FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Algorithmic Approaches for Optimization Problems in Bike Sharing and Security Control

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Christian Kloimüllner
Matrikelnummer 0628060

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

Diese Dissertation haben begutachtet:

Kenneth Sörensen	Nysret Musliu	Günther R. Raidl

Wien, 31. Jänner 2019

Christian Kloimüllner

Algorithmic Approaches for Optimization Problems in Bike Sharing and Security Control

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Christian Kloimüllner
Registration Number 0628060

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

The dissertation has been reviewed by:

Kenneth Sörensen	Nysret Musliu	Günther R. Raidl

Vienna, 31st January, 2019

Christian Kloimüllner

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Christian Kloimüllner
Hofwiese 13, 3204 Kirchberg/Pielach

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. Jänner 2019

Christian Kloimüllner

Danksagung

Ich widme diese Arbeit meiner lieben Mutter Gerda, welche kurz vor dem Einreichen der Arbeit unerwartet und plötzlich aus dem Leben geschieden ist. Ich habe sie sehr geliebt und sie hat mir auch sehr viel Liebe geschenkt und sie wird immer einen Platz in meinem Herzen haben. Die Liebe hat viel zu meinem Leben und auch dieser Arbeit beigetragen und ohne meine Mutter würde ich jetzt nicht derjenige sein, der ich jetzt bin. Danke Mama und Ruhe in Frieden.

Mein größter Dank gebührt dem Betreuer meiner Dissertation Prof. Dr. Günther R. Raidl. Ich habe von ihm im Zuge meiner Dissertation viel Neues gelernt, nicht nur im Bereich der Algorithmen, sondern ebenfalls bezüglich wissenschaftlichen Arbeitens. Ich war stets sehr beeindruckt von der Genauigkeit und Präzision von Prof. Dr. Raidl was mir beim Publizieren sehr zu Gute kam und in den Reviews der Gutachter offensichtlich wurde. Ich schätze Prof. Dr. Raidl nicht nur als exzellenten Wissenschaftler, sondern auch als großartigen Menschen und Freund, der mir auch in Zeiten, in denen meine Forschung und somit auch die Dissertation, stockte, Mut machte und mich motivierte mein Ziel weiter zu verfolgen.

Ich danke auch meinen Kollegen am Institut für die großartige Zusammenarbeit, die Diskussionen rund um unsere Forschungsthemen aber auch für das wunderbare Klima innerhalb der Gruppe wodurch man immer mit größter Freude und Elan an die Arbeit gehen konnte.

Für die ständige Unterstützung in allem was ich bis jetzt gemacht und erreicht habe danke ich meinen Großeltern Franz und Elisabeth. Es ist einfach unbezahlbar welch großen Rückhalt ich in Ihnen erfahren durfte. Ich bedanke mich bei ihnen auch dafür, dass sie mir bei jedem noch so kleinen Problem bei Seite stehen und immer ein offenes Ohr haben, egal um welches Problem es sich handelt. Es sind nicht nur meine Großeltern, sondern auch meine besten Freunde und die Menschen, die mir im Leben am nächsten stehen.

Meiner Lebensgefährtin, Andrea, danke ich dafür, dass sie immer hinter dieser Arbeit gestanden ist auch wenn die Zeit für uns währenddessen nicht immer leicht war. Sie hat mir Trost gespendet, wenn es einmal nicht so gut lief und mich immer aufgemuntert. Ich bin ihr dankbar, dass sie mir neben meiner Leidenschaft, den Algorithmen, auch vieles andere Schöne im Leben gezeigt hat.

Für die finanzielle Unterstützung bedanke ich mich bei der *Österreichischen Forschungsförderungsgesellschaft* (FFG), welche die jeweiligen Projekte durch Verträge 831740, 849028 und 856215 unterstützt hat. Ebenfalls bedanken möchte ich für die Finanzierung seitens der TU Wien, die die finanziellen Mittel für das Doktoratskolleg *Environmental Informatics* bereitgestellt hat.

Acknowledgements

I dedicate this thesis to my beloved mother Gerda who showered me with her love and blessings. Her unconditional love will always stay in my heart forever. Unfortunately, she unexpectedly past away before I submitted this thesis. I would not reach this stage in my life if it was not for her. Thank you mom and may your soul rest in peace.

I would like to express my deepest gratitude to my supervisor Prof. Dr. Günther R. Raidl. I have learned a lot from him and besides algorithms, he has profoundly taught me scientific work. Prof. Raidl has motivated me and encouraged me developing my thoughts on this topic and completing this thesis. I am amazed with his meticulous work and his scientific knowledge in the field. He is not only a great scientist but also a good friend and a wonderful person.

I greatly appreciate the collaboration with my colleagues at the institute. The discussions on our work have been very fruitful, and besides work we had a great time together and a wonderful working atmosphere.

Furthermore, I would like to express my thankfulness to my grandparents Franz and Elisabeth who supported me throughout my life with their sacrificial love. I could always count on them as they tried making my life as easy as possible while I could fully concentrate on my research. I will never forget all that they did for me.

Many thanks also to my girlfriend, Andrea, who always supported me during this PhD thesis, although sometimes it was difficult for our relationship. She supported me and pushed me forward to work while I was struggling with my research. I am grateful to her for showing me all the beautiful things life has to offer beside my passion and algorithms.

Lastly, I am grateful for the financial support provided by the Austrian Research Promotion Agency (FFG) under contracts 831740, 849028 and 856215. I also appreciate the funding of TU Wien for the doctoral college *Environmental Informatics.*

Kurzfassung

Diese Arbeit behandelt drei kombinatorische Optimierungsprobleme, wobei zwei davon ihren Ursprung in öffentlichen Fahrradverleihsystemen haben und das dritte Problem im Kontext von Sicherheitskontrollen auftritt. Öffentliche Fahrradverleihsysteme erfreuen sich stetig wachsender Beliebtheit, vor allem in größeren Städten, da sie zur allgemeinen Gesundheit beitragen indem die Einwohner der Stadt zu mehr Sport angeregt werden, sie verbessern die CO2-Bilanz und unterstützen auch den öffentlichen Transport indem sie einen Lösungsansatz für das sogenannte Last-Mile Problem darstellen. Öffentliche Fahrradverleihsysteme bestehen zumeist aus mehreren festen Stationen, an denen die Fahrräder mittels eines Docking-Mechanismus abgestellt sind, was der Diebstahlsicherung dient. Fahrräder können an einer Station entlehnt werden und an einer anderen wieder zurückgegeben werden woraus folgt, dass manche Stationen leer und andere Stationen hingegen voll werden können. Dadurch ist es notwendig das System zu rebalancieren und das zu Grunde liegende Optimierungsproblem wird als Balancing-Bike Sharing System Problem bezeichnet. Wenn ein neues System geplant werden oder ein bestehendes erweitert werden soll, dann entsteht ein weiteres Optimierungsproblem, nämlich dass unter gewissen Budgetbeschränkungen und weiteren Bedingungen/Einschränkungen, neue Stationen geplant werden sollen, sodass der Nutzen für die Benutzer des Systems maximiert wird. Dieses Problem wird als Bike-Sharing Station Planning Problem bezeichnet. Ein weiteres Problem, das in der Dissertation behandelt wird, tritt im Kontext von Sicherheitskontrollen auf. Es kommt immer wieder zu Diebstählen und Vandalismus an Gebäuden und deren Einrichtung, sodass es notwendig ist diese zu überwachen, um eben diese Diebstähle und den Vandalismus so gut wie möglich einzuschränken. Da es aufgrund beschränkter Ressourcen nicht möglich ist Gebäude konstant zu überwachen, betrachten wir die Minimierung der Anzahl von Routen die notwendig sind, um eine gewisse Menge an Gebäuden zu überwachen. Wir bezeichnen dieses Problem als Districting and Routing Problem for Security Control. Das Problem ist verwandt mit dem Balancing Bike-Sharing System Problem, weil es sich bei beiden Problemen um eine Variante des bekannten Vehicle Routing Problems handelt.

Beim Balancing Bike-Sharing System Problem betrachten wir zuerst die statische Variante, bei der keine Benutzerinteraktionen während des Rebalancierens in Betracht gezogen werden. Zuerst stellen wir hierfür eine PILOT sowie eine Greedy Konstruktionsheuristik vor, welche mit zusätzlicher, geschickter lokaler Suche sinnvolle, aber normalerweise suboptimale Lösungen in kurzen Laufzeiten liefern. In weiterer Folge werden eine variable

Nachbarschaftssuche sowie ein GRASP Ansatz vorgeschlagen, welche in der Lage sind heuristische Lösungen für Instanzen bis zu 700 Stationen zu berechnen. Zusätzlich ist ein wichtiger Teil der Arbeit das effiziente Berechnen der Ladeinstruktionen für die Stationen. Zu diesem Zweck vergleichen wir einen integrierten Greedy-Ansatz sowie zwei Ansätze basierend auf Maximum-Flow Formulierungen und einen Ansatz basierend auf linearer Programmierung. Als nächstes betrachten wir die dynamische Variante des Problems, bei der wir auch die Benutzerinteraktionen während des Rebalancierens in Betracht ziehen. Die wenigen Arbeiten, die es zum dynamischen Fall des Problems gibt, basieren fast ausschließlich auf Zeitdiskretisierung. Eine solche erhöht aber die Laufzeiten der Algorithmen und ist ebenso fehleranfälliger, je nachdem in welcher Größe die Diskretisierung vorgenommen wird. Daher schlagen wir einen neuen, innovativen Ansatz ein, bei dem keine Diskretisierung nötig ist, sondern basierend auf monoton steigenden und monoton fallenden Segmenten die Benutzernachfrage modelliert wird. Dadurch werden schnellere Laufzeiten der Algorithmen erreicht und auch die Genauigkeit der Berechnungen steigt gegenüber der Variante mit Zeitdiskretisierung. Grundsätzlich sind alle unsere bisherigen Ansätze darauf ausgelegt eine beliebige Anzahl von Fahrrädern bei den Ladeinstruktionen zu betrachten. Im Gegensatz dazu werden in der Praxis meist nur volle Wagenladungen verführt, da in der Regel mehr Rebalancierungsarbeit vorhanden ist als erledigt werden kann und mit vollen Wagenladungen intuitiv die größte Effizienz erreicht werden kann. Daher stellen wir eine neue vereinfachte Problemformulierung vor, welche nur ganze Wagenladungen bei den Ladeinstruktionen berücksichtigt. Dadurch ergeben sich neue und effektive Lösungswege und -methoden. Um diese Problemstellung zu lösen, stellen wir einen Ansatz vor, welcher das Problem in ein Zuweisungsproblem sowie ein Routing-Problem zerlegt und anschließend mit logikbasierter Benders-Decomposition gelöst wird. Ein Highlight des Ansatzes ist die Approximation der Routingkosten im Zuweisungsproblem durch eine 0-Arboreszenz, sodass für das Routing-Problem sinnvolle Zuweisungen generiert werden, welche die Berechnungszeit des Algorithmus erheblich beschleunigt.

Beim Planen von Fahrradverleihsystemen für Großstädte ist es zumeist notwendig mehrere tausend potentielle Standorte für neue Stationen in Betracht zu ziehen, wodurch traditionelle Optimierungsverfahren oft in ihre Schranken gewiesen werden. Unser Ziel ist es dabei nicht einen vollautomatisierten Planungsalgorithmus zu entwickeln, sondern ein halbautomatisches Planungstool, welches Vorschläge für die letztendlich manuelle Planung berechnet. Um Instanzen dieser Größenordnung in den Griff zu bekommen, haben wir einen neuen und innovativen Lösungsansatz entworfen, welcher zuerst ein hierarchisches Clustering aufgrund der originalen Inputdaten berechnet und danach vernachlässigbare, kleine Nachfragewerte auf den unteren Ebenen des Clusterings nach oben in den Cluster-Baum aggregiert, wo diese Werte schlussendlich nicht mehr vernachlässigbar sind. Auf Basis dieses Clusterings und angepassten Inputs wird ein Ansatz basierend auf dem Multilevel-Refinement Paradigma vorgestellt, welcher effizient mit dem bereits erfolgten hierarchischen Clustering angewendet werden kann. Dieser innovative Ansatz erscheint nicht nur in dieser Spezialanwendung vielversprechend, sondern auch für andere Optimierungsprobleme welche große Inputdaten in Form einer Matrix mit

Nachfragewerten gegeben haben.

Bei Betrachtung des Districting and Routing Problem for Security Control ist es wünschenswert alle zu betrachtenden Objekte mit einer minimalen Anzahl an Routen zu überwachen. Zuerst wird hierfür eine Greedy-Konstruktionsheuristik vorgestellt, welche mit mehreren, unterschiedlichen Evaluationskritierien entwickelt und getestet wurde. Basierend auf den Ausgangslösungen dieser Konstruktionsheuristik haben wir einen iterativen Ansatz entwickelt, der darauf basiert, bestehende Routen zu zerstören und die übrig gebliebenen Besuche von Objekten noch bestehenden Routen zuweist. Diese freistehenden Besuche werden in einem sogenannten Ejection-Pool gehalten, um die Möglichkeit/Wahrscheinlichkeit diese Besuche unter den bestehenden Routen einzufügen, zu maximieren. Eine weitere Problemvariante ist die Betrachtung von weichen Zeitfenstern für die Besuche von Objekten. In der Praxis stellt sich immer wieder heraus, dass kleine Zeitfensterverletzungen vertretbar sind, wenn dadurch die Lösungsqualität substanziell verbessert werden kann. Dabei entsteht eine neue Problemvariante, bei der optimale Ankunftszeiten für eine gegebene Route beziehungsweise Besuchsreihenfolge gefunden werden müssen, sodass die gesamte Dauer der Route minimiert wird. Dazu wird ein Modell basierend auf linearer Programmierung und eine schnellere hybride Heuristik basierend auf dynamischer Programmierung, welche in den meisten Fällen bewiesen optimale Lösungen berechnet, vorgestellt. Mit dieser Lösungsmethode des Subproblems ist es möglich eine große Nachbarschaftssuche zu implementieren, in der die effiziente Lösung des Subproblems ein integraler Bestandteil ist. Mit dieser Metaheuristik ist es möglich, Lösungen von ausgezeichneter Qualität in moderater Laufzeit zu berechnen.

Abstract

This thesis deals with three combinatorial optimization problems where two of them arise in the context of public bike-sharing systems and the other one in the domain of security control. Public bike-sharing systems are emerging in large cities worldwide and contribute to public health, reduce CO2 emissions and complement public transport. Most public bike-sharing systems have rental stations where bikes are docked into. These stations may get full or empty making it necessary to redistribute the bikes among the docking stations in the system. This is referred to as the balancing bike-sharing system problem. When a bike-sharing system is set up or extended, the new system or the extensions have to be planned such that under certain budget restrictions and other constraints, the prospective user benefit is maximized. This problem is denoted as the bike-sharing station planning problem. Theft and vandalism make it necessary that certain buildings are surveilled by security guards multiple times a day within particular time windows. We consider the optimization problem of minimizing the number of security routes needed to observe all buildings in consideration. This problem is called the districting and routing problem for security control and relates to the balancing bike sharing rebalancing problem as it is also a particular kind of vehicle routing problem.

For the balancing bike-sharing systems problem, we first analyze the static variant of the problem where no user interaction takes place during rebalancing. In this scenario, we propose a greedy and a PILOT construction heuristic combined with additional local-search improvements for obtaining fast ad-hoc solutions in practice which, however, are usually suboptimal. A further developed variable neighborhood search and greedy randomized adaptive search procedure yield heuristic solutions on large instances up to 700 stations. Additionally, to the routing part of the problem we propose efficient algorithms to compute loading instructions for the station visits. We compare an integrated greedy approach, two approaches based on maximum-flow algorithms and a linear programming based method. Furthermore, we study the dynamic variant of the problem where user interaction during the rebalancing process is considered. Previous works model the problem with time discretization, which we aim to avoid, since this slows down the algorithm and also introduces additional computation errors depending on the unit of the discretization. Thus, we model the problem by considering monotonically increasing and decreasing segments of a user demand function which enables substantially faster computations and introduces less errors than previous work. While our approaches are in general able to consider the transportation of arbitrary numbers of bikes, in practice

frequently only full-vehicle loads are considered. We therefore also study the simplified problem formulation with always full-vehicle loads, which is practically, highly relevant and allows more effective solving strategies. Based on this problem formulation we propose a new and efficient algorithm that splits the problem into an assignment part and a routing part which follows a logic-based Benders decomposition approach. The costs of the routing part in the subproblem are approximated with a 0-arborescence in the assignment part of the algorithm, yielding a strong relaxation so that already meaningful assignments are obtained for the routing part.

When planning bike-sharing station locations for large cities, thousands of potential station locations have to be considered, and cannot be handled anymore by traditional (meta)heuristics. The goal is not to create a fully automatized planning but rather to design a decision-support tool which helps the planner of such systems by proposing locations for new station candidates. To this end, we propose a novel solution approach which first computes a hierarchical clustering on the original input data and then aggregates the negligible demands on the lower levels of the hierarchical clustering to an upper level where the aggregated demands play a substantial role for the solution. Based on this meaningfully generated hierarchical clustering, we propose an algorithm based on the multilevel refinement paradigm, which utilizes the hierarchical clustering fo the input data again. This novel solution approach also appears promising for other (real-world) optimization problems with large input data.

For the districting and routing problem for security control, it is desired that all buildings can be observed with a minimum number of routes. We propose a greedy-based districting construction heuristic based on various greedy evaluation criteria. Upon this construction heuristic, we build a route minimization algorithm that iteratively destroys routes and intelligently maintains an ejection pool from where visits of buildings are (re)inserted into other routes than the destroyed ones. Moreover, we extend the problem to a variant with soft time windows as small time window violations are acceptable in practice but may improve solution quality significantly. A new problem arises where optimal arrival times have to be found for each visit of a route such that the makespan of the route is minimized. We therefore propose a linear-programming model and a superior hybrid heuristic based on dynamic programming which is able to yield in most cases proven optimal solutions. With this efficient solution technique for the subproblem we are able to embed the whole mechanism within a large neighborhood search providing high-quality solutions to the districting and routing problem for security control with soft time windows.

Contents

CHAPTER 1

Introduction

Public bike sharing systems (PBS) are constructed in cities all over the world. Most of the time, they consist of self-service rental stations distributed over a city where users can rent and return bikes. In modern systems, each station has a computer terminal for automatic rental and return. For security reasons and to prevent theft and vandalism of the bikes, every station consists of docks where the bikes are docked into. Bikes can only be returned if at least a single dock is free and obviously, they can only be rent, if at least one dock is occupied by a bike. Obviously, both scenarios, i.e., full as well as empty stations hinder the use of the system and easily annoy potential customers. Thus, these cases have to be avoided as far as possible to increase customer satisfaction. In contrast to these station-based systems, there exist also *free floating bike sharing systems* [19, 108] but as station-based systems are clearly more widespread we only concentrate on those in this work. PBSs should also not be confused with more classical bike-rental systems, which are designed for long-term use, higher prices, and to typically return a bike from where it was rent, whereas PBSs are designed for short-term use, cheaper prices, and one-way trips. Sometimes, PBSs are even without costs for the customer, as in the case of *Citybike Wien*[1] in Vienna. In order to make sure that the bikes are constantly available to the customers of these systems, for longer use the price drastically rises. The economic advantage of PBSs is manifold. They help at reducing CO2 emissions, encourage people to do more sports, complement public transport and could also solve the last mile problem in the case of public transport. Lastly, another advantage of these shared mobility systems is that they use public spaces more efficiently as people need not own their individual private vehicles.

For the success of a PBS, the station locations have to be planned such that they are distributed among a city to fulfill potential customer demand as much as possible. When planning a new PBS or extending an existing one, various aspects have to be considered.

[1]https://www.citybikewien.at/

Usually, customer demand is higher in areas of large housing complexes, office buildings, and stations of public transport. It is also important to consider the flow of bikes. Some stations tend to get full whereas others tend to get empty which also makes it necessary to rebalance these systems such that at each station in each time point free bikes as well as docks are available. The expected rebalancing costs should be minimized, as they represent non-negligible operating costs; this aspect should be already considered in the planning phase of a PBS such that there is a minimum requirement of rebalancing when maintaining the system by the operator. To use the limited resources, i.e., workforce and money, as efficiently as possible, difficult combinatorial optimization problems arise when planning and rebalancing a PBS which require elaborate optimization techniques in order to do these tasks as effectively as possible.

When operating a PBS, it is necessary to perform rebalancing activities such that the system is in a certain balance during the whole period of operation. Since it is crucial for the success of such a system that customers have constantly free bikes and docks available at their disposal at nearly all stations of the system, bikes have to be continuously moved from stations with an excess of bikes to stations with a lack of bikes. Typically, vehicles with trailers are used to accomplish this task. The underlying combinatorial optimization problem is to plan rebalancing routes with corresponding loading and unloading instructions at every stop such that the system is in an optimal condition after the rebalancing process. For the static case of this problem, i.e., there is no user interaction during the rebalancing, which can be useful for, e.g., overnight rebalancing, we propose efficient heuristics and metaheuristics for solving the problem. A fast greedy construction heuristic as well as a PILOT construction heuristic are designed for finding meaningful results in short runtimes. Additional local-search components are suggested for improving results from the construction heuristics. For finding high-quality solutions in reasonable runtimes we propose an efficient variable neighborhood search exploiting cleverly chosen neighborhoods as well as a greedy randomized adaptive search procedure (GRASP) which is based on the efficient greedy construction heuristic. We provide rigorous test results for these algorithms on a large benchmark suite based on real-world data occurring at Citybike Wien. The findings have been published in:

> *M. Rainer-Harbach, P. Papazek, B. Hu, G. R. Raidl, and C. Kloimüllner. PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems.* Journal of Global Optimization*, 63(3):597–629, 2015*

Moreover, we also considered the dynamic variant of the problem, when user interaction during the rebalancing is considered. For this case, we adapted methods and methodologies from the static case and propose a smart way of calculating the dynamic aspects by splitting the user demand functions into monotonically increasing and decreasing segments. The corresponding paper was nominated for best paper candidate at the *14th European Conference on Evolutionary Computation in Combinatorial Optimisation*:

> *C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: An approach for the dynamic case. In C. Blum and G. Ochoa, editors,*

> Evolutionary Computation in Combinatorial Optimisation*, volume 8600 of* Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2014*

To be also able to solve practically relevant instances to optimality we considered a novel and simplified problem formulation that considers only full-vehicle loads. For this, we propose a cluster-first route-second heuristic that has been published in:

> *C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl. A cluster-first route-second approach for balancing bicycle sharing systems. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors,* Computer Aided Systems Theory – EUROCAST 2015*, volume 9520 of* Lecture Notes in Computer Science*, pages 439–446. Springer International Publishing, 2015*

This work is subsequently extended to an exact logic-based Benders decomposition approach for finding proven optimal solutions. The algorithmic framework and rigorous tests are published in:

> *C. Kloimüllner and G. R. Raidl. Full-load route planning for balancing bike sharing systems by logic-based Benders decomposition.* Networks*, 69(3):270–289, 2017*

We further consider the bike sharing station planning problem (BSSPP). Given a city and prospective customer demands for a city and a particular maximum budget, the goal is to plan station locations and a corresponding number of bike docks such that a given customer demand is fulfilled as far as possible. As PBSs are mostly implemented in large cities, practical problem instances are large, making it necessary to think about novel optimization techniques to use limited resources such as CPU and memory as efficient as possible, as otherwise it is not possible to solve those problem instances arising in practical scenarios. In this thesis a novel optimization technique is developed which, basically, applies hierarchical clustering to the input data such that a traditional full demand matrix is not needed and algorithms can operate on a much sparser hierarchically clustered input data. An algorithm based on the multilevel refinement paradigm is developed for utilizing this hierarchically clustered input data. This approach is introduced in the following work:

> *C. Kloimüllner and G. R. Raidl. Hierarchical clustering and multilevel refinement for the bike-sharing station planning problem. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors,* Learning and Intelligent Optimization*, volume 10556 of* Lecture Notes in Computer Science*, pages 150–165. Springer International Publishing, 2017*

The approach is then extended to a problem variant more relevant for the practical scenario in Vienna and tested on instances based on real-world data of Vienna. The approach and corresponding results have been submitted to the *13th Learning and Intelligent Optimization Conference*:

> *C. Kloimüllner and G. R. Raidl. A novel approach for solving large-scale instances in the bike sharing station planning problem. Technical report, Institute of Logic and Computation, TU Wien, 2019. submitted to 13th Learning and Intelligent Optimization Conference*

The outcome of the project is a *semi-automated planning tool* for PBSs. A full description of this tool, including the requirement analysis, the input generation, demand modeling, algorithmic approach and the visualization as well as the planning frontend are published in the Proceedings of 7th Transport Research Arena (TRA-2018):

> *M. Straub, C. Rudloff, A. Graser, C. Kloimüllner, G. R. Raidl, M. Pajones, and F. Beyer. Semi-automated location planning for urban bike-sharing systems. In* Proceedings of the 7th Transport Research Arena (TRA 2018)*, pages 1–10, Vienna, Austria, 2018*

The proposed rebalancing problem occurring in PBSs is basically a special kind of *vehicle routing problem* (VRP). In VRPs, routes have to be planned for a vehicle fleet under various conditions and constraints. The VRP is a long studied combinatorial optimization problem and occurs in many different variants in real-world problems. Another example of such a VRP-related problem is the *districting and routing problem for security control* (DRPSC).

