.

- [120] K. Gustafson, “Amazon just had its biggest sales day ever.” <http://www.cnbc.com/2016/07/13/amazon-prime-day-is-biggest-day-for-online-retailer-ever.html>, 2016. Accessed: 2016-07-24.
- [121] M. B. Solomon, “Amazon’s first-quarter shipping costs hit \$3.27 billion, 42-percent jump from 2015.” <http://www.dcvelocity.com/articles/20160428-amazons-first-quarter-shipping-costs-hit-327-billion-42-percent-jump-from-2015>, 2016. Accessed: 2016-08-08.
- [122] B. Popken, “Amazon tests delivery to your car trunk.” <http://www.nbcnews.com/business/autos/amazon-testing-delivery-your-car-trunk-n346886>, 2015. Accessed: 2016-08-07.
- [123] M. Geuss, “Amazon, audi and dhl want to turn a car trunk into a delivery locker.” <http://arstechnica.com/business/2015/04/amazon-audi-and-dhl-want-to-turn-a-car-trunk-into-a-delivery-locker>, 2015. Accessed: 2016-08-07.
- [124] Audi, “Audi, dhl and amazon deliver convenience.” <https://www.audiusa.com/newsroom/news/press-releases/2015/04/audi-dhl-and-amazon-deliver-convenience>, 2015. Accessed: 2016-07-24.
- [125] Volvo, “Volvo in-car delivery.” <https://incardelivery.volvocars.com>, 2015.
- [126] Volvo, “Volvo’s solution for the package theft epidemic: Your car’s trunk.” <http://fortune.com/2016/05/10/volvo-urb-it-delivery>, 2016.
- [127] E. Behrmann and R. Weiss, “Soon your smart car will also be an amazon locker.” <http://www.bloomberg.com/news/articles/2016-07-25/smart-city-cars-to-become-delivery-stations-in-dhl-german-test>, 2016. Accessed: 2016-08-07.
- [128] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, “Branch-and-price: Column generation for solving huge integer programs,” *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998.

- [129] C. H. Christiansen and J. Lysgaard, “A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands,” *Operations Research Letters*, vol. 35, no. 6, pp. 773–781, 2007.
- [130] M. Dell’Amico, G. Righini, and M. Salani, “A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection,” *Transportation Science*, vol. 40, no. 2, pp. 235–247, 2006.
- [131] M. Salani and I. Vacca, “Branch and price for the vehicle routing problem with discrete split deliveries and time windows,” *European Journal of Operational Research*, vol. 213, no. 3, pp. 470–477, 2011.
- [132] N. Boland, J. Dethridge, and I. Dumitrescu, “Accelerated label setting algorithms for the elementary resource constrained shortest path problem,” *Operations Research Letters*, vol. 34, no. 1, pp. 58–68, 2006.
- [133] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, “An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems,” *Networks*, vol. 44, no. 3, pp. 216–229, 2004.
- [134] D. Feillet, “A tutorial on column generation and branch-and-price for vehicle routing problems,” *4OR*, vol. 8, no. 4, pp. 407–424, 2010.
- [135] A. Regan, H. Mahmassani, and P. Jaillet, “Evaluation of dynamic fleet management systems: Simulation framework,” *Transportation Research Record: Journal of the Transportation Research Board*, no. 1645, pp. 176–184, 1998.
- [136] S. Ichoua, M. Gendreau, and J.-Y. Potvin, “Diversion issues in real-time vehicle dispatching,” *Transportation Science*, vol. 34, no. 4, pp. 426–438, 2000.
- [137] A. Attanasio, J. Bregman, G. Ghiani, and E. Manni, “Real-time fleet management at ecourier ltd,” in *Dynamic Fleet Management*, pp. 219–238, Springer, 2007.
- [138] J. v. d. Klundert and L. Wormer, “Asap: the after-salesman problem,” *Manufacturing & Service Operations Management*, vol. 12, no. 4, pp. 627–641, 2010.

- [139] J. Respen, N. Zufferey, and J.-Y. Potvin, *Impact of online tracking on a vehicle routing problem with dynamic travel times*. CIRRELT, 2014.
- [140] F. Ferrucci and S. Bock, “A general approach for controlling vehicle en-route diversions in dynamic vehicle routing problems,” *Transportation Research Part B: Methodological*, vol. 77, pp. 76–87, 2015.