Due to theft and vandalism, certain buildings have to be constantly surveilled. As a result of limited economic resources, such as money and security staff, it is not possible to constantly observe buildings, but these buildings have to be visited one or multiple times a day such that theft and vandalism is minimized. This requires elaborate optimization techniques to be developed to plan efficient routes for performing a given set of visits to fulfill the custodial duty. Here, the underlying combinatorial optimization problem is to perform all visits of all buildings by minimizing the number of routes. Thus, a districting construction heuristic based on various greedy evaluation criteria as well as a novel district elimination algorithm are proposed. The problem formulation is introduced and results are published in:

> *M. Prischink, C. Kloimüllner, B. Biesinger, and G. R. Raidl. Districting and routing for security control. In M. J. Blesa, C. Blum, A. Cangelosi, V. Cutello, A. D. Nuovo, M. Pavone, and E.-G. Talbi, editors,* Hybrid Metaheuristics*, volume 9668 of* Lecture Notes in Computer Science*, pages 87–103. Springer International Publishing, 2016*

Moreover, as in practice small time window violations are negligible but may substantially improve solution quality we also study a variant of the problem when considering soft time windows instead of hard time windows. For solving a subproblem, namely to determine optimal arrival times at objects, given a particular visit order, and minimizing the makespan of the visit order, we propose a linear-programming model as well as a faster hybrid heuristic based on dynamic programming which is used within a large neighborhood search. These novel, interesting algorithmic frameworks and results for the DRPSC with soft time windows have been published in:

> *B.-M. Kim, C. Kloimüllner, and G. R. Raidl. Efficient consideration of soft time windows in a large neighborhood search for the districting and routing problem for security control. In B. Hu and M. López-Ibáñez, editors,* Evolutionary Computation

in Combinatorial Optimization, *volume 10197 of* Lecture Notes in Computer Science, *pages 91–107. Springer International Publishing, 2017*

1.1 Overview of the Thesis

Chapter 2 introduces the methods and methodologies used within this thesis to approach and solve the three selected COPs.

Chapter 3 shows in detail the work done in the area of balancing bike-sharing systems. First of all, the static case is described in detail and rigorous computational experiments are shown for various heuristics, metaheuristics as well as a mixed integer linear programming approaches [125]. Also comparisons between different methods for deriving optimal loading instructions are shown. Furthermore, the dynamic problem variant is introduced and an efficient solution method based on monotonically increasing and decreasing segments is described and results on instances that have been derived from a real-world scenario are shown [86]. In the remainder of this chapter, a simplified problem formulation is introduced and proven optimal solutions derived by logic-based Benders decomposition and a variant thereof, branch-and-check, are shown [87, 83].

In Chapter 4, the BSSPP is explained in detail and it is described how hierarchically clustered input data can be derived from the original input. An algorithm is shown that aggregates negligible demand on the lower levels of the clustering tree to non-negligible demand on the upper levels of the tree, essentially reducing instance size. Based on this hierarchical clustering, an optimization scheme based on multilevel refinement is introduced [84, 85]. The introduced technique is well applicable to other optimization problems which also need to manage large instance sizes. Results are shown on randomly generated instances as well as instances derived from the practical scenario in Vienna.

The DRPSC is introduced in detail in Chapter 5. First, a districting construction heuristic and a sophisticated district elimination algorithm is introduced and results are shown for instances derived from real-world data [117]. The problem is then extended to a variant with soft time windows, where we show two efficient methods for solving the problem of assigning optimal arrival times to an ordered sequence of visits of buildings. A linear programming based approach is shown and a practically superior hybrid heuristic based on dynamic programming is proposed [81]. This procedure is embedded within a large neighborhood search.

Finally, we draw conclusions on the work described in this thesis and present possible future work in Chapter 6.

CHAPTER 2

Methodology

This chapter talks about *combinatorial optimization problems* (COPs) and comes up with basics in solution methods for solving these problems which are relevant in the following chapters of this thesis. In general, COPs can be solved by an exact algorithm providing optimal solutions, or by a (meta)heuristic yielding good solutions but not necessarily optimal solutions. These two types of solution methods are especially important for the work in this thesis, but there exist also *approximation algorithms* which aim at finding provable guarantees for the quality of solutions produced by the approximation algorithm with respect to the optimal solution. Optimal solutions of exact methods come together with optimality proofs showing that the obtained solution is indeed optimal, but if the problem is too complex to be solved optimally in practical time, exact methods sometimes yield heuristic solutions together with upper and lower bounds to an optimal objective value for a given problem instance of a COP. (Meta)heuristic algorithms do not usually prove quality guarantees but are nevertheless highly relevant in practice, as they frequently yield best solutions in practice [58].

2.1 Combinatorial Optimization Problems

The problems which are described in this thesis are all *combinatorial optimization problems*, therefore the term is introduced in this section. According to Wolsey [160], a combinatorial optimization problem consists of a finite set $N = \{1, \ldots, n\}$ and weights $c_j \, \forall j \in N$. Moreover, a set F of feasible subsets of N is given. A *combinatorial optimization problem* is then defined as finding the minimum weight feasible subset:

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j : S \in F \right\} \tag{2.1}$$

The value of the optimal solution to the COP is then the sum of all weights of the minimum weight feasible subset. COPs are not only limited to minimization problems, but can also be maximization problems.

Most COPs are said to be *$\mathcal{NP}$-hard.* Formally, Schrijver [138] describes the complexity class $\mathcal{NP}$ as follows: "*a decision problem* $\alpha \subseteq \sum^*$ belongs to $\mathcal{NP}$ if there exists a polynomially solvable decision problem $\alpha' \subseteq \sum^* \times \sum^*$ and a polynomial ϕ such that for each z in $\sum^*$:"

$$z \in \alpha \Leftrightarrow \exists y \in \sum\nolimits^* : (z, y) \in \alpha' \text{ and } \mathrm{size}(y) \leq \phi(\mathrm{size}(z)) \tag{2.2}$$

Unless $\mathcal{P} \neq \mathcal{NP}$ there exists no polynomial-time algorithm that can solve all problems lying in $\mathcal{NP}$, in particular the subset of *$\mathcal{NP}$-hard* problems. One widely used methodology for solving COPs is *mixed integer linear programming* (MIP) whereas (meta)heuristic approaches to COPs often yield solution with good quality but often lack theoretic information about optimality gaps. There also exist modern approaches which combine exact methods and (meta)heuristics. Those are denoted as hybrid (meta)heuristics [15].

2.2 Exact methods

We describe methods which are able to yield proven optimal solutions to COPs. One of the most prominent method is *mixed-inter linear programming* (MIP). Linear programming (LP) is often used to solve the relaxed problem and provide a straightforward dual bound to the COP. Other exact methods are also constraint programming [130] and dynamic programming as well as SAT solving [11]. As practical relevant problems often consist of a huge number of variables and constraints, it is often not enough to solve the full MIP model but more sophisticated methods based on decomposition techniques are needed such as the cutting plane method, (logic-based) Benders decomposition [26, 118] and column generation [39]. These decomposition techniques take advantage of solving the problem with only considering a subset of the variables and/or constraints of the original problem, and adding constraints and variables until optimality can be proved. Those methods are explained in detail in the following sections. Most of the following part is based on the books written by Bertsimas and Tsitsiklis [10] as well as Wolsey [160].

2.2.1 Branch-and-Bound

Each COP can be solved exactly by enumerating the whole search space and in the end taking the assignment of values to variables which yield the best objective value according to the evaluation function. Obviously, such an exhaustive search is also for small problem instances a very time-consuming task and even not possible for larger instances. Thus, smarter methods need to be developed to obtain proven optimal solutions to a given COP. A very basic solution method that uses information about the search space, to make the procedure much faster, is called *branch-and-bound.* Branch-and-bound is also often used within other methods and methodologies, like e.g., in mixed-integer linear programming. The whole procedure can be represented as a routed tree, where the root node of the tree contains the whole search space and at every node, the problem is splitted via a decision variable, and this node can be seen as a subproblem of the whole problem. In a branch-and-bound algorithm the *pruning* of nodes is important. A node can be pruned when

- the best objective value obtainable with this node is worse than the best objective value found so far,
- the solution becomes infeasible within this node, and
- the objective is not going to improve anymore when examining subnodes of the incumbent.

Basically, the earlier a node can be pruned, the better, as it makes the algorithm faster by pruning more nodes that do not need to be evaluated anymore.

2.2.2 Dynamic Programming

Dynamic programming (DP) is also an exact method which follows the principles of *divide and conquer*. The idea is to solve small subproblems and remember the solutions to the subproblems which is also referred to the term *memoization*. DP makes use of a recursive function where the solutions of the subproblem are used and combined so that finally a solution to the whole problem is obtained. Obviously, dynamic programming only makes sense for problems with a particular structure, i.e., the problem can be expressed in a recursive manner and the subproblems are partly overlapping. DP has been successfully applied to various COPs and is therefore an important optimization technique [74, 76, 81]. For a deeper introduction into DP see also the book by Bellman [8].

2.2.3 Linear Programming

Linear programming (LP) defines the problem of minimizing or maximizing a linear cost function subject to linear inequalities. LP has been shown to be in the complexity class $\mathcal{P}$ which means that for any problem which can be formulated as an LP, there exists a polynomial-time algorithm which solves the problem. First of all, we will give a formal definition of a linear program:

$$\begin{aligned}
\min \quad & \mathbf{c'x} && && (2.3)\\
\text{s.t.} \quad & \mathbf{a_i'x} \geq b_i && \forall i \in M_1 && (2.4)\\
& \mathbf{a_i'x} \leq b_i && \forall i \in M_2 && (2.5)\\
& \mathbf{a_i'x} = b_i && \forall i \in M_3 && (2.6)\\
& x_j \geq 0 && \forall j \in N_1 && (2.7)\\
& x_j \leq 0 && \forall j \in N_2 && (2.8)
\end{aligned}$$

The linear cost function (2.3) is also denoted as *objective function.* The feasible region of the problem is described by inequalities (2.4)–(2.6) and the domain of the variables is described in inequalities (2.7) and (2.8). Every set of feasible variable assignments, i.e., which lies in the feasible region of the problem is regarded as feasible solution. The solution from the feasible set of solutions, which minimizes the objective function (2.3) is denoted as *optimal solution.* Note, that there maybe more than one solution that minimizes the objective function in which case all those solutions would be denoted as optimal solution. The value of the objective function for the optimal solution is regarded as *optimal cost.*

An LP is said to be in *standard form* if it looks like:

$$\begin{aligned} \min \quad & \mathbf{c}'\mathbf{x} && (2.9) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = b && (2.10) \\ & x \geq 0 && (2.11) \end{aligned}$$

By transforming all inequalities of an arbitrary LP to the form $\mathbf{A}\mathbf{x} = b$ and having only non-negative variables $\mathbf{x} \geq 0$ every LP can be brought into standard form where the process of bringing an LP into standard form is regarded to *reduction to standard form.*

To formally define the feasible region of an LP we define a *polyhedron* [10].

Definition 1. *A polyhedron is a set that can be described in the form* $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$*, where* $\mathbf{A}$ *is an* $m \times n$ *matrix and* b *is a vector in* $\mathbb{R}^m$*.*

Now, we define a solution of an LP. First of all, we need to describe the term of an *active constraint*:

Definition 2. *A constraint of the form* $\mathbf{a}\mathbf{x}^* = b$ *is said to be active if vector* $\mathbf{x}^*$ *satisfies it.*

Then, we can define *basic solutions* and *basic feasible solutions* as follows:

Definition 3. *Consider a polyhedron* P *defined by linear equality and inequality constraints, and let* $\mathbf{x}^*$ *be an element of* $\mathbb{R}^n$*.*

- *Vector* $\mathbf{x}^*$ *is a basic solution if all equality constraints are active and out of the constraints that are active at* $\mathbf{x}^*$*, there are* n *of them linearly independent.*
- *If* $\mathbf{x}^*$ *is a basic solution that satisfies all of the constraints, it is called a basic feasible solution.*

Definition 3 formally defines what *basic feasible solutions* are, but we are mostly interested in solving the LP to optimality. Thus, we need also information about the optimality of extreme points:

Theorem 1. *Consider an LP minimizing* $\mathbf{cx}$ *over a polyhedron* P*. Suppose that* P *has at least one extreme point and that there exists an optimal solution. Then, there exists an optimal solution which is an extreme point of* P*.*

For a proof of Theorem 1 see [10]. As *basic feasible solutions* and the term optimality are now introduced we can start thinking of algorithms which are able to solve LPs. This is explained in the next section.

Solution Algorithms for Linear Programs

Well known are the following three different methods for solving LPs:

- simplex algorithm,
- ellipsoid method, and
- interior point method.

Since it was shown by Khachiyan [80] that the *ellipsoid method* is able to solve LPs in polynomial time, it is known that LPs are generally in the complexity class $\mathcal{P}$. This was also shown by the work of Karmarkar [79], an Indian mathematician, in 1984, where he proposed the interior point method which is also a polynomial-time algorithm for solving LPs. Even, if the well known *simplex* algorithm of Dantzig [33] has an exponential runtime in the worst-case scenario, it is effective in practice [88] and implemented in most commercial (mixed integer) linear-programming solvers, such as CPLEX or Gurobi. Basically, the idea of the simplex method is to walk along the edges of the feasible polyhedron in direction of reduced costs (in case of a minimization problem) until no further improvement in the current solutions neighborhood is possible. The simplex algorithm terminates after a finite number of steps. A full implementation guide including also performance enhancements can be found in [10].

Duality

Duality is an important property in LP and it is also a substantial property for finding dual bounds. Most notably, every feasible solution found through the dual problem provides a dual bound on the objective value of the primal problem.

Definition 4. *Two problems*

$$(MIP) \qquad z = \max\{c(x) : x \in X\} \tag{2.12}$$

$$(D) \qquad w = \min\{w(u) : u \in U\} \tag{2.13}$$

form a weak-dual pair if $c(x) \leq w(u) \forall x \in X$ and all $u \in U$. If $z = w$, they form a strong-dual pair.

Theorem 2. *Suppose x is a feasible solution to the primal problem and u a feasible solution to its dual problem, then*

$$p \cdot b \leq c \cdot x. \tag{2.14}$$

if its a minimization problem.

Therefore, any feasible solution of the dual problem gives a dual bound on the objective of the primal problem which is a useful property when solving LPs.

2.2.4 (Mixed) Integer Linear Programming

The basics for (mixed) integer linear programming lies in the theory of LP and with the simplex algorithm of Dantzig there already exists a practically efficient solution method for those programs. However, LPs can only cope with real numbers which means that many practical problems cannot be expressed with an LP. Imagine, a company is planning to build new factories and they have to decide where to place these new facilities. Therefore, a decision is needed whether to build or not to build a factory on a particular place. This cannot be expressed by an LP. When integer variables are needed to model a problem we speak about (mixed) integer linear programs. A model which contains only integer variables is called integer program, a model containing only binary decision variables, i.e., all variables can only take a value of zero or one are called binary programs, and if a model contains integer as well as real-valued variables it is a mixed integer linear program.

The question arising is, how MIPs can be (efficiently) solved. An obvious way in solving those models is to solve the LP instead of the MIP by replacing all integer variable domains with a real-valued domain. Rounding the fractional values to integer values could be a solution to retrieve a valid integer solution for the model. But, of course, rounding is not sufficient as this rounded solution can be far away from the optimal integer solution. For finding an algorithm that solves MIPs, some information is needed at which point a given solution $\mathbf{x}^*$ can be proved optimal. Let z be the optimal solution of integer program *IP*, $\bar{z}$ an upper bound of z and $\underline{z}$ a lower bound. The optimal solution is reached when

$$\bar{z} - \underline{z} \leq \epsilon \tag{2.15}$$

where ϵ is a small optimality tolerance. The question is how to find upper and lower bounds to the optimal solution of the MIP. First, we define what a *relaxation* of the original MIP is.

A simple and useful relaxation of *IP* is the so called LP relaxation when the domains of the integer variables are changed to real values. Obviously, the optimal solution to the LP relaxation is a lower bound or dual bound to the optimal solution (in case of minimization). Upper bounds or primal bounds are any feasible solution to the problem/model.

As all other types of problems, also a MIP could be solved to optimality with complete enumeration. However, as this is not tractable a more sophisticated solution approach has to be found. An approach is described in the next section, namely *LP-based branch-and-bound*.

LP-based branch-and-bound

When considering an integer program $z = \max\{cx : x \in S\}$, the question is how it can be divided into subproblems such that it can be solved faster and more efficiently.

Proposition 1. *Let $S_1, \ldots, S_k$ be a decomposition of S into smaller problem, and let $z^k = \max\{cx : x \in S_k\} \forall k = 1, \ldots, K$. Then, $z = \max_k z^k$.*

A way of reaching such a decomposition is to solve the so called LP relaxation of the MIP. Usually, this results in some integer variables being fractional. If this is not the case the optimal LP solution is also the optimal solution of the MIP. A way of eliminating such fractional variables is to branch over these variables and make them integer. Again a full enumeration tree would not be tractable, so a smarter method has to be applied. This is done by computing bounds and pruning the tree, similar as described in Section 2.2.1. In general there are three types of pruning in a branch-and-bound tree:

prune by optimality if lower and upper bound on a node in the tree have the same value there is no need to expand the node further.

prune by bound if the best found primal bound so far is higher (for maximization problems) than the dual bound in the current node it is not possible to find the optimal solution in the subtree of this node.

prune by infeasibility if the solution would become infeasible, the subtree of the current node can be discarded.

Using this information an algorithm can be implemented which uses LP-based branch-and-bound. However, there are further decisions to be made, like, e.g., in which order should the tree be traversed, in which order should the nodes be expanded. These decisions influence the performance of the algorithm.

2.2.5 Decomposition-based Approaches

Solving a whole MIP model, e.g., through LP-based branch-and-bound as introduced in Section 2.2.4, is often not possible because of the huge number of variables and constraints in practicable problem instances. Thus, there are decomposition approaches such that the whole problem may not be solved at once. The idea behind these approaches is to only consider a subset of the constraints or variables. The resulting problem is much easier to solve than the original. In (logic-based) Benders decomposition, and the cutting-plane method, constraints are iteratively added, whereas in column generation variables are iteratively added.

Cutting-Plane Method and Branch-and-Cut

The cutting plane method iteratively adds cuts/constraints to a MIP. The idea is to use this technique inside the nodes of the branch-and-bound tree to tighten the dual bound by recomputing the LP solution with the new constraint(s), which have to be valid for the original problem, and cut off invalid solutions. For finding such cuts, so called *separation algorithms*, may be used. For instance, in the traveling salesman problem (TSP) subtours are not allowed. However, there are exponential many subtour elimination constraints. Thus, those constraints are added in a branch-and-cut procedure. The separation problem would be, to find the minimum cut in the corresponding branch-and-bound node. Note, that there also exist TSP formulation which does not need subtour elimination constraints when Miller-Tucker-Zemlin (MTZ) constraints are added, see [96].

By using the branch-and-cut algorithm, usually less nodes need to be considered, as the dual bound is strengthened through recomputing the LP. Branch-and-cut is a widely applied decomposition method for MIPs. However, finding good and useful cuts and also finding fast separation algorithms is a challenging task.

Benders Decomposition

Benders decomposition is a useful decomposition technique if the original MIP contains so called “complicating variables” and was originally proposed in 1962 [9]. The problem is divided into a master problem (MP) and a subproblem (SP) where the MP contains the “complicating variables” variables and the subproblem contains only continuous variables. Consider a MIP of the form

$$\begin{aligned} MIP = \min \quad & \mathbf{cx} + \mathbf{c'y} && (2.16)\\ \text{s.t.} \quad & \mathbf{Ax} + \mathbf{By} \geq \mathbf{b} && (2.17)\\ & \mathbf{Dx} \geq \mathbf{d} && (2.18)\\ & \mathbf{x} \in \mathbb{Z}^n && (2.19)\\ & \mathbf{y} \geq 0 && (2.20) \end{aligned}$$

In this case the $\mathbf{x}$ variables are the “complicating” ones and the $\mathbf{y}$ variables are continuous. This MIP can be reexpressed in the form

$$\begin{aligned} MP = \min \quad & \mathbf{cx} + z_{SP}(\mathbf{x}) && (2.21)\\ \text{s.t.} \quad & \mathbf{Dx} \geq \mathbf{d} && (2.22)\\ & \mathbf{x} \in \mathbb{Z}^n. && (2.23) \end{aligned}$$

The value of $z_{SP}(\mathbf{x})$ is the solution of the following subproblem, when the values for the “complicating” $\mathbf{x}$ variables are fixed, and the problem becomes a linear program with only continuous $\mathbf{y}$ variables

$$\begin{aligned} SP = \min \quad & \mathbf{c'y} && (2.24)\\ \text{s.t.} \quad & \mathbf{By} \geq \mathbf{b} - \mathbf{Ax} && (2.25) \end{aligned}$$

$$\mathbf{y} \geq 0 \tag{2.26}$$

where the dual form of this subproblem is as follows

$$\begin{aligned} DSP = \max \quad & \mathbf{w} \cdot (\mathbf{b} - \mathbf{A}\mathbf{x}) && (2.27) \\ \text{s.t.} \quad & \mathbf{w}\mathbf{B} \leq \mathbf{c}' && (2.28) \\ & \mathbf{w} \geq 0 && (2.29) \end{aligned}$$

When the dual of the subproblem (DSP) is solved, we can derive two types of cuts. If the dual of the subproblem is unbounded, it is known that the primal problem is infeasible and thus, we derive (Benders) feasibility cuts. Moreover, if the dual is bounded, we derive (Benders) optimality cuts for the MP of the form

$$z \geq \mathbf{c}\mathbf{x} + \mathbf{w} \cdot (\mathbf{b} - \mathbf{A}\mathbf{x}) \quad \forall w \in W \tag{2.30}$$

where W is the set of extreme points obtained by the solution of DSP. The algorithm stops when no cut violating the solution of the MP can be found by solving the dual problem of the SP, and finally, terminates with the optimal solution.

Logic-Based Benders Decomposition

Logic-based Benders decomposition was introduced by Hooker and Ottosson [72] in 1995 and generalizes the classical Benders decomposition from an LP dual to an inference dual. The difference to classical Benders decomposition is that the SP does not necessarily need to be an LP. However, again as in classical Benders decomposition the problem is divided into a MP and a SP. An initial solution to the reduced MP is computed and with this fixed solution, the SP is solved. The SP can create again feasibility and/or optimality cuts. In case of logic-based Benders decomposition the SP often corresponds to a satisfiability problem where also constraint programming can be a promising, if not superior, approach. If cuts could be generated, because the SP has proven infeasibility of the solution to variables of the MP or an optimality cut could be derived, a cut over the current assignment of the variables of the MP is added and the MP is resolved. The algorithm terminates with the optimal solution when no further cuts can be derived in the SP. Note, that the SP must be solved to optimality in order to ensure optimality of the final solution.

There exists also a variant of logic-based Benders decomposition, which is sometimes also referred as *branch-and-check* [150], that adds (Benders) feasibility and optimality cuts in a branch-and-cut manner.

2.3 Heuristics

The word *heuristic* has its origin in the Greek and is derived from the word “heuriskein” which means “to find”. Basically, a heuristic aims at finding (good) solutions for COPs which are not necessarily optimal. Heuristics try to find a way through the search space

Algorithm 2.1: Construction heuristic

```
1: $x = \emptyset$
2: while $extend(x) \neq \emptyset$ do
3:     $c \leftarrow select(extend(x))$
4:     $x \leftarrow x \cup c$
5: end while
6: return $x$
```

of a COP by using predefined rules that evaluate the current state of a solution and also its final objective value. Usually, heuristics are applied to problems when either fast solutions are needed or exact methods fail to work, because the search space is too large for an exact method to be applied. In the following we introduce two basic concepts of heuristics, namely *construction heuristics* as well as *local search.*

This section as well as Section 2.4 and Section 2.5 are partly based on the book by Blum and Raidl [15].

2.3.1 Construction Heuristics

Construction heuristics provide a fast way to construct initial solutions to a problem. In many applications, their solution quality is good enough for solving the problem. They may also provide starting solutions for metaheuristics, or they can even be extended to an own metaheuristic, see also Section 2.4.2 about greedy randomized adaptive search procedure. The pseudo code for a construction heuristic is given in Algorithm 2.1. The variable x consists of solution components and is called the partial solution. In each step the solution is extended with another solution component until the solution is complete or cannot be extended anymore. The function $extend(x)$ returns all solution components for which the partial solution x can be extended in its current partial state. The function will return the empty set, if the solution is either complete or cannot be extended anymore. The *select* function will select a solution component from the available ones. At the end of the construction heuristic the final solution x is returned.

Greedy construction heuristic

A greedy algorithm always makes the choice which seems currently the best even its not the best choice on global perspective. Usually, an objective function is defined and the greedy algorithm selects the next step which has the best objective function value among all given possibilities. The greedy algorithm stops when a full solution has been constructed. In most cases, a greedy algorithm works only with feasible solutions which in turn produces a final feasible solution when the algorithm terminates. A greedy algorithm can also be randomized which in most cases improves the average final solution quality. The pseudo code for a greedy construction heuristic is shown in Algorithm 2.2. The algorithm adds solution components from a given candidate list C until the solution

Algorithm 2.2: Greedy construction heuristic *greedy*(S, C)

```
Require: possibly empty incumbent solution $S$, candidate list $C$
 1: while solution is not complete do
 2:    $x^{\text{best}} \leftarrow$ uninitialized
 3:    for all $c \in C$ do
 4:       if $x^{\text{best}} =$ uninitialized or better(eval($S, c$), $x^{\text{best}}$) then
 5:          $x^{\text{best}} \leftarrow c$
 6:       end if
 7:    end for
 8:    $S \leftarrow S \cup \{x^{\text{best}}\}$
 9:    $C \leftarrow C \setminus \{x^{\text{best}}\}$
10: end while
11: return $S$
```

is complete. In each step of the algorithm, every possible candidate is evaluated and the best possible choice is made. Then, the solution is extended with this locally best possible choice x^{best} and the candidate is removed from the candidate list C.

Basically, a greedy algorithm do not yield optimal solutions except for matroids. When a problem can be expressed as matroid a greedy algorithm is able to yield optimal solutions like it is the case of the minimum spanning tree problem for which, e.g, Kruskal's algorithm yields optimal solutions.

2.3.2 Local Search

Whereas construction heuristics, such as greedy construction heuristics or PILOT, create solutions from scratch, *local search* starts with an initial solution and tries to improve this solution for a given neighborhood function or neighborhood structure. Often it is the case that local search obtains an initial solution by a construction heuristic and tries to improve this solution. Local search is an effective technique in combinatorial optimization and widely applied to improve a particular solution. Given a set of neighborhood structures $\mathbf{N} = N_1(x), \ldots, N_k(x)$ it is possible to compute locally optimal solutions with respect to the the given neighborhoods $\mathbf{N}$. When the local optimum is reached, no further improvement is possible with these neighborhood structures. There exist methods for escaping local optima, like, e.g., variable neighborhood search (VNS), see Section 2.4.3. According to Blum and Roli [16] a neighborhood structure is defined as follows

Definition 5. *Let* $\mathbf{S}$ *be the search space of a given COP. Then, a neighborhood structure is a function* $\mathbf{N} : \mathbf{S} \rightarrow 2^{\mathbf{S}}$ *that assigns to every* $s \in S$ *a set of neighbors* $\mathbf{N}(s) \subseteq \mathbf{S}$*.* $\mathbf{N}(s)$ *is called the neighborhood of* s*.*

A neighborhood simply describes the changes to be applied to a solution to generate its neighbors. The application of a neighborhood which produces solution s' from solution

Algorithm 2.3: Variable neighborhood descent

```
Require: initial solution $x$, neighborhood structures $N_k \mid k = 1, \dots, k_{\max}$
1:  while improvement obtained do
2:      $k \leftarrow 1$
3:      while $k \neq k_{\max}$ do
4:          $x' \leftarrow BestNeighbor(x, N_k)$
5:          if $x'$ better $x$ then
6:              $x \leftarrow x'$
7:              $k \leftarrow 1$
8:          else
9:              $k \leftarrow k + 1$
10:         end if
11:     end while
12: end while
```

s is called a move. When exhaustively applying a neighborhood function $\mathbf{N}$ one gets a *local minimum* which is defined as follows (Blum and Raidl [15]):

Definition 6. *A* local minimum *with respect to a neighborhood function* $\mathbf{N}$ *is a solution* $\hat{s}$ *such that* $\forall s \in \mathbf{N}(\hat{s}) : f(\hat{s}) \leq f(s)$. *We call* $\hat{s}$ *a strict local minimum if* $f(\hat{s}) < f(s) \forall s \in \mathbf{N}(\hat{s})$.

Such a neighborhood function can be searched either in a first improvement or best improvement fashion. In either case the solution found is a *local minimum* with respect to the given neighborhoods.

2.3.3 Variable Neighborhood Descent

Variable neighborhood descent (VND), initially proposed by Mladenović and Hansen [97], is a typical local-search routine which is often used and embedded in various (meta)heuristics and is used to compute locally optimal solution with respect to given neighborhoods.

It is often used within a VNS, see Section 2.4.3. When running a VND, the change in neighborhoods is always performed in a deterministic way, see also Algorithm 2.3.

2.4 Metaheuristics

A definition of the term *metaheuristic* according to Osman and Laporte [104] is the following:

Definition 7. *A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions.*

Although, MIP techniques are powerful, it is often the case that they are not suitable or applicable for practical problems due to too excessive runtime or memory requirements for $\mathcal{NP}$-hard problems. Sometimes, it is also even impossible to obtain any feasible solution or even bounds when trying to solve huge problem instances with exact techniques like MIP-methods, CP or SAT-based approaches. In these cases, (meta)heuristics may be a practically highly promising way to go.

In general, (meta)heuristics do not provide guarantees on solution quality but provide a way to frequently yield good approximate solutions to a problem if well designed. In the following we will describe heuristics which have been used for solving the problems handled in this thesis.

Metaheuristics have already been studied for decades and a good starting point for developing and learning about them is the *Handbook of Metaheuristics* [61].

There is a wide range of metaheuristics described in the literature and it is not in the scope of this work to describe all of them in detail. The following sections sketch the principles of GRASP, PILOT and VNS, which are those applied in the subsequent chapters. Further prominent metaheuristics are, for example:

- population-based metaheuristics
  - ant-colony optimization [42, 43, 44]
  - genetic algorithms [35, 127]
- local-search based metaheuristics
  - tabu search [57, 60]
  - simulated annealing [82, 152]

We also note, that the term of a *hyperheuristic* is gaining popularity which is basically a special metaheuristic layer that is able to adapt the search by selectively utilizing various lower level (meta)heuristics. The book of Sörensen et al. [31] gives an overview on the topic and discusses recent results in this area.

2.4.1 Preferred Iterative Look ahead Technique

This method proposed by Duin and Voß [45, 46, 157], short as PILOT, tries to reduce problems with the greedy trap by looking ahead a certain number of steps. Basically, any greedy algorithm can be enhanced to the PILOT method when looking ahead the incumbent solution. The PILOT method could do a full dry run of the, e.g., greedy algorithm, or the PILOT depth could also be limited in case the runtimes are to large when applying this method, but this clearly is problem- and instance dependent. When setting the depth to zero this can also seen as a special case where the algorithm corresponds to the usual greedy construction heuristic without look-ahead mechanism.

Algorithm 2.4: Preferred Iterative Look ahead Technique

```
Require: candidate list $C$
 1: $S \leftarrow \emptyset$
 2: while solution is not complete do
 3:   $x^{\text{best}} \leftarrow$ uninitialized
 4:   $S^{\text{best}} \leftarrow$ uninitialized
 5:   for all $c \in C$ do
 6:     $S' \leftarrow \text{greedy}(S \cup \{c\}, C \setminus \{c\})$
 7:     if $S^{\text{best}} =$ uninitialized or $better(S', S^{\text{best}})$ then
 8:       $x^{\text{best}} \leftarrow c$
 9:       $S^{\text{best}} \leftarrow S'$
10:     end if
11:   end for
12:   $S \leftarrow S \cup \{x^{\text{best}}\}$
13:   $C \leftarrow C \setminus \{x^{\text{best}}\}$
14: end while
15: return $S$
```

Algorithm 2.5: GRASP

```
1: $S^{\text{best}} = \emptyset$
2: while termination criterion not met do
3:   $S \leftarrow GreedyRandomizedConstruction()$
4:   $S \leftarrow LocalSearch(S)$
5:   if $S$ better than $S^{\text{best}}$ then
6:     $S^{\text{best}} \leftarrow S$
7:   end if
8: end while
```

The pseudo code for PILOT is given in Algorithm 2.4. First, the solution is initialized to the empty set. Then, the algorithm tries to add solution components until the solution is complete. In each iteration of the outer loop, for every candidate $c \in C$ a full greedy look ahead is done, possibly limited by a given look-ahead depth, and x^{best} is the element from the candidate list which has the best solution for a complete look ahead solution. The function *greedy*(S, C) calls the corresponding Algorithm 2.2. After all elements are evaluated with the look-ahead mechanism the solution is extended with the best possible choice and the element is removed from the candidate list.

2.4.2 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) has been proposed by Resende and Ribeiro [128]. Basically, GRASP is a multistart metaheuristic where each

Algorithm 2.6: GreedyRandomizedConstruction

```
Require: Set of candidates $C$
1: $S = \emptyset$
2: $\forall c \in C$ : evaluate incremental costs
3: while solution is not complete do
4:    Build restricted candidate list (RCL)
5:    pick $c \in RCL$ at random
6:    $S = S \cup \{c\}$
7:    Reevaluate incremental costs
8: end while
9: return $S$
```

Algorithm 2.7: Variable neighborhood search

```
Require: initial solution $x$, shaking neighborhood structures $N^{s}_{l} \mid l = 1, \ldots, l_{\max}$, local
    search neighborhood structures $N_k \mid k = 1, \ldots, k_{\max}$
1: while termination criteria not met do
2:    $l \leftarrow 1$
3:    while $l \neq l_{\max}$ do
4:       $x' \leftarrow Shaking(x, N^{s}_{l})$
5:       $x'' \leftarrow VND(x', N_k)$
6:       if $x''$ better $x$ then
7:          $x \leftarrow x''$
8:          $l \leftarrow 1$
9:       else
10:         $l \leftarrow l + 1$
11:      end if
12:   end while
13: end while
```

step of the metaheuristic consists of a construction phase and a local search phase. The general procedure of GRASP is given in Algorithm 2.5.

When looking at Algorithm 2.6, GRASP makes use of a *restricted candidate list* (RCL). This list usually contains a number of best elements from which one element is chosen at random to be the next part of the solution. The size of RCL obviously directly influences performance and solution quality of the algorithm and should be chosen well.

2.4.3 Variable Neighborhood Search

The Variable neighborhood search (VNS) [97] metaheuristic uses a so called *shaking* mechanism to construct a random point which allows the metaheuristic escaping local optima. VNS is a powerful technique for many real-world applications requiring good solutions in reasonable runtimes. Pseudo code of the VNS is shown in Algorithm 2.7. There are many variations, extensions and hybrids of the basic VNS algorithm where the most prominent ones are reduced VNS (RVNS), variable neighborhood decomposition search (VNDS), parallel VNS (PVNS) and skewed VNS (SVNS).

2.5 Hybrids

We introduced two different guided ways for solving COPs, namely exact methods and (meta)heuristic methods. Both have their advantages and shortcomings and thus, it is occasionally meaningful to combine them to more powerful methods. On one hand, (meta)heuristics are usually fast and are able to yield solutions to larger instances than exact methods. On the other hand exact, methods are able to yield proven optimal solutions (for smaller instance sizes), but they can also provide bounds to the particular COP. Especially, using pure (meta)heuristics it is often hard or impossible to provide dual bounds whereas, e.g., MIPs always can at least provide easily dual solutions via the simple LP dual solution.

There are multiple possibilities of combining (meta)heuristics and exact methods. For instance, (meta)heuristics can also help to boost the efficiency of exact methods by providing good primal bounds to MIPs. In a branch-and-cut manner, heuristics can also support MIPs for finding cutsets or detecting invalid solutions in a fast way. Often, exact methods are also used within (meta)heuristics as, e.g., solving subproblems by LP or MIP. A good example for such a use case can be found in Section 5.3 where the overall problem is solved using a metaheuristic and the subproblem is either solved by LP or DP.

We give a taxonomy of hybrid metaheuristics as proposed by Raidl [119, 120]. However, we want to note, that also Talbi proposed a taxonomy for hybrid metaheuristics [149]. Raidl distinguishes hybrid metaheuristics among the following properties:

hybridized algorithms: metaheuristics with metaheuristics, metaheuristics with problem-specific algorithms/simulations, metaheuristics with other operations research/artificial intelligence techniques, and metaheuristics with human interaction

control strategy: integrative and collaborative

order of execution: batch (sequential), interleaved, parallel

level of hybridization: high-level (weak coupling), low-level (strong coupling)

In the last decade, a lot of effort has been put on developing hybrid metaheuristics as it seems that they are one of the best solution methods to solve large real-world problems. In fact, if well and sophisticated designed, they provide powerful solution methods. Therefore, many surveys [18, 119, 120] and books [15, 17, 149] have been published about hybrid metaheuristics and how their strength can be exploited best.

Giving a full list of hybridization possibilities and methods is out-of-scope for this thesis and even not possible, as in general, they can be combined as desired. However, to design a good hybrid method, one has to be careful what to combine and how to combine it. We will give some examples on hybrids in the remainder of this section. First, we discuss the combination of metaheuristics with metaheuristics, then we will shortly show possibilities of how to combine MIP approaches with metaheuristics, which are also sometimes denoted as *matheuristics*, and in the end we show some methodologies which can be used to solve large-scale real-world problems, namely so called *multistage approaches*.

2.5.1 Combining Metaheuristics with Metaheuristics

Basically we can distinguish between local-search (trajectory) based metaheuristics and population-based metaheuristics. These two types of metaheuristics can be well combined with each other to exploit/combine both advantages. In general, population-based algorithms profit of keeping multiple diverse solutions whereas trajectory-based metaheuristics are able to intensify the search by looking up the neighborhood for a locally optimal solution, based on a particular given solution. An example of this hybridization would be the combination of genetic algorithms with, e.g., VND. The advantage of this combination is that the population-based metaheuristic, the genetic algorithm in this case, can be used for diversifying the search by holding a whole population and the VND can be used to intensify the search by locally optimizing each solution of the population. These algorithms are called *memetic algorithms* [99, 100]. Another successfully applied combination of trajectory-based methods with population-based ones are ant-colony optimization and beam search. Beam search is a tree-search based method which allows to extend partial solutions in a parallel greedy fashion for increasing the probability of finding a good solution [107]. Blum [13] refers this kind of hybridization also as *Beam-ACO*. This hybridization has been successfully applied to several prominent problems, such as the traveling salesman problem with time windows [93] and the longest common subsequence problem [14].

2.5.2 Combining Metaheuristics with Mixed Integer Linear Programming

It has also become very popular to combine metaheuristics with DP, LP or MIP to take advantage of both methodologies. Raidl and Puchinger [121] gave a good overview about the techniques and how they can be combined.

It is often useful to embed (meta)heuristics within an exact approach to provide the exact approach with primal bounds which can also help to keep the branch-and-bound tree smaller if the metaheuristic can provide good primal bounds to the exact approach. Obviously, it is, however, also a trade-off because performing metaheuristic methods at every node of a branch-and-bound tree is also very time consuming. Thus, clever decisions have to be made, whether a metaheuristic should be called in a particular branch-and-bound node or not. Alternatively, if suitable also faster and/or simpler heuristics can be called more often. In a decomposition-based exact approach like, e.g., branch-and-cut, it can also be useful to use metaheuristics to find cutsets faster and even larger cutsets which may speed up the computation of the MIP, see, e.g., Raidl et al. [123].

On the other hand, for metaheuristics, it can also be useful to guide them by relaxations of LPs or MIPs. Sometimes it can be useful to solve the LP-relaxation of a problem, and then, trying to repair infeasible solutions for providing good initial solutions to metaheuristics. Solutions providing dual bounds from, e.g., the LP-relaxation of a problem can also be exploited by metaheuristics, utilizing the primal-dual relationships.

Another possibility, which is sometimes also referred as *fix-and-optimize* strategy [156], is to fix some variables and let the other variables free, so that they can be optimized by a MIP solver like CPLEX or Gurobi. This is often done within a *(very) large neighborhood search* (VLSN). Ahuja et al. [1] provided a survey on very-large scale neighborhood search techniques.

2.5.3 Multistage Approaches

Multistage approaches are particularly useful for (very) large real-world problems. They decompose the problem such that the original problem may not be solved at once, but only a smaller problem, or the problem can even be divided into a master and a subproblem. In the first part of this section, we describe the *cluster-first route-second* methodology originally proposed by Fisher and Jaikumar [50]. This is particularly useful for vehicle routing problems (VRP), when at first the given items, which need to be visited are assigned to a particular vehicle, and in a second step for each vehicle a routing problem is solved separately. The second metaheuristic we describe, is the multilevel refinement strategy, initially proposed for combinatorial optimization by Walshaw [158], where the original, usually large, problem is coarsened until a problem size is reached which can be easily solved. After that, the problem is iteratively extended and refined (possibly by local search such as a VND) until the lowest level is reached and a solution to the original problem is retrieved.

Cluster-First Route-Second

Cluster-first route-second is a decomposition-based metaheuristic and was first proposed by Fisher and Jaikumar [50] in 1981. The basic idea is to divide the problem into a master problem and a subproblem, where the master problem is associated with the clustering and the subproblem is an independent problem, which is much easier to solve. The union of the solutions from all subproblems corresponds to the solution of the overall problem. Note, that, e.g., for the vehicle routing problem with maximum-tour constraint an estimation of the tour length of each cluster has to be integrated into the master problem. The success of this approach is also dependent on the quality of this estimation, which results in a better assignment for the subproblem. The subproblem is usually easy to compute respectively solve.

Algorithm 2.8: Multilevel refinement

```
1: $l \leftarrow 0$
2: **while** $P_l$ is too large to be reasonably considered in a direct way **do**
3:     $P_{l+1} \leftarrow \mathit{coarsen}(P_l)$
4:     $l \leftarrow l + 1$
5: **end while**
6: $x_l\{P_l\} \leftarrow \mathit{initialize}(P_l)$
7: **while** $l > 0$ **do**
8:     $l \leftarrow l - 1$
9:     $x_l^0\{P_l\} = \mathit{extend}(x_{l+1}\{P_{l+1}\}, P_l)$
10:    $x_l\{P_l\} = \mathit{refine}(x_l^0\{P_l\}, P_l)$
11: **end while**
```

It is also possible to do this the other way round, which would result in the route-first cluster-second methodology, which has been shown by Prins [115], to be also a fruitful approach for VRPs. This way, a giant tour is constructed, and then, by applying splitting algorithms, multiple tours are created, such that e.g., in the VRP, each vehicle has its own route.

Multilevel Refinement

The multilevel refinement paradigm is a metaheuristic which, e.g., can be used to improve results of existing approaches. It was proposed by Walshaw [158, 159] and it was shown to be able to improve results of existing approaches. This is an intuitive paradigm as it tries to make the problem size smaller and first solve a small variant of the instance and then iteratively extend solutions to obtain a solution to the initial problem instance. After each extension step a possible refinement can also be made, like, e.g., VND (see Section 2.3.3). The process of making the initial problem size smaller is also denoted as *coarsening*.

The procedure is shown in pseudo-code in Algorithm 2.8. The problem is coarsened until an instance size is reached which can be easily solved, possibly by exact methods like LP or MIP. After the coarsening is finished, the solution x_l is initialized at the highest respectively coarsest level. Then, the solution is iteratively extended to a lower level $l - 1$ and optionally refined at the level $l - 1$. If level $l = 0$ is reached, a solution to original problem instance is found. Crucial decisions in this metaheuristic are the criterion when to stop coarsening and how to most accurately coarsen the instance, as usually information is lost while coarsening. Keeping as much information as possible in the upper levels of the coarsening is crucial for the success of the algorithm.

CHAPTER 3

Balancing Bike-Sharing Systems

In this chapter, the work on the Balancing Bike Sharing Systems (BBSS) problem is presented. Essentially, we show three approaches. We present the static BBSS problem which was published in the *Journal of Global Optimization* [125]. Subsequently, the dynamic BBSS is discussed where we have been nominated as best paper candidate at the *14th European Conference on Evolutionary Computation in Combinatorial Optimisation* [86]. In the end, we introduce a simplified problem formulation, where we first proposed a cluster-first route-second heuristic (see Section 2.5.3), published at International Conference on Computer Aided Systems Theory [87] which is then extended to an approach based on logic-based Benders decomposition (see Section 2.2.5) which yields optimal solutions to practical relevant problem instances for which the input data is also based on real-world data. This approach has been published in *Networks* [83].

3.1 Introduction

Public bike sharing systems (PBSs) provide a modern way of shared public transport within cities. These systems, most frequently, consist of *rental stations* distributed in parts of a city. In state-of-the-art PBSs every station has a self-service computer terminal authenticating the customers, and ideally also used to allow instant registration for new clients. Customers have to authenticate and provide a payment method to reduce theft and vandalism. Rental stations consist of *slots* which can either be empty or occupied by a bike. These slots are connected to the whole computer system allowing the operators as well as the customers to have an overview of the status of each station. If there is at least one slot occupied by a bike, customers have the opportunity to rent a bike via the terminal, and if there is at least one slot free, customers may return a bike by putting it into the free slot. To work well, a PBS has to have a reasonable density of stations in the covered region. Users can rent bikes at any station and return them at any other station. An overview of the structure of a PBS is shown in Figure 3.1. Note that user demand can be satisfied via different stations. For instance, the demand between o_1 and t_1 is satisfied by utilizing two distinct bike-sharing stations. A PBS should not be confused with classical bike rental as both have different use cases, client bases and revenue models. The major differences are that in PBS short-term usage is promoted whereas in bike rental longer rental times are not unusual, PBSs are distributed over a larger area, whereas bike rentals are more stationary with bikes usually to be returned at the same place where they have been rent [139].

PBSs are mostly implemented in public-private partnership and are financed through advertisements on the bikes, subsidies from the municipalities, and subscription fees from the users. The costs for building and operating the system have to be covered. The problem of building or extending a PBS can in principle be seen as a facility location or hub location problem with network design aspects [91] and more detailed information about it can be found in Section 4.

For continuous operation of the system, besides maintaining the bikes and stations, providers in particular have to take care of *rebalancing* bikes among the stations such that users can rent and return bikes at any station with high probability. Stations should ideally neither run full nor empty, as these situations obviously significantly impact customer satisfaction.

Different approaches to achieve and maintain a reasonable balance exist. Most commonly, the PBS operator actively rebalances the stations by employing vehicles with trailers that pickup bikes at stations with excess of bikes and deliver them to stations with a lack of bikes. This is the scenario that will be considered within this work, but there are also alternative approaches in which balance should be achieved by the users themselves [53, 111]. There, the operator provides incentives for their customers to rent bikes at stations with excess and to return them at stations with a lack of bikes. These incentives can be reduced subscription fees, prizes or discounts at special partners of the PBS. Both rebalancing strategies can also be used in conjunction.

The active rebalancing of a PBS by a vehicle fleet has in the literature been referred to as a *capacitated single commodity split pickup and delivery vehicle routing problem with multiple visits* [125]. Diverse variants of this problem, with different objectives and constraints, have already been considered, and different algorithmic approaches have been proposed, ranging from *mixed integer linear programming* (MIP) methods to metaheuristics and hybrids. To our knowledge, all these approaches allow for an arbitrary number of bikes to be picked up at some stations and delivered to other stations, just limited by the vehicles' and stations' capacities. Observations in practice, however, indicate that in a larger well-working bike sharing system it makes rarely sense to move only few bikes for rebalancing. Drivers actually almost always pickup a full vehicle load and deliver it completely to another station. Many stations even require several visits with full load pickups or deliveries. Due to budgetary reasons, typically only just enough drivers and vehicles are employed to achieve a reasonable balance most of the time, but basically never an ideal one where single bikes play a substantial role. Drivers should use their limited working time in a best way to optimize the PBS's overall balance as far as

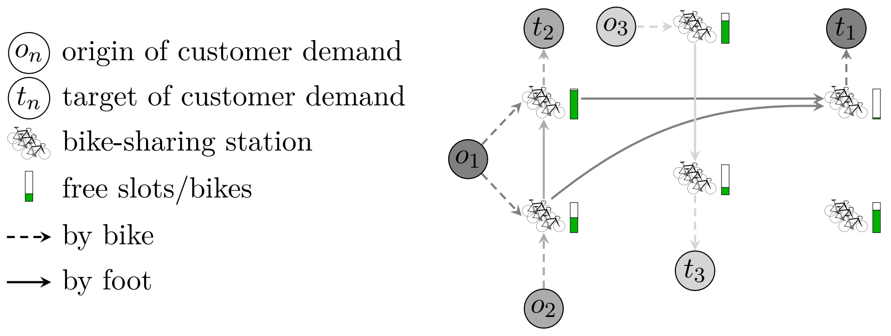


Figure 3.1: A small example of a public bike sharing system is shown. Customers want to travel from their origins o_n to their desired target destinations t_n. They have to walk to the first bike-sharing station, travel to the second bike-sharing station by bike, and then, walk the last part to their desired target. Since the starting bike-sharing station and the final bike-sharing station can be chosen arbitrarily by the users and the stations have fixed capacities, rebalancing becomes necessary.

possible. The described scenario is particularly true in case of our collaboration partner Citybike Wien.

In this work, we consider three different problem variants of the BBSS problem, each one particularly useful in different scenarios:

- the static BBSS problem,
- the dynamic BBSS problem,
- and a variant considering only full vehicle loads.

3.2 Related Work

Nowadays, there also exist practical implementation guides for PBS, mostly inspired by practical considerations and experience. The book *"Bicycle Sharing 101: Getting the Wheels Turning"* [139] discusses nearly all aspects of PBSs from an introduction over the conceptualization to maintenance and implementation. Another book *"The Bike-share Planning Guide"* [55] was published by the Institute for Transportation & Development Policy located in New York.

In this section we give an overview on existing algorithmic approaches for finding reasonable routes for balancing PBSs and other problems related to our simplified problem formulation considering full vehicle loads only.

As already pointed out in the above section, essentially all existing models for rebalancing PBSs consider flexible numbers of bikes to be loaded or unloaded at each visit, and most work addresses the static case only. Several different problem variants with different objectives and side constraints exist, and different solution approaches have been proposed for them. Direct comparisons are therefore quite hard. Many of the described approaches rely on MIP techniques, but there also exist (meta)heuristics and hybrid metaheuristics, which appear to be particularly well suited for larger scenarios.

In his PhD thesis, Vergeylen [153] also aims at solving the bike sharing repositioning problem, but considers a different view on the problem. In his problem definition, he considers a decomposition-based approach. First, a list of requests is given which defines the actions needed to be taken such that the system keeps balanced. In a second process, the best selection of requests needs to be made and these requests have to be assigned to a particular vehicle from the existing fleet. Basically, the thesis consists of two parts where in the first part it is described, how routes are generated from the list of repositioning requests whereas the second part deals with the prediction, when a particular station becomes full or empty.

Before starting with the literature review it should be pointed out that an overview paper about *shared mobility systems* has been published by Laporte et al. [90] containing chapters about rebalancing incentives and vehicle repositioning approaches.

3.2.1 MIP Approaches

Chemla et al. [21] proposed an exact branch-and-cut approach for the single-vehicle case considering it a hard constraint to exactly reach all given target fill levels. The approach is based on a relaxed MIP model yielding a lower bound and a tabu search for obtaining heuristic solutions and thus upper bounds.

Raviv et al. [126] proposed several MIP models minimizing user dissatisfaction and operational costs. These include a time-indexed as well as an arc-indexed formulation which is restricted in the sense that a station may only be visited once by the same vehicle. They also incorporate loading and unloading times proportional to the number of bikes moved. By additionally applying algorithmic enhancements to their MIP models they are able to solve instances up to 60 stations with reasonable optimality gaps.

Schuijbroek et al. [140] describe approaches for determining service level requirements at the stations and vehicle routes for the rebalancing at the same time. An initial MIP model turns out to be intractable for instances of practical size. Consequently, the authors derive a cluster-first route-second heuristic where they first assign stations to clusters by a MIP model and then they solve an independent vehicle routing problem (VRP) for each cluster. In our approach, we will follow a similar basic idea for decomposition, but extend it to an exact LBBD.

Similarly to Schuijbroek et al., Erdoğan et al. [47] define demand intervals for each station. They consider only the single-vehicle case and aim at minimizing traveling costs for the vehicle and handling costs for the rebalanced bikes. Erdoğan et al. present a branch-and-cut formulation, apply valid inequalities from the VRP literature and also present a Benders decomposition scheme. Their approaches solve instances up to 50 stations to optimality.

3.2.2 (Meta)Heuristics and Hybrid Approaches

Due to the practical complexity of BBSS, (meta)heuristics appear also particularly meaningful especially for larger systems and many of them have already been proposed in the literature. Rainer-Harbach et al. [124] introduced a greedy construction heuristic (GCH) and a variable neighborhood search (VNS) with an embedded variable neighborhood descent. These methods have been tested for instances with up to 700 stations, for which they provided very reasonable results. Papazek et al. [109] have developed a pilot heuristic [157] which improved the GCH from [124] significantly, a greedy randomized adaptive search procedure (GRASP) upon both construction heuristics, performing very well on instances with a high number of rental stations. Raidl et al. [122] examined different strategies for determining optimal loading and unloading decisions for given routes within a metaheuristic by specialized maximum-flow and linear programming approaches. Rainer-Harbach et al. [125] refined their work on metaheuristics for the static case by providing comprehensive computational tests and have also introduced their time-indexed and hop-indexed MIP models. Papazek et al. [110] investigated diverse path relinking extensions for GRASP.

The dynamic case was considered by Kloimüllner et al. [86], who proposed a problem model in which flexible demand functions in dependence of time can be considered for all the stations. By separating the demand functions into continuous monotonic pieces and dealing with them appropriately, a complete discretization of time could be avoided. As solution approaches, the authors extended the GRASP and VNS metaheuristics from [125]. The VNS was able to solve instances with up to 90 stations reasonably well.

Di Gaspero et al. further describe a constraint programming approach [41] and a hybridization of it with ant colony optimization [40]. They tested on the same benchmark set as Rainer-Harbach et al. [125]. Although the hybrid ant colony optimization performed better than the pure constraint programming, these methods were not able to yield competitive results.

Vogel et al. [156] propose a MIP model for the resource allocation problem arising in PBSs. They aim at minimizing the traveling costs as well as the handling costs for the relocated bikes. Furthermore, they add a penalty to the objective function for missing bikes and missing free slots at the stations. As for real-world instances the size of the MIP model is too large to be solved directly, the authors suggest a MIP-based large neighborhood search following a fix-and-optimize strategy.

Forma et al. [51] propose the following 3-step hybrid metaheuristic. First, stations are clustered according to geographical data and initial inventory by using a savings heuristic. In a second step, it is decided which vehicle visits which clusters of stations by using a revised MIP model originally stated in [126]. Vehicles are allowed to visit multiple clusters but one cluster is assigned to exactly one vehicle. In a third step, routing problems are solved for each cluster independently. The authors report results for instances with up to 200 stations and three vehicles.

Sörensen and Vergeylen [144] present the bike request scheduling problem which aims also at rebalancing a PBS but from a different perspective. In the problem formulation, a request list for repositioning bikes has to be generated and this request list has to be scheduled. Sörensen and Vergeylen also present solution space analysis [154] therefore.

3.2.3 Other Related Problems and Approaches

Obviously, our simplified BBSS model in which only full vehicle loads are considered is related to diverse other vehicle routing and in particular pickup and delivery problems. There are, however, several special aspects that need to be considered by a meaningful solution approach, in particular that not all stations need to be visited, that a time budget is given, and that tours are sought on a bipartite graph.

A similar problem occurs in the domain of waste collection, for which Aringhieri et al. [3] describe a GRASP and a tabu search. In this problem there is also given a bipartite graph resulting in alternating tours between pickup and delivery places. However, multiple commodities representing different types of waste are considered there. The objective is to reduce the number of tours needed to dispose all the waste and thus, collecting all the waste is considered here as hard constraint, whereas we aim to optimize the quantity of moved commodity within the given time budget.

Another problem related to the one introduced here is the one-commodity full-truckload pickup and delivery problem (1-FTPDP) proposed by Gendreau et al. [59]. This is a variant of the well-known pickup and delivery problem where a truck has to alternatively visit pickup as well as delivery customers all demanding a unit-capacity pickup respectively delivery. In contrast to our problem the supply and demand of each customer has to be satisfied. Thus, the authors add copies of the depot either to the set of pickup customers or to the set of delivery customers to ensure enough supply respectively demand of the customers. There are no time-budget constraints and all customers have to be visited exactly once. The authors model the problem by solving a routing problem through the set of pickup customers and then, assign the delivery customers to the pickup customers. The problem is solved to optimality by relying on classical and generalized Benders decomposition. They also present a traveling salesman problem (TSP) formulation of the problem based on classical subtour elimination constraints. Starting with an initial empty set of subtour elimination constraints they separate them by detecting all connected components and adding subtour elimination constraints for them accordingly. They compare their two approaches based on classical and generalized Benders decomposition with their TSP formulation and conclude that the TSP formulation outperforms the classical as well as the generalized Benders decomposition, although the authors note that there is room for improvement of the approaches based on Benders decomposition.

Related to our problem formulation also is the one-commodity pickup and delivery traveling salesman problem (1-PDTSP) described by Hernández-Pérez et al. [65, 68, 66, 67], and the selective pickup and delivery problem (SPDP) studied by Ting and Liao [151]. In the 1-PDTSP a depot and several customers are given which are either pickup or delivery customers and the aim is to find a minimum distance route visiting all customers starting and ending at the depot and satisfying all the supplies and demands. In addition, Salazar-González and Santos-Hernández [135, 69] introduce the split-demand one-commodity pickup and delivery traveling salesman problem where a truck has to visit a number of delivery and pickup customers multiple times respecting a maximum number of visits per customer. Also the depot may be visited multiple times. However, they do not consider time-budget constraints and all demands have to be fulfilled. They propose an exact model which is solved by Benders decomposition where the separation problem is modeled as a maximum-flow problem. They report interesting and excellent results on an extensive set of benchmark instances. In the SPDP not all pickup nodes have to be visited, but all delivery demands need to be fulfilled. Moreover, somewhat related also is the prize collecting traveling salesman problem introduced by Balas [5], in which a prize is paid for every visited city and/or a penalty has to be paid for each city which is not visited. A minimum prize money has to be earned, and the objective is to minimize the routing costs as well as the penalty incurred by cities which have not been visited.

Especially when considering our decomposition approach which will be described in Section 3.6.2, we obtain as subproblems independent Hamiltonian path problems for the individual vehicles. These problems can be modeled as classical asymmetric TSPs (ATSP) on bipartite graphs. Concerning this special TSP variant, not much specific work exists. To the best of our knowledge, Frank et al. [133] have been the first researchers considering bipartite, symmetric TSPs for which they proposed a 2-factor approximation algorithm. Srivastav et al. [145] analyzed the problem of finding tours for pick-and-place robots which showed up of consisting of a an assignment problem and a bipartite TSP. Given an initial bin assignment the authors proposed several approximation algorithms for the bipartite TSP. Further work on approximation algorithms for the bipartite TSP was done by Baltz and Srivastav [7] as well as Shurbevski et al. [142]. However, these algorithms are more of theoretical interest. We will apply the well-known *Concorde* TSP solver [2, 30] to tackle these subproblems, not further exploiting the underlying bipartite graph structure.

Another related problem is the clustered vehicle routing problem which is basically a generalization of the capacitated vehicle routing problem. This problem was introduced by Sevaux and Sörensen [141] and Defryn et al. [36] present and apply a solution framework therefore.

3.3 Static Balancing Bike Sharing Systems Problem

In this section we describe the static BBSS problem and organize it as follows: The next section formalizes the problem, while Section 3.3.2 describes the construction heuristic and its extension to a PILOT method. Four alternative methods for deriving loading instructions from candidate tours are discussed in Section 3.3.3. Sections 3.3.4, 3.3.5, and 3.3.6 describe the VND, GRASP, and VNS approaches including the used neighborhood structures, respectively. Information on the test instances and the results of diverse experiments are given in Section 3.3.7. Finally, Section 3.3.8 draws conclusions and sketches promising future work.

3.3.1 Problem Definition

We start by providing a formal definition of the BBSS problem. In this work we consider the static problem variant that neglects any user activities during the rebalancing process and where we strive to reach a target fill level of bikes that is pre-specified for each station. Suitable target fill levels are obtained in practice from a statistical demand forecast model that considers several aspects such as season, day, time, as well as the weather forecast [131]. This is another major research issue that exceeds the scope of the current article. By using such models operators are able to estimate reoccurring demands quite well in order to derive expected target values. Note that in most practical scenarios this static case of BBSS is already a useful approximation, since stations are usually designed sufficiently large in order to compensate short-term fluctuations. However, the balancing is still necessary because imbalances arise over longer time horizons, such as one or several days.

The BSS is represented by a complete directed graph $G_0 = (V_0, A_0)$. Node set $V_0 = V \cup \{0\}$ consists of nodes for the rental stations V and a node 0 for the depot (i.e., parking place of the vehicles). Each arc $(u, v) \in A_0$ has associated a time $t_{u,v} > 0$. This value not only includes the time needed for traveling from u to v, but also an expected average time needed for parking, handling the local computer terminal, and loading or unloading bikes at v. Let the subgraph induced by the bike stations V only be $G = (V, A)$, $A \subset A_0$.

Each station $v \in V$ has associated three values: The capacity $C_v \geq 0$, i.e., the number of available bike parking positions, the number of available bikes at the beginning of the rebalancing process p_v, and the target number of bikes that should ideally be available after rebalancing q_v, with $0 \leq p_v, q_v \leq C_v$.

The BSS operator has a fleet of vehicles $L = \{1, \ldots, |L|\}$ that is available for moving bikes between stations. Each vehicle $l \in L$ has a capacity to transport $Z_l > 0$ bikes simultaneously, a total time budget $\hat{t}_l$ within which it has to finish a route, i.e., the worker's shift length. Each route has to start and end at the depot 0. We assume that all vehicles start and finish their routes empty. A practical rationale behind this is that frequently vehicles are publicly accessible at the depot and bikes cannot be locked at the vehicles' trailers.

Solutions to the BBSS problem consist of two parts. The first one is the route for each vehicle $l \in L$ specified by an ordered sequence of visited stations $r_l = (r_l^1, \ldots, r_l^{\rho_l})$ with $r_l^i \in V$, $i = 1, \ldots, \rho_l$ and ρ_l representing the number of stations traveled to. Note that stations may be visited multiple times by the same or different vehicles. For reasonable solutions these multiple visits are necessary as the station capacities C_v are sometimes much larger than the vehicle capacities Z_l.

The second part of a solution consists of loading instructions $y^i_{l,v} \in \{-Z_l, \ldots, Z_l\}$ with $l \in L$, $v \in V$, and $i = 1, \ldots, \rho_l$, specifying how many bikes are to be picked up ($y^i_{l,v} > 0$) or delivered ($y^i_{l,v} < 0$) at station v at the i-th stop of vehicle l. Of course loading actions may only take place at visited stations, i.e., $\forall v \neq r^i_l : \; y^i_{l,v} = 0$, and thus, for simplicity we also write y^i_l for y^i_{l,r^i_l}, i.e., if no station index is explicitly specified we assume the station to be the visited one (r^i_l).

Note that an option would be to further limit the domains of these loading instructions by the station capacities, i.e., $y^i_{l,v} \in \{-\min(Z_l, C_v), \ldots, \min(Z_l, C_v)\}$. We, however, stay more general and potentially allow vehicles meeting at a station to exchange bikes directly. Imposing a limit based on station capacities would be too restrictive in this case.

Several conditions must hold for a solution to be feasible: The number of bikes available at each station $v \in V$ always needs to be within $\{0, \ldots, C_v\}$. For any vehicle $l \in L$ the number of simultaneously transported bikes may never exceed the capacity Z_l, and the total tour length t_l

$$t_l = \begin{cases} t_{0,r^1_l} + \sum_{i=2}^{\rho_l} t_{r^{i-1}_l, r^i_l} + t_{r^{\rho_l}_l, 0} & \text{for } \rho_l > 0 \\ 0 & \text{for } \rho_l = 0, \end{cases} \tag{3.1}$$

is restricted by the time budget $\hat{t}_l$, $\forall l \in L$.

Let a_v be the final number of bikes at each station $v \in V$ after the rebalancing operation

$$a_v = p_v - \sum_{l \in L} \sum_{i=1}^{\rho_l} y^i_{l,v}. \tag{3.2}$$

The objective is to find a feasible solution that primarily minimizes the deviation from the target number of bikes $\delta_v = |a_v - q_v|$ at each station $v \in V$ and secondarily the number of loading activities including the overall time required for traveling all routes. Therefore, our objective function is given by

$$\min \quad \omega^{\text{bal}} \sum_{v \in V} \delta_v + \omega^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} |y^i_l| + \omega^{\text{work}} \sum_{l \in L} t_l, \tag{3.3}$$

where $\omega^{\text{bal}}, \omega^{\text{load}}, \omega^{\text{work}} \geq 0$ are scaling factors controlling the relative importance of the respective terms. Following the advice from experts at Citybike Wien, we assume that any improvement in balance is always preferred over decreasing the number of loading actions or reducing the work time, and to ensure this preference we use appropriate scaling factors. In all our tests we use the setting $\omega^{\text{bal}} = 1$ and $\omega^{\text{load}} = \omega^{\text{work}} = 1/100\,000$.

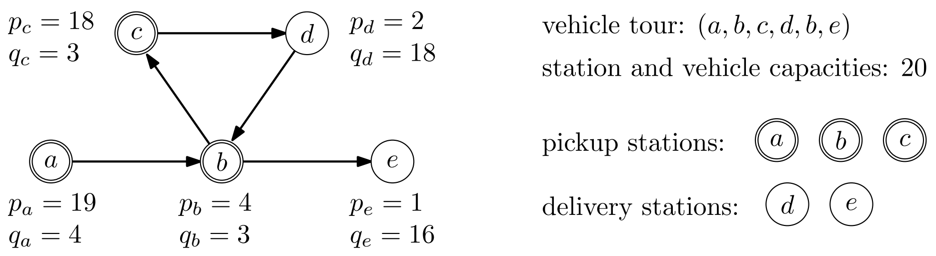


Figure 3.2: Example where the restriction to monotonicity yields a worse solution. With monotonicity, the best possible loading instructions are $y_1 = (+15, +1, +4, -16, 0, -4)$ resulting in a total imbalance of 22. In the general case, node b can be used as buffer and loading instructions $y_1 = (+15, -14, +15, -16, +15, -15)$ yield perfect balance.

Monotonicity for Fill Levels of Stations

A natural simplification for the BBSS problem is the restriction to *monotonicity* regarding the fill levels of stations. By exploiting it we will see that algorithms for deriving good or optimal loading instructions for given tours become simpler while in general solutions are not substantially worse in comparison to the general case.

Let $V_{\text{pic}} = \{v \in V \mid p_v > q_v\}$ denote *pickup stations*, i.e., the set of stations from which ultimately bikes should be removed, and $V_{\text{del}} = \{v \in V \mid p_v < q_v\}$ denote the set of *delivery stations*. The remaining stations $V \setminus V_{\text{pic}} \setminus V_{\text{del}}$ are initially already in balance.

In the monotonic case, vehicles are only allowed to load bikes at pickup stations and unload them at delivery stations. In this way a station's fill level only decreases or increases monotonically, and consequently the order in which different vehicles visit a single station does not matter. Stations that are already balanced at the beginning do not need to be considered at all as no pickups or deliveries are allowed there.

While monotonicity appears to be a very intuitive simplification, enforcing it may exclude better solutions that, e.g., use stations as buffers to temporarily store bikes or by exchanging bikes between vehicles when they meet at some stations. An example of such a situation is shown in Figure 3.2.

Experiments in Section 3.3.7 will show that the impact of monotonicity on the objective values of solutions is recognizable but small. We assume that this trend also depends on the scaling factors in the objective function which put a substantially lower weight on the traveling time than on the imbalance. In practice, excellent solutions can be found even under the assumption of monotonicity.

3.3.2 Construction Heuristics

We present two construction heuristics aimed at generating meaningful initial solutions within short time. The first basic heuristic, presented in the following subsection, has already been used in [125] and follows a classic greedy principle, but utilizes a greedy function specifically designed for BBSS. While fast, local greedy decisions can be far from optimal with regard to the whole solution. This is especially true for BBSS as the greedy function is a compromise that combines multiple objectives. To mitigate this problem, we extend the basic heuristic by evaluating each candidate station considered for addition to a partial tour in a deeper way by also considering its potential successors via recursive calls. This second approach follows the PILOT method [157] and is described in Section 3.3.2. Both methods assume monotonicity regarding fill levels of stations as defined in Section 3.3.1.

Greedy Construction Heuristic (GCH)

This greedy method builds solutions by iteratively creating a tour for each vehicle following a local best successor strategy. From the last station of a partial tour (or initially the depot), we first determine the set $F \subseteq V$ of feasible successor stations. Set F includes all stations that are not yet balanced and additionally can be serviced by the current vehicle l without exceeding the shift length $\hat{t}_l$, i.e., there is enough working time left to visit the station and to go back to the depot.

For each such candidate station $v \in F$, we calculate the maximum number of bikes that can be picked up or delivered by

$$\gamma_v = \begin{cases} \min(a_v - q_v, Z_l - b_l) & \text{for } v \in F \cap V_{\text{pic}} \text{ and} \\ \min(q_v - a_v, b_l) & \text{for } v \in F \cap V_{\text{del}}, \end{cases} \tag{3.4}$$

where b_l represents the final load of vehicle l so far and a_v the final number of bikes at station v in the currently considered partial tour. For an empty tour (i.e., $\rho_l = 0$) they are initialized with $b_l = 0$ and $a_v = p_v$, respectively. If routes for other vehicles have already been constructed, a_v is modified to correctly reflect the number of available bikes under consideration of the other vehicles' actions.

We assume that no bikes are allowed to remain on a vehicle when returning to the depot. Therefore, an additional correction is important for pickup stations. For this purpose, we determine an estimation of the number of bikes b^{del} which can still be delivered to successive stations after visiting the last station within the remaining time. This is achieved by a recursive call of the construction heuristic which only considers delivery stations and assumes to have an unlimited amount of bikes available at the vehicle.

Note that here we deviate in a detail from the greedy heuristic in [125]: In that work, the estimation of deliverable bikes is individually determined for each candidate station $v \in F \cap V_{\text{pic}}$ considering it as the starting point. Tests indicated that the higher precision gained by these individual calculations is relatively small while the computational effort is substantially higher by a factor of $O(|V|)$. Especially when considering the extension to the PILOT method in the next chapter and the larger instances with up to 700 stations used here, the differences in running time become dramatical, and thus, we rely on the described simpler approach.

Having determined b^{del}, we discard all remaining pickup stations from F if $a_v \geq b^{\text{del}}$, because in this case further pickups appear to be not possible anymore; i.e., the construction of the route is finished with delivery stations only. Otherwise, e.g., if further pickups are allowed, the number of bikes to be collected at each candidate pickup station $v \in F \cap V_{\text{pic}}$ is corrected by considering the limit b^{del}:

$$\gamma_v \leftarrow \min(\gamma_v, b^{\text{del}} - b_l) \qquad \forall v \in F \cap V_{\text{pic}}. \tag{3.5}$$

Having calculated γ_v for all candidate stations $v \in F$, we finally evaluate them by the ratio $\gamma_v / t_{u,v}$, where $t_{u,v}$ is the time needed to travel from the vehicle's last location u to station v and service v. Thus, this greedy evaluation criterion considers the balance increase per time unit. The node $v \in F$ with the highest ratio is then appended to the tour r_l; ties are broken randomly. Loading instructions are set as follows:

$$y_{l,v}^{\rho_l} = \begin{cases} \gamma_v & \text{if } v \in V_{\text{pic}} \text{ and} \\ -\gamma_v & \text{if } v \in V_{\text{del}}. \end{cases} \tag{3.6}$$

Furthermore, b_l and a_v are updated accordingly and the procedure continues with the next extension, evaluating stations in F from scratch, until no feasible extension remains, i.e., $F = \emptyset$.

As b^{del} is only an estimation, it may occasionally happen that a few bikes remain in the vehicle at the end of a route. As we do not allow this in feasible solutions, we repair the situation by reducing the last pickup(s) correspondingly. If some y_l^i, $i = 1, \ldots, \rho_l$, becomes zero, then we remove visit i from the route.

PILOT Construction Heuristic

The PILOT construction heuristic extends the greedy construction heuristic using the PILOT (Preferred Iterative LOok ahead Technique) method according to [157]. On several occasions, this metaheuristic has already shown to yield better solutions than its simple greedy counterpart with only moderate and scalable computational overhead. In particular, we consider it to be a promising alternative to the VNS/VND approach for large instances where the VND might already take very long in execution. The basic idea of this method is to look ahead in order to escape the greedy trap, i.e., to further evaluate every candidate successor in a greedy way and thus avoid short-sighted results. The main issue of the greedy construction heuristic is that it always chooses the single locally best successor as long as the solution remains feasible. As a result, e.g., a dense cluster of stations which is in a greater distance from the current station than an isolated single station might yield a larger balance gain altogether, but the simple greedy algorithm does not recognize the cluster's overall benefit and selects the isolated station as successor. Contrarily, the PILOT construction heuristic evaluates each candidate station not just by its own distance and balance gain, but instead also in possible future gains by visiting further stations in corresponding recursive calls. To some degree, the PILOT approach is also related to probing techniques in Mixed Integer Programming [137].

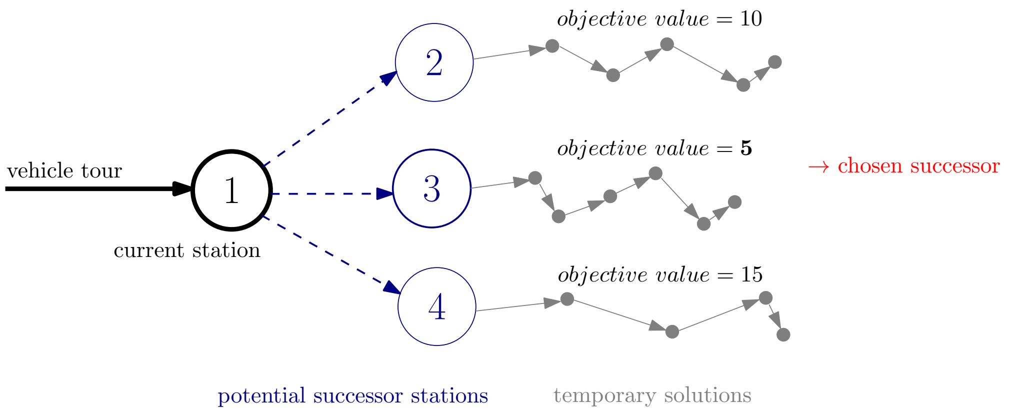
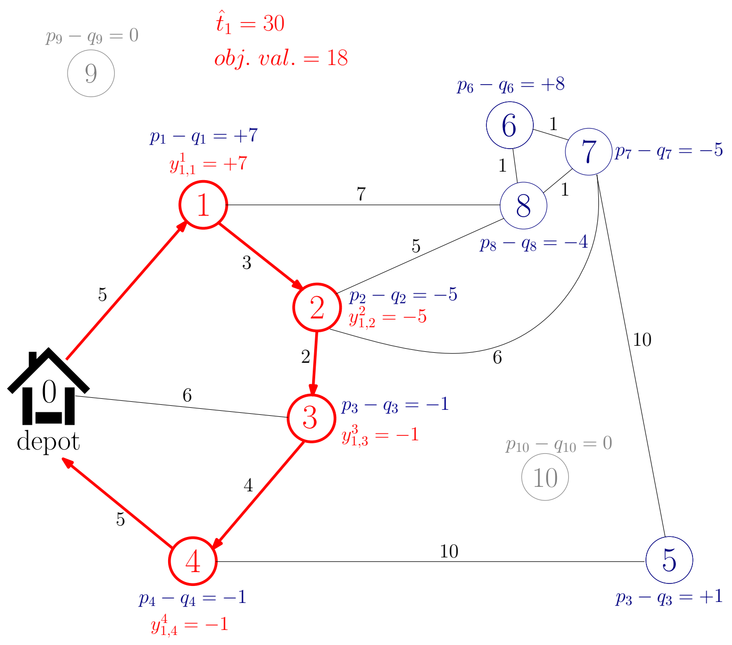


Figure 3.3: Basic principle of one iteration of the PILOT method for evaluating stations.

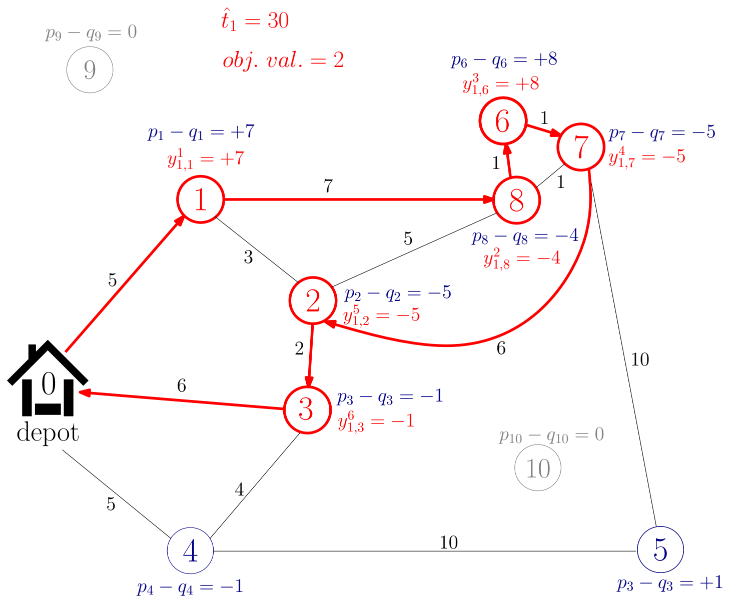
Figure 3.3 shows the basic idea of PILOT in the context of BBSS. The vehicle is currently at station 1 and we evaluate all potential successors by greedily determining individual extensions with them. In this example we only show the evaluation for the stations $\{2, 3, 4\}$. It is performed by trying to temporarily append each candidate station to the current route and continuing the basic greedy construction process until no further station can be added. Furthermore, the constructed extensions are evaluated on a defined criterion which is in our case the total decrease of the objective function value (3.3). Finally, the candidate station with the highest benefit (i.e., objective function decrease) is selected – in our example station 3 – and appended to the route; all temporary solutions are discarded and PILOT continues with the next round of successor evaluations until F becomes empty and the route is completed.

Note that the construction of the temporary extensions is done exactly the same way as in the basic greedy construction heuristic, including the calculation of the number of bikes to be picked up or delivered, and taking into account the estimation of the number of bikes that can still be delivered.

Figure 3.4 shows an example how the PILOT approach dominates the simple greedy variant where the shift length is assumed to be $\hat{t}_1 = 30$min. For simplicity we only show the most lucrative connections and assume symmetric traveling times which are printed for each edge. The objective function values only show the imbalance and omit the other factors (working time and total number of loading instructions), in order to simplify the visualization. Figure 3.4a visualizes the solution of the greedy construction heuristic. Note that in particular the path from station 1 to 2 has a higher greedy value ($\frac{5}{3} = 1.67$) than to station 8 ($\frac{4}{7} = 0.57$), and again the path from station 2 to 3 is preferred over station 8. After the visit of station 4 no further feasible station is left. On the contrary, the PILOT method will select station 8 as second one because when considering it, the most lucrative extension with further stops at the stations 6, 7, 8, 2, and 3 is identified.



(a) Greedy construction heuristic.



(b) PILOT method.

Figure 3.4: Exemplary solutions of the greedy construction heuristic and the PILOT method with one vehicle and and $\hat{t}_1 = 30$ min showing the benefits of the latter.

Due to the recursive evaluation of candidates the time complexity of the PILOT approach is higher than the time complexity of the basic greedy heuristic by a factor of $O(|V|)$. One possibility to improve the running time while still following the general idea is to apply a short-cut policy, i.e., to limit the recursive look-ahead to a certain number of successor stations, which is referred to as the *PILOT depth* β. In such a limited-depth PILOT approach, we do not evaluate each candidate extension by the overall gain in the objective function since the required time becomes a crucial factor again. Instead, we follow the criterion of the greedy heuristic, i.e., use the ratio of the balance gain and the time for the whole extension.

We tested our PILOT extension with various restricted depths and the unrestricted case on our benchmark instances, which are introduced in more detail in Section 3.3.7. Figure 3.5 shows the objective values and computation times for varying β on benchmark instances including 700 and 90 stations, where $\beta = 0$ represents the simple greedy approach and $\beta = \infty$ the unrestricted depth. Since the unrestricted case still runs very fast compared to our other metaheuristics and yields significantly better results than when imposing any depth limit, we finally decided to only consider the unrestricted case in all further work.

3.3.3 Solution Representation and Deriving Loading Instructions

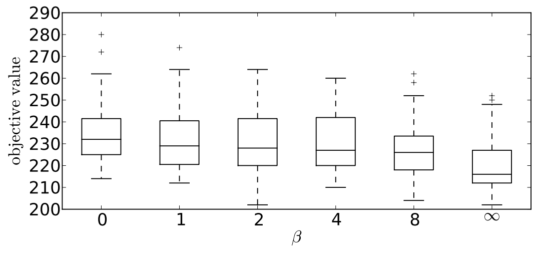
Our VND, GRASP, and VNS metaheuristics will be described in detail in Sections 3.3.4 to 3.3.6 and use an incomplete solution representation inspired by [20]. They process the search space of vehicle routes, while corresponding loading instructions $y^i_{l,v},\ l \in L,\ v \in V,\ i = 1, \ldots, \rho_l$, are derived for each candidate solution by an embedded procedure. We consider four alternative methods for calculating loading instructions for a given set of routes r. The next sections describe them and examine their individual assets and drawbacks.

Greedy Heuristic (GH)

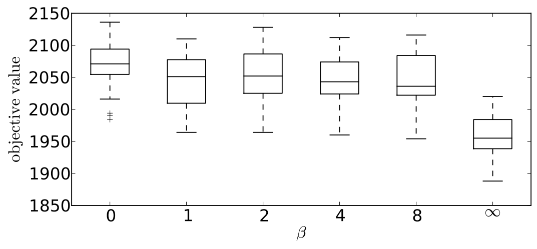
This fast heuristic approach follows the strategy from the greedy construction heuristic for a whole solution. It processes the routes vehicle by vehicle in a sequential way. Stations are considered in the order as they are visited and loading instructions are computed in a similar way as described in Section 3.3.2. If the current station $v = r^i_l$, $l = 1, \ldots, |L|$, $i = 1, \ldots, \rho_l$, is a delivery station, then

$$y^i_{l,v} = -\min(q_v - a_v, b_l) \qquad (v \in V_{\mathrm{del}}), \tag{3.7}$$

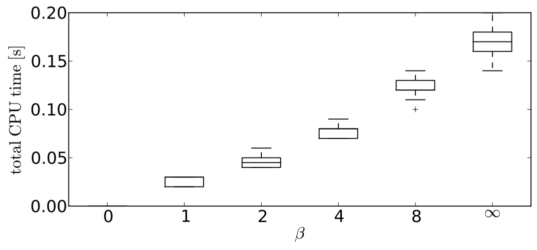
with a_v indicating the current number of bikes at station v and b_l the number of currently loaded bikes at vehicle l. In case of v being a pickup station, an estimation b^{del} of the number of bikes which can still be delivered is calculated, but now on the basis of the



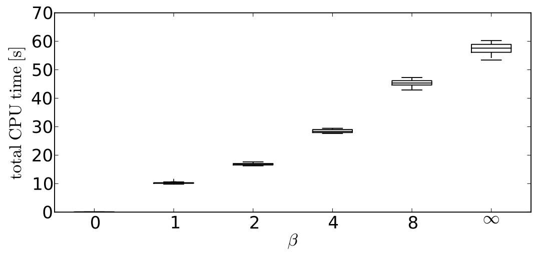
(a) Objective values for instances with $|V| = 90$, $|L| = 2$, and $\hat{t}_l = 8$ h, $\forall l \in L$.



(b) Objective values for instances with $|V| = 700$, $|L| = 14$, and $\hat{t}_l = 8$ h, $\forall l \in L$.



(c) CPU times for instances with $|V| = 90$, $|L| = 2$, and $\hat{t}_l = 8$ h, $\forall l \in L$.



(d) CPU times for instances with $|V| = 700$, $|L| = 14$, and $\hat{t}_l = 8$ h, $\forall l \in L$.

Figure 3.5: Finally best objective values and CPU times in seconds for different PILOT depths β.

already known successive delivery stations in the route. Loading instructions are then set to

$$y^i_{l,v} = \min(a_v - q_v, Z_l - b_l, b^{\text{del}} - b_l) \qquad (v \in V_{\text{pic}}). \tag{3.8}$$

GH is able to calculate loading instructions very quickly, but it is, as the construction heuristic, restricted to the monotonic case of the BBSS problem, i.e., does not make use of temporarily buffering bikes at stations or exchanging of bikes among vehicles. However, also under the assumption of monotonicity, GH is not guaranteed to find optimal loading instructions. For example, in a route where a station v is visited twice, it can be beneficial to retain bikes in the vehicle at the first visit of v in order to be able to satisfy a following delivery station. Station v may later be also satisfied on its second visit.

Maximum Flow Approach for the Monotonic Case (MF-MC)

The MF-MC approach assumes monotonicity like GH, but it is an exact method, i.e., it always derives proven optimal loading instructions for a given set of routes. We apply a maximum flow computation on a specifically defined flow network. The approach is similar to [20], but we extend this method to our problem definition by considering multiple vehicles and handling balance as a goal in the objective function instead of a hard constraint. The design of the flow network implicitly enforces all constraints of the BBSS problem with regard to the number of bikes present in the stations and vehicles.

We define the graph $G_{\text{fm}} = (V_{\text{fm}}, A_{\text{fm}})$ with node set $V_{\text{fm}} = \{\sigma, \tau\} \cup V_{\text{pic}} \cup V_{\text{del}} \cup V_L$, where σ and τ are the source and target nodes of the flow network, respectively, and $V_L = \bigcup_{l \in L} V_l$ with $V_l = \{v^i_l \mid l \in L,\ i = 1 \ldots, \rho_l\}$ represents the stops of all routes. The arc set $A_{\text{fm}} = A_\sigma \cup A_L \cup A_{\text{pic}} \cup A_{\text{del}} \cup A_\tau$ consists of:

- $A_\sigma = \{(\sigma, v) \mid v \in V_{\text{pic}}\}$ with capacities $p_v - q_v$ representing the surplus number of bikes at each pickup station.
- $A_\tau = \{(v, \tau) \mid v \in V_{\text{del}}\}$ with capacities $q_v - p_v$ representing the lacking number of bikes at each delivery station.
- $A_{\text{pic}} = \{(v, v^i_l) \mid v^i_l \in V_L,\ v = r^i_l,\ v \in V_{\text{pic}}\}$, i.e., each pickup node in V_{pic} is connected with every node representing a stop at this station in any route $l \in L$. These arcs' capacities are not limited.
- $A_{\text{del}} = \{(v^i_l, v) \mid v^i_l \in V_L,\ v = r^i_l,\ v \in V_{\text{del}}\}$, i.e., each node representing a stop at a delivery station is connected to the corresponding delivery node in V_{del}. These arcs' capacities are also not limited.
- $A_L = \{(v^{i-1}_l, v^i_l) \mid v^i_l \in V_L,\ i > 1\}$, i.e., the nodes representing the stops in each tour are connected according to the tour. Arc capacities are given by the vehicle capacities Z_l.

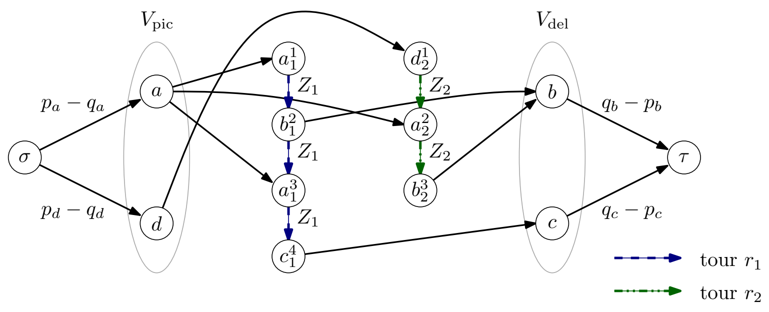


Figure 3.6: Exemplary flow network under the assumption of monotonicity for the tours $r_1 = (a, b, a, c)$ and $r_2 = (d, a, b)$ with $V_{\text{pic}} = \{a, d\}$ and $V_{\text{del}} = \{b, c\}$.

An exemplary network for an instance with four stations and two vehicles is shown in Figure 3.6. It can be seen easily that calculating a maximum (σ, τ)-flow on the network directly yields (under the assumption of monotonicity) optimal loading instructions y_l^i via the flows on the corresponding arcs A_{pic} and A_{del}, respectively. In our implementation, we use the efficient push-relabel method from Cherkassky and Goldberg [25] for the maximum flow computations.

Linear Programming Approach (LP)

In the more powerful LP approach we are able to determine optimal loading instructions for the general, not necessarily monotonic case by solving a minimum cost flow problem on another network by linear programming (e.g., the network simplex algorithm). The main difference is that we now consider the order in which vehicles make their stops (at possibly the same stations). In this model, bikes can be buffered at stations or even be directly transferred from one vehicle to another when they meet.

Let $t(r_l^i)$ denote the absolute time when vehicle l makes its i-th stop at station r_l^i. We define the multi-graph $G_{\text{f}} = (V_{\text{f}}, A_{\text{f}})$ with node set $V_{\text{f}} = \{\sigma, \tau\} \cup V_t$ where $V_t = \{v^j \mid \exists v_l^i \in V_l : t(r_l^i) = j\}$, i.e., besides source and target nodes σ and τ we have a node v^j for each station v and time j when a vehicle arrives at v. Furthermore, let $V^{\text{first}} = \{v^{j_{\min}} \in V_t \mid j_{\min} = \min\{j \mid v^j \in V_t\}\}$ denote the set of nodes representing the first visit of all stations among all routes and $V^{\text{last}} = \{v^{j_{\max}} \in V_t \mid j_{\max} = \max\{j \mid v^j \in V_t\}\}$, denote the set of nodes representing the last visit of all stations. Arc set $A_{\text{f}} = A_\sigma \cup A_\tau \cup A_R \cup A_V$ consists of:

- $A_\sigma = \{(\sigma, v^j) \mid v^j \in V^{\text{first}}\}$ with capacities p_v.
- $A_\tau = \{(v^j, \tau) \mid v^j \in V^{\text{last}}\}$ with capacities q_v.
- $A_R = \bigcup_{l \in L} A_{R,l}$ with $A_{R,l} = \{(u^{j'}, v^j) \mid u = r_l^{i-1},\ v = r_l^i,\ j' = t(r_l^{i-1}),\ j = t(r_l^i),\ i = 2, \ldots, \rho_l\}$, $\forall l \in L$, i.e., the arcs representing the flow induced by the vehicles. Capacities are Z_l. Note that multiple arcs exist between two nodes if two (or more) vehicles leave and arrive at the same stations exactly at the same time.

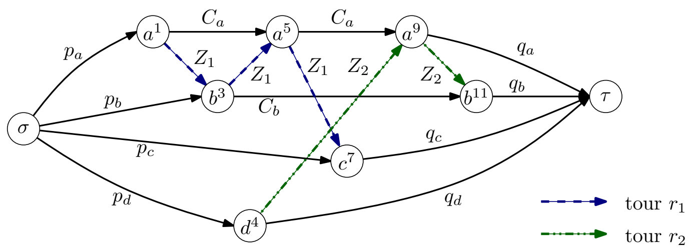


Figure 3.7: Exemplary flow network for the general case with tours $r_1 = (a, b, a, c)$ and $r_2 = (d, a, b)$.

- $A_V = \bigcup_{v \in V} A_v$ with $A_v = \{(v^{j_1}, v^{j_2}), \ldots, (v^{j_{\max-1}}, v^{j_{\max}})\}$, $(v^{j_1}, \ldots, v^{j_{\max}})$ is the sequence of nodes $\{v^j \in V_t\}$ sorted according to increasing j. Capacities are C_v.

An example of such a network is given in Figure 3.7. Now, a simple maximum (σ, τ)-flow calculation would in general not yield optimal or even feasible loading instructions anymore as it must be guaranteed that all arcs A_σ are satisfied (corresponding to the initially available bikes) and we do not have a correspondence between the achieved balance and the total flow. Instead, we have to minimize a certain objective function that depends on the flow, i.e., we have to solve the following minimum cost flow problem which is done by linear programming. Let the flow variables be $f_{u,v}$, $\forall (u, v) \in A_\mathrm{f}$. By $pred_l(v^j) \in V_t$ we denote the predecessor of the node v^j on the route of vehicle l, i.e., $pred_l(v^j) = u^{j'}$ with $u = v_l^{i-1}$, $j' = t(r_l^{i-1})$, and by $succ_l(v^j) \in V_t$ we denote its successor, i.e., $succ_l(v^j) = w^{j''}$ with $w = v_l^{i+1}$, $j'' = t(r_l^{i+1})$. To calculate the balance as final absolute deviations of the target values and the total amount of loading operations, we split the variables for the loading instructions $y_{l,v}^i \in \{-Z_l, \ldots, Z_l\}$ into $y_{l,v}^{+,i} \in \{0, \ldots, Z_l\}$ for pickups and $y_{l,v}^{-,i} \in \{0, \ldots, Z_l\}$ for deliveries of bikes, i.e., $y_{l,v}^i = y_{l,v}^{+,i} - y_{l,v}^{-,i}$, $\quad y_{l,v}^{+,i} = 0 \ \vee \ y_{l,v}^{-,i} = 0$, and $|y_{l,v}^i| = y_{l,v}^{+,i} + y_{l,v}^{-,i}$.

$$\min \quad \omega^{\mathrm{bal}} \sum_{\forall v \in V^{\mathrm{last}}} \delta_v + \omega^{\mathrm{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} \left(y_{l,r_l^i}^{+,i} + y_{l,r_l^i}^{-,i} \right) \tag{3.9}$$

subject to

$$\sum_{(u,v^j) \in A_\sigma \cup A_V} f_{u,v^j} + \sum_{l \in L} \sum_{(u,v^j) \in A_{R,l}} f_{u,v^j} = \sum_{(v^j,w) \in A_\tau \cup A_V} f_{v^j,w} + \sum_{l \in L} \sum_{(v^j,w) \in A_{R,l}} f_{v^j,w} \qquad \forall v^j \in V_\mathrm{t} \tag{3.10}$$

$$y^{+,i}_{l,v} - y^{-,i}_{l,v} = \begin{cases} f_{v^j,succ_l(v^j)} & \forall l \in L,\ i = 1,\ v = r^i_l,\ j = t(r^i_l) \\ f_{v^j,succ_l(v^j)} - f_{pred_l(v^j),v^j} & \\ & \forall l \in L, i = 2,\dots,\rho_l - 1, v = r^i_l, j = t(r^i_l) \\ -f_{pred_l(v^j),v^j} & \forall l \in L,\ i = \rho_l,\ v = r^i_l,\ j = t(r^i_l) \end{cases} \tag{3.11}$$

$$f_{\sigma,v^j} = p_v \qquad \forall (\sigma, v^j) \in A_\sigma \tag{3.12}$$

$$f_{v^j,\tau} - q_v \le \delta_v \qquad \forall (v^j, \tau) \in A_\tau \tag{3.13}$$

$$q_v - f_{v^j,\tau} \le \delta_v \qquad \forall (v^j, \tau) \in A_\tau \tag{3.14}$$

$$0 \le f_{v^j,\tau} \le C_v \qquad \forall (v^j, \tau) \in A_\tau \tag{3.15}$$

$$0 \le f_{u^{j'},v^j} \le Z_l \qquad \forall l \in L,\ (u^{j'}, v^j) \in A_{R,l} \tag{3.16}$$

$$0 \le f_{v^{j'},v^j} \le C_v \qquad \forall (v^{j'}, v^j) \in A_V \tag{3.17}$$

$$\delta_v \ge 0 \qquad \forall (v^j, \tau) \in A_\tau \tag{3.18}$$

$$y^{+,i}_{l,v} \in \{0,\dots,Z_l\} \qquad \forall l \in L,\ v \in V,\ i = 1,\dots,\rho_l \tag{3.19}$$

$$y^{-,i}_{l,v} \in \{0,\dots,Z_l\} \qquad \forall l \in L,\ v \in V,\ i = 1,\dots,\rho_l \tag{3.20}$$

The objective function (3.9) is directly derived from our BBSS objective (3.3). Equations (3.10) are the flow conservation equalities, while equations (3.11) link the loading instruction variables with the flows. The flows at arcs $(\sigma, v^j) \in A_\sigma$ are fixed to the station's initial number of bikes p_v in (3.12).

As we have a capacitated but unrestricted flow network with all capacities being integer, the LP is totally unimodular and the corresponding polytope's extreme points are all integer. Therefore by solving this LP with a common LP solver (or more specifically a network simplex algorithm), we obtain optimal integral values for the loading instructions.

Maximum Flow Approach for the General Case (MF-GC)

Since solving the above minimum cost flow problem on G_f by linear programming is computationally expensive, we developed an alternative approach for obtaining the same optimal loading instructions based on two maximum flow calculations and an additional post-processing step; details of this rather complex procedure can be found in [122].

Although this approach, which we call here MF-GC, is computationally significantly more efficient than LP, it is still slower than MF-MC and especially GH. In preliminary results we observed that similar to the LP approach, the additional computational effort for allowing the solution to overcome the monotonicity restriction does not pay off in most cases. In this article we omit a detailed description but will include comparative results in Section 3.3.7.

In [122], we further evaluated a hybrid approach, in which the different strategies for calculating loading instructions are applied in a combined, adaptive way. In the VND, an additional neighborhood structure is used to determine the best suited method for a solution, and this method is inherited by its descendants. Results on instances with up to 90 nodes indicated small advantages for this approach. However, the benefits diminish for larger instances as considered in the current work, and running times become again considerably larger. Thus, we do not further consider the combination in this article.

3.3.4 Variable Neighborhood Descent (VND)

For locally improving candidate solutions, we employ several classical neighborhood structures that were successfully applied in various VRPs together with new structures exploiting specifics of BBSS within a Variable Neighborhood Descent [97]. All these neighborhood structures augment each other. Concerning the classical neighborhood structures, we based our design on the experience from [112].

The following neighborhoods are traversed in a best improvement fashion and applied in the given static order that has been determined experimentally. We also tried to use a dynamic reordering strategy but it did not yield any significant advantages.

After each move inside a neighborhood, all candidate tours that have changed are efficiently checked for feasibility with respect to time budgets using incremental computations. If one station appears multiple times in direct succession within a route, only the first stop is retained. Infeasible solutions, i.e., solutions where at least one vehicle route became infeasible, are discarded immediately. For solutions where all routes stay feasible we derive loading instructions by one of the methods from the last section. Obsolete stops without any loading or unloading operations, i.e., where $y^i_l = 0$, are immediately removed from the routes.

Remove station (REM-VND): This neighborhood considers all possible removals of a single station in each route. Thus, a successful move avoids an unnecessary visit of a station: In this case, the same overall balance can be obtained without the visit, resulting in a shorter total working time and a higher potential to include some other station.

Insert unbalanced station (INS-U): This neighborhood tries to improve balance by considering each single yet unbalanced station for insertion at any position of any route.

Intra-route 2-opt (2-OPT): This is the classical 2-opt neighborhood from the traveling salesman problem applied individually to each route. Each possible segment of at least two stations is tried for inversion.

Replace station (REPL): Similarly to INS-U, this neighborhood considers stations which are currently unbalanced. However, it considers the replacement of an existing station by another unbalanced station.

Intra or-opt (OR-OPT): Here we consider solutions where sequences of one, two, or three stations are moved to another position within the same route.

2-opt* inter-route exchange (2-OPT*): This classical neighborhood of vehicle routing problems considers pairs of routes. All feasible exchanges of arbitrarily long end segments of the routes are enumerated. The neighborhood is implemented efficiently such that if an exchange of an end segment already resulted in an infeasible route, no end segments of larger length will be considered for moving to the route which became infeasible.

Intra-route 3-opt (3-OPT): This neighborhood structure resembles a restricted variation of the well-known 3-opt neighborhood, individually applied to each route. For any partitioning of a route into three nonempty subsequences r_l =(a,b,c), the routes (b,a,c) and (a,c,b) are considered. An effective enumeration scheme excludes all solutions of the previous neighborhoods.

3.3.5 Greedy Randomized Adaptive Search Procedure (GRASP)

In order to prolong the heuristic search and obtain potentially better solutions, we extend our two construction heuristics to *Greedy Randomized Adaptive Search Procedures* (GRASP) according to [128]. For this purpose we iteratively apply a randomized version of the greedy or PILOT construction heuristic, respectively, and locally improve each solution with the VND. The overall best solution is returned.

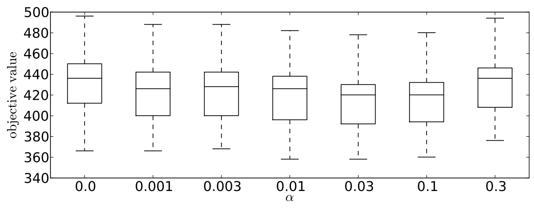
The construction heuristics are randomized in order to obtain a diversified set of starting solutions for the VND. This randomization takes place in a GRASP-typical way: At each iteration of the construction heuristic, we do not always pick the locally best successor station but rank all candidates from F (the serviceable, not yet balanced stations) according to the heuristic evaluation criterion. A restricted candidate list $RCL \subseteq F$ of best successors is preselected, and from these, one station is chosen uniformly at random. This successor is appended to the route, and the construction heuristic continues with its next iteration.

More precisely, the restricted candidate list RCL contains the following elements:

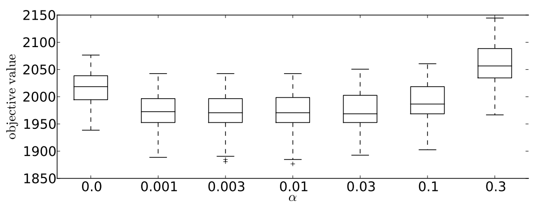
$$RCL = \{v \in F \mid g(v) \geq g_{\max} - \alpha\,(g_{\max} - g_{\min})\}, \tag{3.21}$$

where $g(v)$ is the greedy value of candidate station v, while $g_{\max} = \max\{g(v) \mid v \in F\}$ and $g_{\min} = \min\{g(v) \mid v \in F\}$ are the maximum and minimum evaluation values that occur in F, respectively. Parameter $\alpha \in [0,1]$ controls the strength of the randomization, with $\alpha = 0$ resulting in a purely greedy solution, while $\alpha = 1$ turns the heuristic into a completely random construction method.

In preliminary tests on the benchmarks instances described in Section 3.3.7, we evaluated different values for α in a fixed and a randomized version. In the fixed version α remains constant throughout all GRASP iterations whereas in the randomized variant we choose



(a) Instances with $|V| = 180$, $|L| = 4$.



(b) Instances with $|V| = 700$, $|L| = 14$.

Figure 3.8: GCH-GRASP: Final objective values in dependence of α; $\hat{t}_l = 8\text{h},\ \forall l \in L$.

an individual α' randomly from $[0, \alpha]$ for each GRASP-iteration. The randomized version turned out to be significantly more robust, and consequently we employ it in all further tests.

Moreover, our tests indicated that large instances w.r.t. $|V|$ tend to require smaller values for α than small instances. Figure 3.8 shows the impact of different values for α exemplarily for GCH-based GRASP on instances with $|V| = 180$, $|L| = 4$ and instances with $|V| = 700$, $|L| = 14$, and $\hat{t}_l = 8\text{h},\ \forall l \in L$. One can see that differences are generally relatively small, indicating the robustness of the method w.r.t. the choice of α. Based on those preliminary tests we finally concluded to set

$$\alpha = 0.1 \cdot e^{-\frac{|V|}{200}} \tag{3.22}$$

in all following tests.

3.3.6 General Variable Neighborhood Search (VNS)

As an alternative to GRASP for diversification of the search, we embed the previously described VND into a Variable Neighborhood Search (VNS) as described in [97]. Similarly to the VND, the VNS neighborhood structures represent a combination of both classical neighborhoods from vehicle routing problems and more problem-specific neighborhoods of the BBSS problem.

In contrast to the VND, the VNS neighborhoods are generally larger and thus, they are not systematically searched, but instead used for *shaking*, i.e., for randomly deriving single new solutions in some distance to the incumbent solution. These solutions are always locally improved by the VND before the VNS decides upon their acceptance.

We use four types of VNS neighborhood structures, and each is parameterized by six different possible values of a parameter δ, yielding a total of 24 individual neighborhoods. If a VNS move results in an infeasible solution containing routes that violate the available time budget of a vehicle, the respective routes are repaired by removing stations from the end. If the VNS does not find a better solution with the last neighborhood, it restarts with the first, until a termination criterion (i.e., CPU time or a certain number of iterations without improvement) is fulfilled.

Move sequence (MV-SEQ$_\delta$): This neighborhood selects a sequence from one to $\min(\delta, \rho_l)$ consecutive stations of a source route $r_l,\ l \in L$ randomly, and moves it to a random position of a different route. If the original route contains less than δ stations, the whole route is inserted into the target route and the first route becomes empty. Both source and target routes are selected randomly, with the restriction that they must not be the same route. $\delta \in \{1, \dots, 5, \rho_l\}$.

Exchange sequence (EX-SEQ$_\delta$): This neighborhood also selects two routes at random. In each route, a randomly selected segment of length one to $\min(\delta, \rho_l)$ is chosen and exchanged with the respective other route. With a probability of 10% each exchanged segment is added to the target route in reversed order. This particular feature is adopted from [112]. $\delta \in \{1, \dots, 5, \rho_l\}$.

Remove stations (REM-VNS$_\delta$): Here we sequentially process all stops of all routes and remove each station visit with probability $\delta \in \{10\%, 14\%, 18\%, 22\%, 26\%, 30\%\}$. We trust on the VND to again add fruitful visits.

Destroy and recreate (D&R$_\delta$): In this neighborhood we select a random position in a randomly chosen route $r_l,\ l \in L$. Then, all nodes from this position to the end of the route are removed and a new end segment is created by applying a randomized version of the PILOT construction heuristic. The randomization is done as described in Section 3.3.5, but with the threshold parameter set to $\alpha = \delta \in \{0\%, 4\%, 8\%, 12\%, 16\%, 20\%\}$.

3.3.7 Computational Results

We performed extensive tests in order to assess the performance of our algorithms. After describing the way we generated our test instances, Section 3.3.7 compares the performance of VNS variants with different methods for deriving loading instructions and analyzes the efficiency of the VND neighborhood structures. We also use results from a sequence-indexed mixed integer programming (MIP) model as a baseline. For these preliminary analyses we only use small to medium-sized instances since a clear trend is visible and the MIP approach also reaches its limits. Finally, Section 3.3.7 compares the two construction heuristics with or without a subsequent VND run, as well as the GRASP and VNS approaches on all instances.

Benchmark Instances

We tested our algorithms on a new benchmark suite based on real-world data from the year 2011 provided by Citybike Wien which runs a bike-sharing system with, at the time, 92 stations in Vienna, Austria. In order to evaluate the performance of the approaches on very large instances, our project partner Austrian Institute of Technology (AIT)[1] provided a larger set of 664 additional stations placed in Vienna at realistic positions.

[1] http://www.ait.ac.at/

We generated the instances, which are available on the web[2], from the pool of 92 real plus 664 artificial stations as follows:

- Travel times $t_{u,v},\ (u,v) \in A_0$, are real average driving times from u to v plus an estimation for parking the vehicle and loading or unloading bikes at v.
- Station capacities C_v of artificial stations were chosen randomly according to the distribution of real stations' capacities.
- The number of initially available bikes p_v was taken from a snapshot of the system for all real stations. For the artificial ones, we first dedicated some of them as *support points* to which we assigned fill levels at random according to a certain distribution derived from the real stations. The fill levels of the remaining artificial stations were then determined by an Akima bicubic spline interpolation. In this way we achieve a small correlation between the fill levels of geographically close stations, as it can also be observed in the real data.
- Target values q_v were derived from accumulated demands of the stations over a whole day which are known for the real stations. For terminals with a high number of bike demands we set the target value to 75% of the stations capacity, for a high number of parking position demands we set the target value to 25% and, if both are similar, then we set the target value to 50% of the capacity. For the artificial stations accumulated demands were determined randomly in a similar way as initial fill levels, i.e., to achieve a similar distribution and geographic correlation as in the real data.
- We derived instances with different numbers of stations $|V|$ by choosing the first station randomly from the pool of 756 stations and adding its $|V|$ nearest neighbors with regard to Euclidean distances. From the now $|V|+1$ stations, one was randomly selected to be the depot.
- In order to make complete balance at least theoretically possible when having enough working time and vehicles available, $\sum_{v \in V} p_v = \sum_{v \in V} q_v$ must hold in each instance. This was achieved by randomly selecting a station v and incrementing or decrementing p_v by one, as required. We iterated this process until the above equation was fulfilled. Note that there might not necessarily be the right number of bikes available to meet all target levels in reality. Still, operators strive to fulfill target levels by controlling the number of bikes in the system accordingly.
- We assume a homogeneous fleet of vehicles with capacities $Z_l = 20,\ \forall l \in L$.
- The time budget was set to $\hat{t} \in \{2, 4, 8\}$ hours, $\hat{t}_l = \hat{t},\ \forall l \in L$.
- The number of available vehicles $|L|$ varies and is stated in the following sections.

[2]https://www.ac.tuwien.ac.at/files/resources/instances/bbss/bench3.tar.gz

- For each configuration of $|V|$, $|L|$ and $\hat{t}$, 30 independent instances were created.

For the real-world scenario at Citybike Wien the configuration with $|L| = 2$ and $\hat{t} = 8$ is the most relevant. The planning for Citybike Wien's two vehicles is done in the morning and they are usually in action for a whole working day shift of eight hours. Considering shorter shift times thus becomes interesting when performing an "on-line" re-optimization for the remaining day after some major disruption only. Note that the considered static case is not exactly met here, as Citybike Wien operates 24/7. Nevertheless solving the static BBSS is still considered to be a reasonably good approximation as expected demands are relatively well known and the target values are set correspondingly. Furthermore, stations are designed with a significant reserve so that the balancing is not usually needed for solving short-term fluctuations but to resolve imbalances occurring over a longer time horizon, e.g., one or even more days. Citybike Wien has its highest activity in the evening and during the night.

Note that this benchmark suite is different from the one used in our former work [125]. While in the latter instances initial fill levels of artificial stations have been chosen simply at random and all target values have been set to 50% of the stations' capacities, the new instances are more realistic.

We implemented all algorithms in C++ using GCC 4.6. Each test run was performed on a single core of an Intel Xeon E5540 machine with 2.53 GHz and 3 GB RAM per core. For solving the LP-based approach to determine loading instructions CPLEX 12.4 was used with default settings, except for restricting CPLEX to only use a single thread.

As already mentioned in Section 3.3.1, the scaling factors in the objective function were set to $\omega^{\text{bal}} = 1$, $\omega^{\text{load}} = \omega^{\text{work}} = 1/100\,000$. Using these factors, improving the system balance always has a greater impact on the objective value than reducing the tour lengths or the number of loading operations. This aspect is a result of our discussions with project partner Citybike Wien. Reducing the lengths of routes and/or the number of loading operations can lower operation costs and has a positive environmental impact. In addition, tours with obviously redundant station visits or loading actions would strongly reduce the practical acceptance by the drivers. However, the top priority is still to balance the system so that target values are reached as far as possible and user satisfaction is maximized. From the operator's point of view, workers are paid for the whole shift length anyway, and therefore a reduction in the tour lengths is just a secondary aspect.

Objective values of different solutions must be compared with care due to the small scaling factors ω^{load} and ω^{work} for the secondary terms in the objective function. If two solutions achieve the same balance but differ regarding the secondary terms, the difference between the objective values will be very small although possibly important. Therefore, we list in the result tables for each algorithm variant and each instance set the number of runs for which the variant yielded the best results (#best) besides average objective values. We consider #best to be a better indicator for analyzing performance differences than average objective value differences. Maximum #best values are printed bold for each instance set.

Comparison of VNS Variants

In this section we analyze the influence of the four alternative procedures for deriving loading instructions within the VNS in Section 3.3.7 and further compare them to a MIP approach. Moreover, we study the performance of the VND neighborhood structures.

The following instance settings are used in these tests:

- The number of stations is $|V| \in \{10, 20, 30, 60, 90\}$.
- The vehicle fleet size is $|L| \in \{1, 2, 3, 5\}$.
- Each instance set uses a unique combination of $|V|$, $|L|$, $\hat{t}$ and contains 30 instances, resulting in a total of 30 sets and 900 instances.
- For each instance five independent runs were performed.
- Each run was terminated when no improvement could be achieved within the last 5 000 VNS iterations or after a CPU time of one hour. In the first case we consider the heuristic search as converged, major further improvements in prolonged runs are unlikely.

Tables 3.3.7 and 3.3.7 show aggregated results for these 30 instance sets. They also cover both the situation where perfect balance cannot be achieved due to a small number of vehicles and/or insufficient time budgets in relation to the number of stations, and the situation where perfect balance is possible. Table 3.3.7 shows for all approaches the number of runs where the respective approach obtained the best objective values (#best). For the MIP model, we additionally give mean upper bounds $\overline{\mathrm{ub}}$, mean lower bounds $\overline{\mathrm{lb}}$ and their respective standard deviations $\mathrm{ub}_{\mathrm{sd}}$, $\mathrm{lb}_{\mathrm{sd}}$. For each variant of the VNS, in addition to #best, we also list mean objective values $\overline{\mathrm{obj}}$ and their standard deviations sd. Table 3.3.7 gives the median total run times $\widetilde{\mathrm{t_{tot}}}$ for all approaches. For the VNS variants we also list the median number of performed VNS iterations $\widetilde{\mathrm{g_{tot}}}$.

MIP models: In addition to the metaheuristic algorithms, we implemented a MIP model based on the sequence-indexed formulation from [126] but adapted to our problem formulation in a straight-forward way. Just like GH and MF-MC, this model is not able to consider dependencies among vehicles and is therefore restricted to monotonicity. CPLEX 12.4 was used for trying to solve the instances with this model. Again, we used default settings, except for the restriction to use only a single thread. A CPU time limit of one hour was imposed. Furthermore, we investigated a second MIP model based on a time-indexed formulation [126] for the general case. Unfortunately, experiments indicated that this approach led to worse results than the first model due to the higher complexity of the model caused by the discretization of station visit times. Thus, we omit the results of the time-indexed formulation here.

We can clearly observe that the pure MIP approach is only able to solve the smallest instances to optimality within the given time limit. For most realistically sized instances very large gaps remain between upper and lower bounds. The MIP approach scales very badly with increasing numbers of vehicles and especially with longer vehicle time budgets. This is reflected in the low #best values for all but the smallest instances. For the larger instances CPLEX often only found trivial solutions where all vehicles are staying at the depot, or even no solutions at all. Also, no useful lower bounds can be derived for these more complex instances.

For the few instance groups with $|V| = 10$ where the MIP approach yielded small gaps, we observe that the VNS variants are almost always able to find solutions with equal or better objective values. Especially on instance sets with more than 10 stations, the VNS dramatically outperforms the MIP approach by obtaining better solutions in substantially shorter run times.

Comparing different variants for deriving loading instructions: Among our four VNS variants, the one applying GH is clearly the fastest. MF-MC required about 120% more run time on average for each call of the auxillary algorithm for deriving loading instructions. LP is very slow, resulting in run times that are about 250 times longer than those of GH. MF-GC improves on the performance of LP, but is still around eight times slower than GH.

The solution quality of the VNS approach strongly depends on the ability to perform a substantial number of iterations. Therefore, a computationally more expensive variant needs to achieve rather large improvements in order to compete with the faster variants. We will now analyze the solution quality of the different variants for deriving loading instructions.

VNS with GH: As the fastest variant for deriving loading instructions, GH has a very high number of #best values (2969) that is only slightly exceeded by MF-MC. GH's total of the objective values is even better by a small degree. Especially on instance sets with large numbers of stations and shift lengths, this variant is able to achieve the highest #best values among all VNS variants. On small instances, GH performs slightly worse than MF-MC, but in general both variants obtain very similar results with regard to solution quality. In addition, GH has the advantage of significantly shorter run times.

VNS with MF-MC: This method shows the highest sum of #best values of all approaches (2987), as well as the second lowest total objective values. A Wilcoxon signed-rank test comparing MF-MC with the strongest contender GH on each instance set shows significant advantages with error probabilities of less than 5% for 18 of the 60 classes. We observe a clear tendency that MF-MC performs better on smaller instances while GH performs better on larger instances. There is no statistically significant difference between the two methods when looking at the total results of all 60 instance sets.

VNS with LP: In general the LP approach is able to construct better solutions than GH and MF-MC since it is not restricted to monotonicity but computes the optimal loading instructions for any given vehicle routes. However, this difference is barely visible in the results. We conclude that for the static BBSS problem the assumption of monotonicity does not have a practically significant negative impact on the solution quality. On the contrary, the LP approach suffers from the longest run times in comparison to the other variants and only a substantially smaller number of iterations can be performed within the time limit. About 60% of all runs were terminated before the VNS converged reasonably. This usually leads to worse objective values for LP in comparison to the other variants. When compared to GH, LP performs significantly worse when considering all 60 instance sets in total with an error probability of less than 0.1%. This is also reflected in the low sum of #best values for LP (2161).

VNS with MF-GC: Similarly to LP, MF-GC is theoretically able to construct better solutions than GH and MF-MC since it is not restricted to monotonicity. However, even with the vastly improved performance of MF-GC over LP, this difference is barely visible in the results. Again, higher computational expense of MF-GC compared to GH and MF-MC impedes the ability to find good solutions for larger instances because only a substantially smaller number of iterations can be performed within the time limit. While the #best values of MF-GC are quite good for small to medium instances, the faster methods outperform it for many instances with 60 and 90 stations, especially when the number of vehicles and/or shift length is large. A Wilcoxon test confirms this by showing that GH is significantly better than MF-GC for nine of the large instance sets and also in total over all instance sets with error probabilities of less than 1%. The total sum of #best values (2625) is about 10% lower than those of GH and MF-MC.

We conclude that it is more important to perform a large number of VNS iterations than to employ a more sophisticated but slower method for deriving loading instructions. While MF-GC and LP are theoretically more powerful than GH and MF-MC, they only infrequently lead to slightly better solution qualities and therefore cannot justify the higher computational effort. GH is by far the fastest method for deriving loading instructions, and no other method offers consistent significant advantages with regard to solution quality. Thus, GH is the best method for deriving loading instructions in practice.

Instance set			MIP					VNS with GH			VNS with MF-MC			VNS with LP			VNS with MF-GC		
$\|V\|$	$\|L\|$	$\hat{t}$ [h]	#best	$\overline{\text{ub}}$	ub_{sd}	$\overline{\text{lb}}$	lb_{sd}	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd
10	1	2	**150**	27.801434	14.211779	27.801434	14.211779	145	27.868101	14.302006	145	27.868101	14.302006	140	28.001431	14.321390	140	28.001431	14.321390
10	1	4	95	3.469482	4.690047	0.175360	0.513799	132	3.536151	4.711462	**142**	3.509487	4.691593	130	3.536152	4.711463	141	3.522819	4.701557
10	1	8	35	0.003216	0.000455	0.002599	0.000462	135	0.003195	0.000453	146	0.003194	0.000452	**150**	0.003193	0.000452	**150**	0.003193	0.000452
10	2	2	**150**	9.136065	8.894377	8.804364	9.173847	140	9.269374	8.815557	145	9.269374	8.815557	145	9.269374	8.815557	141	9.322702	8.789374
10	2	4	110	0.003405	0.000564	0.003262	0.000542	145	0.003386	0.000559	148	0.003385	0.000559	149	0.003385	0.000559	**150**	0.003385	0.000559
10	2	8	115	0.003236	0.000500	0.003151	0.000426	130	0.003195	0.000453	143	0.003193	0.000452	**145**	0.003193	0.000452	**145**	0.003193	0.000452
20	2	2	64	52.469726	15.575644	29.779824	17.605633	140	51.629729	15.678652	139	51.643060	15.664408	**146**	51.523067	15.732142	140	51.589728	15.669937
20	2	4	0	16.005675	9.195814	0.003873	0.000536	79	5.272503	5.147186	**101**	5.245838	5.111881	79	5.299168	5.145725	99	5.245838	5.111880
20	2	8	0	0.141535	0.720372	0.003499	0.000540	110	0.006378	0.000615	120	0.006376	0.000615	93	0.006380	0.000614	**126**	0.006376	0.000616
20	3	2	6	34.737654	12.331921	2.087799	4.842072	117	28.791113	12.655299	120	28.751115	12.622697	**124**	28.964448	13.143979	118	28.857786	12.835286
20	3	4	0	0.941061	2.301619	0.005357	0.001232	72	0.006701	0.000591	**106**	0.006696	0.000589	86	0.006700	0.000588	96	0.006697	0.000590
20	3	8	0	8.142002	31.790072	0.003441	0.000519	104	0.006379	0.000615	121	0.006377	0.000615	91	0.006381	0.000614	**126**	0.006376	0.000616
30	2	2	15	112.869799	21.740922	73.209360	21.776380	145	110.003155	22.309574	143	110.003157	22.309576	**147**	109.843162	22.000016	143	110.029821	22.279456
30	2	4	0	72.272515	24.260691	0.005688	0.000644	71	34.659678	10.415634	72	34.659674	10.551349	51	34.939667	10.467490	**75**	34.779667	10.574958
30	2	8	0	192.733679	38.103015	0.003746	0.002333	61	0.009488	0.000590	**95**	0.009483	0.000585	17	0.009505	0.000589	83	0.009486	0.000588
30	3	2	0	91.904456	16.821933	27.790735	15.884329	107	79.551281	18.292426	**111**	79.257939	17.935839	108	79.177952	17.822698	104	79.431270	18.026606
30	3	4	0	39.208437	19.988591	0.005442	0.000690	53	6.035519	3.506518	**60**	6.008853	3.589846	19	6.462163	3.657592	56	5.942195	3.525073
30	3	8	0	198.533333	26.842485	0.000000	0.000000	74	0.009490	0.000591	**82**	0.009487	0.000585	20	0.009514	0.000591	75	0.009488	0.000589
60	3	2	0	276.304578	30.436585	176.901880	32.916749	117	253.351513	29.879859	115	253.231519	30.422328	119	253.138190	30.559345	**120**	253.138184	30.205040
60	3	4	0	395.200000	41.452164	0.000000	0.000000	43	119.103159	17.025314	51	119.023163	16.828918	5	121.689781	16.831045	**61**	118.983163	16.889415
60	3	8	0	—	—	—	—	**88**	6.084625	2.916488	65	6.324594	2.882422	0	10.457516	4.111593	6	7.137850	2.980819
60	5	2	0	302.137768	60.288079	53.576799	33.369663	66	185.661202	25.467138	**77**	185.327872	25.074180	43	186.207850	25.117945	64	185.541202	25.174178
60	5	4	0	395.200000	41.452164	0.000000	0.000000	58	37.895268	8.183309	**81**	37.615265	7.922555	0	43.241742	8.411625	14	39.068543	8.039271
60	5	8	0	—	—	—	—	**80**	0.019482	0.000674	71	0.019477	0.000683	0	0.019829	0.000718	8	0.019543	0.000674
90	3	2	0	514.336170	69.373279	253.108777	171.334278	119	429.711772	45.932137	**122**	429.458444	45.947627	113	429.618440	46.091014	113	429.578445	45.746608
90	3	4	0	594.333333	48.336303	0.000000	0.000000	**62**	262.837077	36.385916	56	262.957075	36.569536	5	267.370337	36.036094	42	263.290408	36.582442
90	3	8	0	—	—	—	—	**104**	77.326039	16.216842	47	78.046021	16.003556	0	86.672450	15.847739	2	80.405974	16.121748
90	5	2	0	594.333333	48.336303	0.000000	0.000000	**77**	348.061580	46.531085	64	348.248250	46.695211	36	349.634889	46.266354	**77**	348.381573	46.632770
90	5	4	0	—	—	—	—	**81**	140.469627	24.803457	65	140.842943	25.123944	0	150.882756	25.744827	7	143.282910	25.304140
90	5	8	0	—	—	—	—	**114**	0.801714	1.265215	34	1.148350	1.597211	0	8.000637	3.952740	3	1.974977	2.185755
		Total	740	3932.221892	587.145678	653.276390	321.636453	2969	2217.987874	370.446215	2987	2218.507762	370.667375	2161	2263.999252	374.793550	2625	2227.574223	371.702839

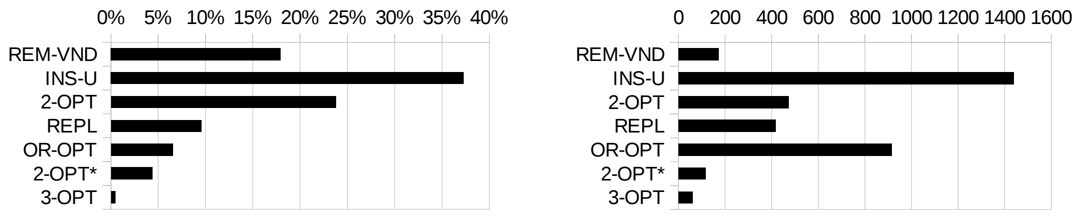
Table 3.1: Qualitative results of the MIP approach and the VNS considering the four variants of deriving loading instructions. Each instance set contains 30 instances, five runs per instance were performed.

Instance set			**MIP**	**VNS with GH**		**VNS with MF-MC**		**VNS with LP**		**VNS with MF-GC**	
$\|V\|$	$\|L\|$	$\hat{t}$ $[h]$	$\widetilde{t_{\mathrm{tot}}}$ $[s]$	$\widetilde{t_{\mathrm{tot}}}$ $[s]$	$\widetilde{g_{\mathrm{tot}}}$	$\widetilde{t_{\mathrm{tot}}}$ $[s]$	$\widetilde{g_{\mathrm{tot}}}$	$\widetilde{t_{\mathrm{tot}}}$ $[s]$	$\widetilde{g_{\mathrm{tot}}}$	$\widetilde{t_{\mathrm{tot}}}$ $[s]$	$\widetilde{g_{\mathrm{tot}}}$
10	1	2	3.0	0.8	5001.0	1.8	5001.0	224.0	5001.0	5.3	5001.0
10	1	4	3600.0	4.8	5003.0	12.1	5003.0	1382.7	5004.5	41.3	5002.0
10	1	8	3600.0	7.8	5001.0	19.2	5001.0	2056.6	5001.0	70.5	5001.0
10	2	2	897.5	1.8	5004.0	4.1	5008.0	488.9	5007.0	14.0	5004.5
10	2	4	3600.0	5.8	5009.5	12.9	5005.0	1467.7	5006.0	47.4	5006.5
10	2	8	3600.0	6.7	5001.0	16.7	5001.0	1795.9	5001.0	60.7	5001.0
20	2	2	3600.0	4.3	5012.0	10.5	5014.5	1346.9	5021.5	38.7	5020.0
20	2	4	3600.0	38.9	5670.0	95.3	5456.0	3601.0	2320.5	358.1	5374.0
20	2	8	3600.0	75.4	5271.0	193.4	5177.0	3601.3	1197.5	773.2	5163.5
20	3	2	3600.0	8.4	5103.0	19.3	5151.5	2017.0	5109.5	79.8	5137.0
20	3	4	3600.0	41.8	5372.5	101.9	5510.5	3600.8	2273.0	385.5	5361.0
20	3	8	3600.0	72.4	5400.0	188.6	5282.0	3601.5	1304.5	680.2	5258.0
30	2	2	3600.0	6.9	5032.0	17.6	5019.0	1848.8	5020.0	61.5	5025.5
30	2	4	3600.0	63.4	5866.0	158.6	5657.5	3601.2	1503.0	635.2	5780.0
30	2	8	3600.0	242.9	5806.0	695.3	5932.0	3603.4	423.5	2664.1	5684.5
30	3	2	3600.0	12.8	5072.5	29.7	5087.5	3000.9	5035.0	117.5	5095.5
30	3	4	3600.0	130.4	6813.0	316.9	6612.5	3602.3	938.0	1346.8	7015.5
30	3	8	3600.0	238.4	6446.5	615.6	5978.0	3603.4	476.5	2523.3	6155.0
60	3	2	3600.0	29.7	5129.5	74.5	5124.5	3600.9	2612.0	289.5	5136.5
60	3	4	3600.0	301.1	7046.5	878.1	7705.0	3605.9	393.0	3474.4	6441.5
60	3	8	3600.0	3009.4	10407.5	3600.4	4269.5	3632.1	59.0	3602.1	1022.0
60	5	2	3600.0	75.6	6115.0	185.3	5726.0	3601.4	1441.0	760.1	5757.5
60	5	4	3600.0	1115.9	10492.0	2703.7	10387.5	3613.7	170.0	3600.7	3126.0
60	5	8	3600.0	2826.7	9488.0	3600.4	4325.5	3643.6	52.0	3601.8	979.5
90	3	2	3600.0	46.7	5200.0	136.3	5184.0	3601.0	1803.5	474.1	5185.5
90	3	4	3600.0	490.8	7343.0	1506.6	7326.0	3608.7	239.0	3600.5	4629.5
90	3	8	3600.0	3600.2	6914.5	3600.9	2401.5	3666.5	34.0	3604.0	552.0
90	5	2	3600.0	126.5	5924.5	330.2	5772.5	3602.3	809.5	1412.4	5923.5
90	5	4	3600.0	1856.5	10899.5	3600.2	7425.5	3626.1	97.5	3601.7	1705.5
90	5	8	3600.0	3600.7	3164.0	3602.0	1077.5	3721.7	15.0	3608.1	236.0
		Total	101700.5	18043.5	185008.0	26328.1	162622.0	87968.2	68368.5	41532.5	136780.5

Table 3.2: Results regarding computation times and iteration counts of the MIP approach and the VNS considering the four variants of deriving loading instructions.

Up to this point we only used instances with up to 90 stations. However, it is already obvious that the more complex methods perform worse with increasing instance size. Since GH already turns out to be the best overall method on these instances, we refrained from testing even larger instances.

Performance of VND neighborhood structures: Figure 3.9 shows relative success rates (i.e., the number of times a particular neighborhood structure was able to improve the solution divided by the number of applications) and CPU times of the VND neighborhoods



(a) Relative success rates. (b) CPU times [s].

Figure 3.9: VND neighborhood performance for the instance set with $|V| = 90$, $|L| = 5$, $\hat{t} = 8\text{h}$ using the VNS with GH.

for runs of the VNS with GH on the large instance set with $|V| = 90$, $|L| = 5$, $\hat{t} = 8\text{h}$. However, these results are typical for all instance sets except the smallest ones.

REM-VND is applied as the first neighborhood despite having lower relative success rates than the two following neighborhoods because it is meant to provide space in a route for the insertion of additional stations. Experiments confirmed that it is advantageous to use REM-VND as the first neighborhood directly followed by INS-U.

In the VNS, all shaking neighborhoods have similar relative success rates, therefore we omit the corresponding chart. These results show that all neighborhood structures contribute well to the overall performance.

Comparing the GRASP/PILOT Hybrid with VNS

In this section we compare our GRASP/PILOT hybrid (see Section 3.3.5) with VNS with GH for deriving loading instructions. In order to analyze the different approaches in detail, we performed extensive tests across all instance sizes.

The following configurations were used in this section:

- The number of stations is $|V| \in \{30, 60, 90, 180, 300, 400, 500, 600, 700\}$.
- Meaningful numbers of vehicles $|L| \in \{1, \ldots, 21\}$ were chosen in dependence of $|V|$ and $\hat{t}_l$ and are listed in the result tables.
- Each instance set uses a unique combination of $|V|$, $|L|$, $\hat{t}$ and contains 30 instances, resulting in a total of 2310 instances grouped into 30 sets.

First of all, Table 3.3 compares the results of the basic greedy construction heuristic (GCH) and the PILOT method. It lists the number of runs where the respective approach obtained the best objective values (#best), the mean objective values $\overline{\text{obj}}$, and their median computation times $\widetilde{\text{t}_{\text{tot}}}$. According to a Wilcoxon signed-rank test with an error level of 5%, PILOT yields significantly better results on all aggregated results. This dominance can also be easily observed by looking at the objective values and #best.

Instance set			GCH				PILOT			
$\|V\|$	$\|L\|$	$\hat{t}$ [h]	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ [s]
30	1	2	8	152.001530	26.203390	< 0.1	**30**	147.934980	25.818900	0.0
30	1	4	1	105.336470	22.402790	< 0.1	**30**	98.536640	20.824000	0.0
30	1	8	0	38.939550	11.694270	< 0.1	**30**	33.206330	9.834630	0.0
60	1	4	2	283.403320	30.702870	< 0.1	**30**	274.536880	30.974760	0.0
60	1	8	0	189.473390	23.429750	< 0.1	**30**	181.006870	23.042860	0.1
60	2	2	3	309.069590	35.718110	< 0.1	**27**	299.203200	33.768400	0.0
90	2	4	1	374.206650	39.689670	< 0.1	**29**	358.273740	40.821710	0.1
90	2	8	0	204.479890	28.385090	< 0.1	**30**	190.146840	28.724430	0.4
90	4	2	0	423.739160	54.371890	< 0.1	**30**	397.606520	45.975640	0.0
180	4	4	0	772.613200	45.352750	< 0.1	**30**	737.747390	45.958780	0.5
180	4	8	0	427.826300	34.797430	< 0.1	**30**	395.827040	33.669300	2.9
180	5	8	0	317.565360	29.071670	< 0.1	**30**	285.432860	28.725460	3.4
300	6	4	0	1321.753520	56.947210	< 0.1	**30**	1271.554660	60.493890	2.0
300	6	8	0	782.906610	37.341860	< 0.1	**30**	736.841050	35.493270	11.8
300	9	8	0	462.657110	25.704740	< 0.1	**30**	413.858500	22.799850	15.4
400	8	4	0	1754.493320	75.600710	0.1	**30**	1681.628380	68.761200	4.4
400	8	8	0	1028.119890	47.776700	0.1	**30**	971.388000	41.910880	27.6
400	12	8	0	607.542700	35.657190	0.1	**30**	543.077910	32.626580	36.0
500	10	4	0	2205.566730	74.431490	0.1	**30**	2118.902180	70.022140	8.1
500	10	8	0	1301.733160	50.863390	0.1	**30**	1225.801700	44.761010	50.9
500	15	8	0	764.095080	34.038710	0.1	**30**	681.564200	29.478280	67.6
600	12	4	0	2712.906470	50.845290	0.1	**30**	2597.909120	40.370350	13.7
600	12	8	0	1608.079730	29.548010	0.2	**30**	1514.815390	27.303960	87.8
600	18	8	0	954.114180	31.063330	0.2	**30**	854.983800	26.979600	116.6
700	14	4	0	3115.779650	72.914380	0.2	**30**	2986.049480	62.149560	21.1
700	14	8	0	1866.959250	51.772820	0.2	**30**	1755.028720	41.214960	132.8
700	21	8	0	1104.599490	43.605020	0.2	**30**	984.469630	34.184230	178.5
		Total	15	25189.961300	—	1.7	806	23737.332010	—	781.7

Table 3.3: Computational results of the greedy construction heuristic (GCH) and the PILOT method.

Although PILOT requires more computation time, even on the largest instances the additional effort does not get out of hand, and we consider it to be a good tradeoff. Consequently, for practical applications that require solutions in short time, the PILOT heuristic is a good choice.

In order to analyze the improvement potentials of the solutions obtained by the construction heuristics, we add a local improvement step via VND at the end and compare the results in Table 3.4. First we observe that PILOT with VND performs significantly better than GCH with VND, which is supported by the Wilcoxon signed-rank test with an error probability of less than 5%. The total run times of both approaches are very close now. The reason is that although GCH is much faster than PILOT, VND has to spend more iterations to reach a local optimum afterwards which evens up at the end. During these tests, PILOT is the clear winner.

Instance set			**GCH-VND**				**PILOT-VND**			
$\|V\|$	$\|L\|$	$\hat{t}$ $[h]$	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ $[s]$	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ $[s]$
30	1	2	22	148.601630	25.806420	< 0.1	**25**	147.801650	25.764711	0.0
30	1	4	14	97.869980	20.537200	< 0.1	**24**	97.736620	20.649725	0.0
30	1	8	9	32.339640	10.280110	< 0.1	**22**	31.339650	9.661051	0.1
60	1	4	10	276.270150	29.556540	< 0.1	**22**	273.670200	30.617129	0.0
60	1	8	**17**	178.406880	22.605600	< 0.1	14	178.740170	22.806587	0.1
60	2	2	12	299.936470	32.798160	< 0.1	**24**	297.736530	33.968484	0.0
90	2	4	6	361.007000	39.099410	< 0.1	**25**	356.607050	40.592714	0.1
90	2	8	6	189.546730	29.401890	0.3	**24**	185.013490	28.761965	0.6
90	4	2	4	404.472980	46.165080	< 0.1	**26**	396.339860	46.010674	0.0
180	4	4	2	749.213760	46.891200	0.2	**28**	735.547360	45.351396	0.6
180	4	8	5	400.026790	35.451870	1.9	**25**	392.560250	32.355673	3.8
180	5	8	5	289.299210	28.640580	3.8	**25**	281.632680	28.291244	5.4
300	6	4	3	1282.421030	59.696840	1.0	**27**	1265.487980	60.006868	2.4
300	6	8	1	745.240620	37.650540	6.4	**29**	729.374280	36.928317	15.4
300	9	8	2	424.057690	24.342590	19.4	**28**	407.191520	24.301169	25.4
400	8	4	0	1704.227880	71.566280	3.0	**30**	1674.895060	70.285444	5.6
400	8	8	1	980.987530	42.888140	16.2	**29**	965.054470	41.931984	34.9
400	12	8	0	557.743570	31.479010	54.5	**30**	536.144270	34.064915	58.7
500	10	4	0	2145.301500	73.154580	6.3	**30**	2110.102140	70.194881	10.5
500	10	8	0	1240.201050	48.395080	39.7	**30**	1216.801520	44.478994	65.5
500	15	8	0	699.629590	31.067280	107.8	**30**	673.563680	27.767719	117.0
600	12	4	0	2630.441670	45.924340	13.2	**30**	2590.242360	40.951831	17.7
600	12	8	0	1541.014560	27.596550	61.5	**30**	1506.548520	27.790606	109.0
600	18	8	1	880.182330	26.932450	183.0	**29**	847.450010	27.957043	172.1
700	14	4	1	3023.781870	66.177400	24.6	**29**	2977.782790	60.409126	26.2
700	14	8	0	1778.027740	45.659610	101.0	**30**	1743.695180	41.563750	167.0
700	21	8	0	1015.934540	40.265470	262.9	**30**	974.269160	34.392786	274.7
		Total	121	24076.184390	—	906.7	725	23593.328450	—	1112.8

Table 3.4: Computational results for construction heuristics with local improvement via VND.

Finally, Table 3.5 shows the final results for VNS, GCH-GRASP (GRASP with randomized greedy construction heuristic), and PILOT-GRASP (GRASP with randomized PILOT construction heuristic). Although we performed tests for the entire instances collection of 77 sets, the table only contains a selection of these sets. They represent the global trend and are sufficient for the conclusions.

Due to the highly different characteristics of our metaheuristics and in order to compare them in a fair way, we use a common time limit (t_{max}) as the sole termination criterion. For the largest instances (180 to 700 stations) it is set to one hour, for medium instances (60 to 90 stations) we use 30 minutes, and for small instances (30 stations) we use 15 minutes. The median number of iterations $\widetilde{g_{tot}}$ suggests that all approaches scale well up to 700 instances.

For the GRASP variants we set α according to equation (3.22) in Section 3.3.5. We observe that PILOT-GRASP yields significantly better results than GCH-GRASP, which is supported by the Wilcoxon signed-rank test (< 5% error probability). We also see a clear improvement over the results of GCH-VND and PILOT-VND from the previous table.

As the main competitor for PILOT-GRASP, we chose VNS with GH since we concluded in Section 3.3.7 that this is the most robust VNS variant with the best balance between computation time and solution quality. Especially on larger instances which we are considering now, VNS with GH outperforms the other VNS variants. For the VND inside the VNS we use a fixed order of neighborhood structures as suggested in Section 3.3.4. However, for the GRASP runs the random VND order yields overall better results and we thus employ it for all GRASP tests. Comparing VNS with PILOT-GRASP, we observe an interesting trend. While all approaches perform comparably good on small instances with 30 nodes, VNS with GH dominates the medium-sized instances with 60 to 180 nodes. On large instances with 400 or more nodes, PILOT-GRASP is the clear winner.

In order to check if this trend depends on the chosen time limits (t_{max}), we also performed computational tests with shorter time limitations. Therefore, we apply the same general settings as before, however set the run time t_{max} for instances with small instances (30 stations) to 5 minutes, for medium instances (60 to 90 stations) to 10 minutes and for largest instances (180 to 700 stations) to 15 minutes. Computational results of these short-time runs indicate as well that VNS dominates on medium instances and PILOT-GRASP on large instances. A Wilcoxon signed-rank test with an error level of 5% confirms this observation. Table 3.6 shows a few selected results from these tests. Naturally, the obtained solutions are usually worse than those of Table 3.5 with the larger computation time limits.

Instance set				VNS with GH				GCH-GRASP				PILOT-GRASP			
$\|V\|$	$\|L\|$	$\hat{t}$ [h]	$t_{\max}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{g_{\text{tot}}}$	#best	$\overline{\text{obj}}$	sd	$\widetilde{g_{\text{tot}}}$	#best	$\overline{\text{obj}}$	sd	$\widetilde{g_{\text{tot}}}$
30	1	2	900	28	147.201660	25.480765	1215760.5	19	148.268300	26.048913	2145205.5	**29**	147.135000	25.519076	618965.5
30	1	4	900	**28**	95.203360	19.886650	269610.5	15	96.336670	20.335746	164902.5	24	95.736670	20.582820	100840.0
30	1	8	900	27	29.206390	9.650602	37254.0	8	29.806360	9.845139	14294.5	**28**	29.273050	9.762614	10879.0
60	1	4	1800	**30**	269.603620	30.419674	310747.5	11	271.870230	30.148855	210631.5	18	271.203580	31.599269	77673.0
60	1	8	1800	**26**	170.473670	22.082650	44431.5	6	171.073660	22.082956	23064.5	16	171.006990	22.045548	10889.5
60	2	2	1800	**25**	293.803310	33.040634	505746.0	7	297.536530	31.825752	299233.0	**25**	294.069990	33.236838	190196.5
90	2	4	1800	**26**	346.273900	40.509658	50771.5	2	349.807160	40.166443	44325.0	7	348.740520	41.049992	17740.5
90	2	8	1800	**23**	173.147050	28.282862	7677.0	3	176.146990	28.420301	3566.5	4	175.347020	28.817433	2325.5
90	4	2	1800	**23**	386.740060	47.207416	131270.0	6	391.139920	48.325450	64951.0	16	387.540030	46.335124	43086.5
180	4	4	3600	**28**	718.080960	44.929107	14634.5	0	724.014190	44.884625	9261.5	3	722.614230	44.746593	5184.0
180	4	8	3600	**25**	374.227260	31.242738	1687.5	2	380.893790	33.121376	960.5	3	379.960480	32.680340	739.0
180	5	8	3600	**28**	264.033070	27.593286	996.5	0	272.832870	27.314928	558.5	2	269.832900	26.952757	510.0
300	6	4	3600	**28**	1241.288310	60.463810	3209.0	0	1255.754790	59.872953	1944.5	2	1248.821580	59.773513	1355.5
300	6	8	3600	**15**	715.107800	37.577745	322.5	4	719.241030	35.357035	254.0	11	715.641130	35.701784	201.0
300	9	8	3600	3	403.258240	25.367040	127.5	1	404.524820	23.528193	94.5	**26**	394.391770	22.652595	114.5
400	8	4	3600	**24**	1654.161990	68.573479	1179.5	0	1671.628390	69.262913	747.5	6	1658.495340	69.151828	612.0
400	8	8	3600	3	958.587940	40.943953	120.0	0	959.654480	41.520911	104.5	**27**	949.054670	41.763910	91.5
400	12	8	3600	0	543.477300	33.272886	53.0	1	540.343790	32.175396	37.0	**29**	524.344430	32.440027	52.0
500	10	4	3600	12	2092.169020	70.525137	517.0	0	2111.435410	69.433623	362.5	**18**	2091.902400	69.460983	322.0
500	10	8	3600	1	1225.468080	46.925477	55.5	2	1217.467930	45.867290	51.0	**27**	1202.935050	45.181073	49.5
500	15	8	3600	0	690.229840	34.079096	26.5	0	683.229880	29.705483	18.0	**30**	660.430570	29.605088	28.0
600	12	4	3600	4	2581.042510	37.595799	271.0	0	2596.042260	39.244476	176.0	**26**	2570.176080	39.503520	191.0
600	12	8	3600	0	1526.481530	27.629870	30.5	0	1515.214760	26.484485	30.0	**30**	1491.282070	28.096225	30.0
600	18	8	3600	0	872.849160	27.313824	23.0	0	869.915450	25.603318	10.5	**30**	835.650230	26.489396	18.0
700	14	4	3600	0	2985.649220	62.279120	157.5	0	2989.049100	64.250746	110.0	**30**	2957.783110	61.351625	127.0
700	14	8	3600	0	1764.227980	44.899458	27.0	0	1758.894670	44.698339	18.0	**30**	1733.295320	40.430672	20.0
700	21	8	3600	0	1002.934830	36.485459	25.5	0	1005.400970	36.097162	7.0	**30**	965.869280	31.968564	12.0
		Total	78300	407	23524.928060	—	2596732.0	87	23607.524400	—	2984919.5	527	23292.533490	—	1082253.0

Table 3.5: Computational results of VNS with GH and two GRASP variants with equal run times on a selection of the entire instances collection.

Instance set				**VNS with GH**				**PILOT-GRASP**			
$\lvert V\rvert$	$\lvert L\rvert$	$\hat{t}$ [h]	$t_{\max}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{g_{\text{tot}}}$	#best	$\overline{\text{obj}}$	sd	$\widetilde{g_{\text{tot}}}$
30	1	2	300	28	147.201660	25.480770	569766.5	**29**	147.135000	25.519080	207275.0
30	1	4	300	**28**	95.203360	19.886650	101678.5	24	95.736670	20.582820	32566.0
60	1	4	600	**29**	269.603620	30.419670	107947.0	18	271.203580	31.599270	25369.0
60	1	8	600	**28**	170.473680	21.976200	14632.0	16	171.073660	22.001610	3515.0
90	2	4	600	**26**	346.273910	40.604890	16489.5	6	349.007180	41.164970	5910.0
90	2	8	600	**25**	174.680360	27.952510	2384.5	6	176.680330	28.783920	783.0
300	6	4	900	**21**	1248.954860	61.250840	748.0	9	1250.754900	60.019630	338.5
300	9	8	900	0	411.991360	24.300130	35.5	**30**	398.524990	23.012590	29.0
500	10	4	900	0	2115.502050	70.683370	130.0	**30**	2094.169080	69.288040	80.0
500	15	8	900	0	692.029890	29.335570	16.0	**30**	666.163990	29.783500	8.0
700	14	4	900	0	3003.715600	61.370360	47.5	**30**	2962.583020	59.063480	33.0
700	21	8	900	0	1011.267910	35.516260	7.0	**30**	968.735990	32.905700	4.0
		Total:	8400	185	9686.898260	—	813882.0	258	9551.768390	—	275910.5

Table 3.6: Short runtime (limited to 5 to 15 minutes) results of VNS with GH and PILOT-GRASP.

3.3.8 Conclusions and Future Work

We investigated different metaheuristic approaches for finding good solutions to the problem of balancing bike sharing systems using a fleet of vehicles, focusing on the static problem variant where any user activities during the rebalancing process are neglected.

We started with a greedy construction heuristic for quickly generating meaningful initial solutions. Its particular feature is the greedy criterion in which an estimation of still deliverable bikes is considered. In a further step, the construction heuristic was extended to a PILOT method. The impact of different choices for the PILOT depth parameter were studied. Results showed that the unrestricted variant yielded clearly better results than any depth restriction while still performing relatively fast even on the largest instances.

To locally improve solutions, we applied a VND employing seven neighborhood structures focusing on different problem aspects. For obtaining even better solutions in longer running times, we suggested GRASP based on randomized versions of the greedy construction heuristic as well as the PILOT method and a general VNS that makes use of further, larger neighborhoods exploited by shaking.

All these local search based metaheuristics focus on the search space of vehicle routes, and corresponding loading instructions are derived for each candidate solution as second level decision variables by an embedded method. Four alternative strategies have been described for this purpose and were experimentally compared. The most general method based on linear programming and its functionally equivalent but computationally more efficient implementation based on two maximum flow calculations yield proven optimal solutions for the general non-monotonic case, where bikes may, e.g., be buffered at stations and directly be exchanged between vehicles. However, results indicate that this flexibility only rarely yields better solutions and the average quality gains are rather small. The significantly higher computational complexity does not pay off in general, and therefore we conclude to better stay with the fast and relatively simple greedy heuristic approach for determining loading instructions. Monotonicity is thus a very reasonable restriction for practical applications.

During computational experiments we compared the performance of the two construction heuristics, each optionally followed by the VND, as well as GRASP and VNS. Results show that the PILOT method yields significantly better solutions than the pure greedy construction heuristic in still relatively short running times. Furthermore, GRASP and VNS are able to significantly improve the results obtained by the construction heuristics. They exhibit significant performance differences in dependence of instance size. For medium-sized instances, the VNS showed significant advantages, while for large to very large instances it was outperformed by GRASP. We conclude that for this problem GRASP scales better with respect to instance size and complexity.

In future work further performance gains might possibly be achieved by considering additional advanced neighborhood structures, e.g., very large neighborhoods based on ejection chains or mixed integer programming. For much larger BBSS instances with possibly thousands of stations, it appears to become crucial to combine the suggested methods with clustering, decomposition or multi-level optimization techniques to achieve even better scalability.

Another interesting extension would be to drop the assumption that an increase in balance is always preferred over savings in travel times and the number of loading operations. While our heuristics in principle still work with arbitrary weights for the corresponding terms in the objective function, the advanced techniques for calculating loading instructions do not necessarily produce optimal results anymore. Thus, corresponding extensions of these procedures and, in general, more advanced multi-objective optimization techniques should be considered.

Finally, it is important to also consider the dynamic (online) scenario where bikes are rented or returned by users even during the balancing process. As this problem variant introduces uncertainties, stochastic aspects need to be addressed. From the optimization point of view, it will not suffice to consider the final deviations from target balance values anymore because user demands will constantly change the fill levels of stations over time. Therefore the order in which stations are visited becomes much more relevant. In addition, it is necessary to consider user demand shifts which occur when full stations cause an increased parking position demand whereas empty stations cause an increased bike demand in the neighboring stations.

3.4 Dynamic Balancing Bike Sharing Systems Problem

In this section we extend our previous algorithms for the static case from Section 3.3 towards the dynamic scenario where we take user demands over time into account, and try to reduce unfulfilled demands during the rebalancing process as well as reaching target fill levels for stations at the end. We propose an efficient way to model and simulate these dynamics as well as adapt a greedy and PILOT construction heuristic, Variable Neighborhood Search (VNS) and GRASP accordingly. This section is organized as follows. In Section 3.4.1 the problem for the dynamic scenario is formalized and in Section 3.4.2 we show how the dynamics can be modeled such that the dynamics can be efficiently and incrementally calculated. In Section 3.4.3 adaptations to the greedy construction heuristic are discussed and in Section 3.4.4 metaheuristic approaches from the static case are reviewed. Computational results for the dynamic scenario are given in Section 3.4.5 and finally, we conclude in Section 3.4.6.

3.4.1 Problem Definition

We consider the dynamic scenario of BBSS, referred to as *DBBSS*, where rebalancing is done while we simulate system usage by considering expected cumulated user demands. In addition to the input data for the static problem variant we particularly consider expected user demands from a prediction model.

For the BSS infrastructure we are given a complete directed graph $G = (V \cup \{0\}, A)$ where node set V represents rental stations, node 0 the vehicles' depot, and arc set A the fastest connection between all nodes. Each arc $(u, v) \in A$ is assigned a weight corresponding to the travel time $t_{u,v} > 0$ (including average times for loading and unloading actions). Each station $v \in V$ has a capacity $C_v \geq 0$ denoting the total number of bike slots. The initial fill level p_v is the number of available bikes at the beginning of the rebalancing while the target fill level q_v states the desired number of bikes at the end of the rebalancing. For the rebalancing procedure we are given a fleet of vehicles $L = \{1, \ldots, |L|\}$ where each vehicle $l \in L$ has a capacity $Z_l > 0$. Finally, let $\hat{t}_{\max}$ be the time budget within a vehicle has to finish its route which starts and ends at the depot 0.

Regarding user demands over time we assume the *expected cumulated demand* $\mu_v(t) \in \mathbb{R}$ occurring at each station $v \in V$ from the beginning of the rebalancing process until time t, $0 \leq t \leq \hat{t}_{\max}$ to be given as an essentially arbitrary function. The cumulated demand is calculated by subtracting the expected number of bikes to be returned from the expected number of bikes to be rent over the respective time period. An example of a demand function is shown in Figure 3.10. Note that we display $p_v - \mu_v(t)$ as the dash-dotted line in order to highlight the area where unfulfilled demands occur. Thus, a positive slope of $\mu_v(t)$ indicates that more users are expected to rent bikes than to return them in the time period t, and vice versa. Demands are always fulfilled immediately as far as possible, i.e., bikes or parking slots are available. Unfulfilled demands cannot be fulfilled later and are penalized in the objective function. Let $\hat{\delta}_v^{\mathrm{unf},-}$ denote the total amount of *unfulfilled bike demands* for station $v \in V$, i.e., the number of users who want to rent a

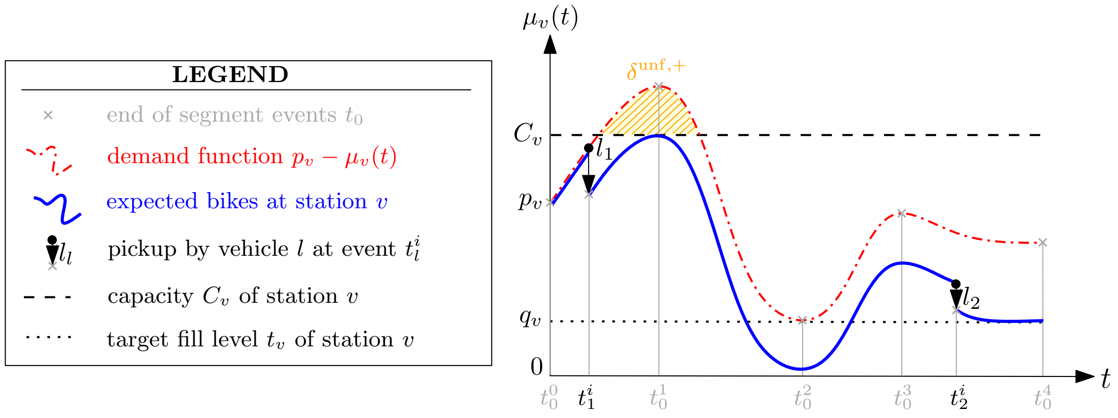


Figure 3.10: Example of a demand function and two pickup events.

bike but are not able to at the desired station. Analogously, let $\hat{\delta}_v^{\text{unf},+}$ refer to the total number of *unfulfilled slot demands*, i.e., the amount of users who cannot return bikes as the station is already full.

A solution to DBBSS consists of a route for every vehicle and corresponding loading instructions for every stop at a station. A route of length ρ_l is defined as an ordered, arbitrarily long sequence of stations $r_l = (r_l^1, \dots, r_l^{\rho_l})$, $r_l^i \in V$ where the depot is assumed to be implicitly added as start and end node. The loading instruction for vehicle $l \in L$ during the i-th stop at station $v \in V$ is denoted as $y_{l,v}^i$. Positive values for $y_{l,v}^i$ denote the corresponding number of bikes to be picked up, negative values denote deliveries. Feasible solutions must fulfill the following conditions. For any station, its fill level (i.e., the number of currently available bikes) must always lie between 0 and its capacity C_v. For any vehicle $l \in L$ the load may never exceed its capacity, i.e., $b_l \leq Z_l$. Moreover, a solution is only feasible, if and only if no route's total travel time, denoted by t_l, exceeds the time budget, and additionally, every vehicle must return empty to the depot 0.

The goal is to find a route for each vehicle with corresponding loading instructions such that the following objective function is minimized:

$$\begin{aligned} f(r,y) = \omega^{\text{unf}} \sum_{v \in V} (\hat{\delta}_v^{\text{unf},-} + \hat{\delta}_v^{\text{unf},+}) + \omega^{\text{bal}} \sum_{v \in V} |q_v - p_v| \\ + \omega^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} |y_{l,r_l^i}^i| + \omega^{\text{time}} \sum_{l \in L} t_l \end{aligned} \tag{3.23}$$

Parameters ω^{unf}, ω^{bal}, ω^{load}, and $\omega^{\text{time}} \geq 0$ are used for controlling the relative importance of the corresponding term in the objective function. The most important goal is to minimize unfulfilled demands as well as to minimize the deviation from the target fill levels. Secondarily, we also want to keep the total number of loading instructions and the total driving time as small as possible, however, those aspects are considered to be clearly less important.

3.4.2 Modeling the dynamic scenario

In this section we show how DBBSS can be modeled by calculating dynamic behavior of the system, i.e., considering the user demands, so that it can be approached by metaheuristics.

Segments and Events

One of our major aims is to avoid a time discretization of the demand functions and corresponding fill level calculations as this would introduce errors and is also time consuming if done in an appropriate resolution. Alternatively, we follow the approach of splitting each cumulated demand function into weakly monotonic segments instead of iterating through all discrete time points. Along with the practically reasonable assumption that the number of segments per station is relatively small, such an approach is much more efficient.

For this purpose, we split function $\mu_v(t)$ into monotonically weakly increasing or decreasing segments. Let $t_0 = (t_0^0, \ldots, t_0^{\rho_0})$ with $t_0^0 = 0$ be an ordered sequence of ρ_0 extreme values of $\mu_v(t)$ so that $\mu_v(t)$ is weakly monotonic for $t \in [t_0^{i-1}, t_0^i]$, $\forall i = 1, \ldots, \rho_0$, see Figure 3.10. Time t_0^i, $i = 1, \ldots, \rho_0$, refers to the end of the i-th weakly monotonic segment. In general, let t_l^i, $\forall l \in L$, $i = 0, \ldots, \rho_l$, be the time when vehicle l performs its i-th stop, i.e.,

$$t_l^i = \begin{cases} 0 & \text{for } i = 0 \\ t_{s_l, r_l^1} & \text{for } i = 1 \text{ if } \rho_l \geq 1 \\ t_l^{i-1} + t_{r_l^{i-1}, r_l^i} & \text{for } i = 2, \ldots, \rho_l \text{ if } \rho_l \geq 2. \end{cases} \tag{3.24}$$

For each station $v \in V$ we define a data structure which denotes the *series of events* $W_v = \langle (l_1, i_1), \ldots, (l_{|W_v|}, i_{|W_v|}) \rangle$. Each event (l_j, i_j), $j = 1, \ldots, |W_v|$ with $l_j \in \{0\} \cup L$ and $i_j \in \{1, \ldots, \rho_{l_j}\}$ either refers to a *station-visit event*, in which case $l_j \in L$ indicates the corresponding vehicle and i_j the number of its stop, or an *end-of-segment event*, in which case $l_j = 0$ and i_j denotes the respective segment of $\mu_v(t)$. Following the above definitions, the time of event (l_j, i_j) is $t_{l_j}^{i_j}$, and all events in W_v are ordered according to increasing times. Multiple events occurring at the same time are ordered arbitrarily, except that an end-of-segment event always appears last.

Expected Number of Bikes at Stations

For each station and event we need to derive a fill level considering the cumulated user demand as well as all performed loading or unloading instructions occurred up to this event.

Let $a_{v,j} \in [0, C_v]$ denote the expected number of bikes at station $v \in V$ and event $j = 1, \ldots, |W_v|$ by considering all expected demands fulfilled as far as possible and all pickups and deliveries performed up to and including event j. Note that, as the cumulated user demand is only a forecast model based on historical data, the fill level of every event may also be fractional. Formally, $a_{v,j}$ is calculated as follows:

$$a_{v,j} = \begin{cases} p_v & \text{for } j = 0 \\ \max(\min(a_{v,j-1} - (\mu_v(t^{i_j}_{l_j}) - \mu_v(t^{i_{j-1}}_{l_{j-1}})), C_v), 0) - y^{i_j}_{l_j} & \text{for } j = 1, \dots, |W_v|. \end{cases} \tag{3.25}$$

End-of-segment events are considered for the correct computation of unfulfillable demands. For the ease of notation, the above formula considers them in the same way as *vehicle-visit events*. Since no bikes are delivered or picked up by these events, we define the loading instructions to be $y^i_0 = 0$, for $i = 1, \dots, \rho_0$.

With respect to unfulfilled demands, we distinguish between *unfulfilled bike demands* $\hat{\delta}^{\text{unf},-}_v$ and *unfulfilled slot demands* $\hat{\delta}^{\text{unf},+}_v$ for each station $v \in V$. They occur whenever the expected cumulated demand $\mu_v(t)$ over time horizon $t \in [0, \hat{t}_{\max}]$ cannot be satisfied, i.e., when $\mu_v(t) < 0 \wedge a_v(t) = 0$ or $\mu_v(t) > 0 \wedge a_v(t) = C_v$, respectively. Unfulfilled demands occurring at station v between events $j-1$ and j, $j = 1, \dots, |W_v|$, can formally be described as

$$\delta^{\text{unf},-}_{v,j} = \max(\mu_v(t^{i_j}_{l_j}) - \mu_v(t^{i_{j-1}}_{l_{j-1}}) - a_{v,j-1} + y^{i_j}_{l_j}, 0) \tag{3.26}$$

$$\delta^{\text{unf},+}_{v,j} = \max(-(\mu_v(t^{i_j}_{l_j}) - \mu_v(t^{i_{j-1}}_{l_{j-1}})) - (C_v - a_{v,j-1}) - y^{i_j}_{l_j}, 0), \tag{3.27}$$

and consequently the overall unfulfilled demands are

$$\hat{\delta}^{\text{unf},-}_v = \sum_{j=1}^{|W_v|} \delta^{\text{unf},-}_{v,j}, \quad \text{and} \quad \hat{\delta}^{\text{unf},+}_v = \sum_{j=1}^{|W_v|} \delta^{\text{unf},+}_{v,j}. \tag{3.28}$$

Classification of Stations

We assume that the stations are well-designed in a sense that their capacities are sufficiently large for daily fluctuations, i.e., it will not be necessary to pick up and deliver bikes to the same station at different times on a single day in order to fulfill all demands. Furthermore, we have shown in previous work [125] that the monotonicity restriction (i.e., it is allowed to either only pick up or deliver bikes at a station) has in practice only a neglectably small impact on the solution quality but substantially simplifies the problem. Additionally, our project partner Citybike Wien mentioned that they only perform either pickups or deliveries at a particular station on the same day. Therefore, we again classify the stations into pickup stations $V_{\text{pic}} \subseteq V$ and delivery stations $V_{\text{del}} \subseteq V$ and impose monotonicity, i.e., allow only the respective operations.

In the static case this classification is done by considering the total deviation in balance for a particular station, i.e., $p_v - q_v \ \forall v \in V$. If this value is less than 0, then the corresponding station refers to the set of delivery stations, and otherwise it is classified as a pickup station. However, in the dynamic case we have to consider user demands during the rebalancing process along with the scaling factors in the objective function (3.23).

Thus, we consider the situation when no rebalancing is done at all. Based on equation (3.28) and objective function (3.23) we determine for each station $v \in V$ the total penalty for slot deficit and bike deficit:

$$\begin{aligned} \delta_v^{\text{missing}} = \omega^{\text{unf}} \hat{\delta}^{\text{unf},+} + \omega^{\text{bal}} \min(0, a_{v,|W_v|} - q_v) - \\ \omega^{\text{unf}} \hat{\delta}^{\text{unf},-} - \omega^{\text{bal}} \min(0, q_v - a_{v,|W_v|}). \end{aligned} \tag{3.29}$$

If $\delta_v^{\text{missing}} < 0$, v is a delivery station. If $\delta_v^{\text{missing}} > 0$, v is a pickup station. Otherwise, if $\delta_v^{\text{missing}} = 0$, the station is already balanced, and thus, will not be considered anymore.

Restrictions on Loading Instructions

For every stop of a vehicle, we need to calculate how many bikes the vehicle is allowed to pick up or deliver at most so that the capacity constraints are never violated and unnecessary unfulfilled demands are never introduced. These bounds are then utilized to set loading instructions for the corresponding vehicle stops later during the optimization process. Formally, we define slacks $\mathit{\Delta} y_{v,j}^-$ and $\mathit{\Delta} y_{v,j}^+$ as the maximum amount of bikes which may be delivered/picked up at station v and event $j = 1, \ldots, |W_v|$.

$$\mathit{\Delta} y_{v,j}^- = \begin{cases} \max(0, q_v - a_{v,|W_v|}) & \text{for } j = |W_v| \\ \min(C_v - a_{v,j}, \mathit{\Delta} y_{v,y+1} + \delta_{v,j+1}^{\text{unf},-}) & \text{for } j = 1, \ldots, |W_v| - 1 \end{cases} \tag{3.30}$$

$$\mathit{\Delta} y_{v,j}^+ = \begin{cases} \max(0, a_{v,|W_v|} - q_v) & \text{for } j = |W_v| \\ \min(a_{v,j}, \mathit{\Delta} y_{v,j+1} + \delta_{v,j+1}^{\text{unf},+}) & \text{for } j = 1, \ldots, |W_v| - 1. \end{cases} \tag{3.31}$$

Note, that we have to iterate backwards by starting with the last event until we reach the time when the currently considered vehicle stop occurs.

By definition, let $\Delta y_{v,j}^{\text{unf},+}$ and $\Delta y_{v,j}^{\text{unf},-}$ denote the slack without including the last event, i.e., starting with event $j = |W_v| - 1$. These two terms are used by the construction heuristic in the next section.

3.4.3 Greedy Construction Heuristic

We extend the Greedy Construction Heuristic (GCH) from our previous work [125] to fit the dynamic case. A vehicle tour is built by iteratively appending stations from a set of *feasible successors* $F \subseteq V$. This set includes each station which can be *improved* and is reachable within the vehicles time budget. An improvement may be achieved if $\delta_v^{\text{missing}} > 0$, $\forall v \in V_{\text{pic}}$, or $\delta_v^{\text{missing}} < 0$, $\forall v \in V_{\text{del}}$. Then, for each station $v \in F$ we compute the total number of bikes that can either be picked up from or delivered to this station:

$$\gamma_v = \begin{cases} \min(\mathit{\Delta} y_{v,j}^-, Z_l - b_l) & \text{for } v \in F \cap V_{\text{pic}}, \\ \min(\mathit{\Delta} y_{v,j}^+, b_l) & \text{for } v \in F \cap V_{\text{del}}. \end{cases} \tag{3.32}$$

Note that b_l denotes the number of bikes currently stored in vehicle l. As shown in (3.32), we need the slacks for determining possible loading instructions as they are calculated by equation (3.30). In order to guarantee that vehicles return empty to the depot, we correct the load for pickup stations by estimating the amount of bikes that can be delivered afterwards in the same fashion as in [125] by recursively looking forward.

It is necessary to consider impacts of loading instructions on a station with respect to target fill level and unfulfilled demands separately and weight them with ω_{bal} and ω_{unf} in the same way as it is done in the objective function (3.23). We obtain

$$g(v) = \begin{cases} \frac{\omega_{\text{bal}} \cdot \min(\gamma_v, \max(0, a_{v,|W_v|} - q_v)) + \omega_{\text{unf}} \cdot \min(\gamma_v, \Delta y_{v,j}^{\text{unf},+})}{t_{u,v}} & \forall v \in V_{\text{pic}}, \\ \frac{\omega_{\text{bal}} \cdot \min(\gamma_v, \max(0, q_v - a_{v,|W_v|})) + \omega_{\text{unf}} \cdot \min(\gamma_v, \Delta y_{v,j}^{\text{unf},-})}{t_{u,v}} & \forall v \in V_{\text{del}}, \end{cases} \tag{3.33}$$

where $t_{u,v}$ is the travel time from the vehicle's last stop u to station v. In each greedy iteration the station with the highest $g(v)$ is appended to the currently considered vehicle tour. Loading instructions are set as follows:

$$y_{v,j} = \gamma_v \text{ if } v \in V_{\text{pic}}, \quad \text{and} \quad y_{v,j} = -\gamma_v \text{ if } v \in V_{\text{del}} \tag{3.34}$$

Since in the dynamic case timing is important, we additionally introduce a term which we refer to as *urgency*. It states how urgent it is to visit stations with future unfulfilled demands. We propose two methods to compute this value.

Additive urgency: For a station v we consider the time of the next period where unfulfilled demands occur. If the vehicle cannot reach the station until the first period starts, we consider the next period, and so on. In case a station has no periods of unfulfilled demands at all or none of them are reachable in time, it is ignored. Moreover, we introduce an additional scaling factor ω_{urg} which denotes the importance of urgency. Formally,

$$u_{\text{add}} = \begin{cases} 0 & \text{if } t_v^{\text{unf}} < t_{u,v} \\ \omega_{\text{urg}} \cdot \frac{\delta_{\text{unf}}}{t_v^{\text{unf}}} & \text{if } t_v^{\text{unf}} \geq t_{u,v} \end{cases} \tag{3.35}$$

where t_v^{unf} is the time left up to the start of the next unfulfilled demand for station $v \in V$ and $t_{u,v}$ is the travel time to the considered station which, by definition, has to be greater than 0. The greedy value including the urgency of the visit $g'(v)$ is then calculated as $g'(v) = g(v) + u_{\text{add}}$.

Multiplicative urgency: In the multiplicative approach we multiply the basic $g(v)$ from (3.33) by an *exponential function.* Again, we consider the time until the next unfulfilled demand, analogously as for the additive approach. The term is computed as

$$u_{\text{mul}} = \exp(-\max(0, t_v^{\text{unf}} - t_{u,v}) \cdot \omega_{\text{urg}}). \tag{3.36}$$

The greedy value criterion is then extended to $g'(v) = g(v) \cdot u_{\text{mul}}$.

PILOT Construction Heuristic: Due to the nature of greedy algorithms, short-sighted decisions cannot be completely avoided no matter how we choose the greedy evaluation criterion. Therefore, we use the PILOT method [157] to address this drawback. The functionality remains the same as in [109] which extends GCH by evaluating each potential successor in a deeper way by constructing a complete temporary route from it, and finally considering its objective value as $g(v)$.

3.4.4 Metaheuristic Approaches

In order to further improve the results obtained by the construction heuristics, we apply Greedy Randomized Adaptive Search (GRASP) and Variable Neighborhood Search (VNS). For both metaheuristic approaches we use an incomplete solution representation based on storing for each vehicle $l \in L$ its route $r_l = (r_l^1, \dots, r_l^{\rho_l})$ only. The loading instructions $y_{l,v}^i,\ l \in L,\ v \in V,\ i = 1, \dots, \rho_l$ are efficiently calculated during evaluation by applying the same greedy strategy as in GCH, see Section 3.4.3, utilizing the restriction procedure from Section 3.4.2 to obtain bounds on the y-variables and accelerate the calculations.

Variable Neighborhood Search: The VNS approach from [125] is adapted with respect to the procedure for deriving loading instructions. The general layout and neighborhood structures remain the same. We use remove-station, insert-unbalanced-station, intra-route-2-opt, replace-station, intra-or-opt, 2-opt*-inter-route-exchange and intra-route 3-opt neighborhood structures for local improvement within an embedded Variable Neighborhood Descent (VND), while for shaking we apply move-sequence, exchange-sequence, destroy-&-recreate, and remove-stations operations.

Greedy Randomized Adaptive Search: We also consider GRASP by extending the construction heuristics in the same way as in our previous work [109] with adaptations for the dynamic problem variant. The idea is to iteratively apply GCH or PILOT from Section 3.4.3 and locally improve each solution with VND. While there we always select the best successor, we use for GRASP a restricted candidate list with respect to the greedy evaluation criterion. The degree of randomization is controlled by a parameter $\alpha \in [0, 1]$. In the dynamic case we used the same values which turned out to work best in the static case.

3.4.5 Computational Results

We performed comprehensive tests for our DBBSS approaches. Generating new benchmark instances was necessary in order to introduce the user demand values. They are based on the same set of Vienna's real Citybike stations we used in our previous works [122, 125, 109]. Cumulated user demands for the stations are piecewise linear functions derived from historical data based on an hourly discretization. The instances that we use for the computational results contain 30 to 90 stations with different numbers of vehicles and

Instance set			GCH				PILOT				GCH-VND				PILOT-VND			
$\|V\|$	$\|L\|$	$\hat{t}_{\max}$	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{t_{\text{tot}}}$ [s]
30	1	8h	0	54.06	12.50	< 0.1	0	50.98	11.19	0.1	**18**	50.61	11.56	0.4	13	49.97	10.95	0.4
30	2	4h	0	59.79	15.65	< 0.1	1	55.47	13.78	< 0.1	9	55.87	13.58	0.2	**22**	54.89	13.44	0.1
60	1	8h	0	186.49	28.14	< 0.1	1	180.59	28.81	0.5	8	180.20	28.72	0.5	**23**	178.85	29.29	0.9
60	2	4h	0	202.69	31.82	< 0.1	0	191.06	29.98	0.2	9	193.32	30.06	0.4	**21**	189.69	29.81	0.4
60	2	8h	0	104.49	12.77	< 0.1	0	98.64	11.03	0.9	12	98.00	12.05	2.4	**18**	96.74	10.80	3.2
60	4	4h	0	118.76	17.38	< 0.1	0	106.98	13.53	0.4	4	108.96	13.34	1.8	**26**	105.36	13.41	1.3
90	1	8h	0	354.83	34.79	< 0.1	0	346.73	33.49	1.1	6	348.20	34.74	0.7	**24**	344.99	33.45	1.5
90	2	4h	0	371.13	34.55	< 0.1	0	360.49	36.06	0.5	5	362.74	35.53	0.5	**25**	358.21	35.09	0.9
90	2	8h	0	232.86	27.07	< 0.1	0	221.02	24.24	2.1	3	223.16	24.71	2.6	**27**	218.57	23.86	4.1
90	3	8h	0	155.26	19.27	< 0.1	0	144.35	16.80	2.8	6	144.43	17.94	8.6	**24**	141.25	16.26	6.9
90	4	4h	0	254.25	27.51	< 0.1	0	239.70	27.86	1.0	7	242.91	27.90	2.1	**23**	237.47	27.63	2.2
90	5	4h	0	210.12	24.26	< 0.1	0	194.03	24.55	1.2	7	195.75	24.38	4.2	**23**	191.72	23.96	3.3
		Total	0	2304.73	285.72	< 0.1	2	2190.04	271.31	10.8	94	2204.15	274.48	24.4	269	2167.71	267.95	25.2

Table 3.7: Results of GCH, PILOT, and the variants with VND.

different time budgets and are available at[3]. As the BSS in Vienna currently consists of 111 stations and 1300 bikes the instances used inhere are realistic and relevant for practice. For each parameter combination exists a set of 30 independent instances. All our algorithms are implemented in `C++` using GCC 4.6. Each test run was performed on a single core of an Intel Xeon E5540 machine with 2.53 GHz. The scaling factors of the objective function are set to $\omega^{\text{unf}} = \omega^{\text{bal}} = 1$, $\omega^{\text{load}} = \omega^{\text{time}} = \frac{1}{100\,000}$, i.e., an improvement with respect to balance and/or unfulfilled demands is always preferred over reducing the tour length and/or the number of loading instructions. For the greedy evaluation criterion we use multiplicative urgency. We omit a detailed comparison since the difference between these strategies becomes more significant only on larger instances with hundreds of stations.

In Table 3.7 we compare different methods for quickly obtaining good starting solutions, namely GCH, PILOT, GCH with VND, and PILOT with VND. The columns show the instance characteristics, and for each algorithm the number of times the corresponding approach yields the best result (#best), the average objective values ($\overline{\text{obj}}$), the standard deviations (sd), and the average CPU-times $\widetilde{t_{\text{tot}}}$. The differences of the average objective values are frequently relatively small due to the weight factors ω^{load} and ω^{time}, but they are still crucial for evaluating the quality of solutions. Therefore, the #best numbers give us a better indication of which algorithm variants perform best. We observe that PILOT outperforms GCH on every instance while the additional time is only moderate. This trend continues when we add VND to further improve the solutions. Not only does PILOT-VND outperform GCH-VND, but it also requires less time. This is due to the superior starting solutions, so VND terminates after fewer iterations.

In Table 3.8 we test our metaheuristic approaches and additionally compare them to the VNS for the static case from [125], denoted by SVNS. For a reasonable comparison, SVNS initially converts a DBBSS instance into a static one by adding for each station

[3]`https://www.ac.tuwien.ac.at/files/resources/instances/bbss/bench3.tar.gz`

Instance set			**SVNS**			**DVNS**			**PILOT-GRASP**		
$\|V\|$	$\|L\|$	$\hat{t}_{\max}$	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd
30	1	8h	4	54.90	10.93	**20**	47.36	10.51	9	47.54	10.57
30	2	4h	4	58.68	13.08	**19**	50.84	11.50	11	50.84	11.35
60	1	8h	0	197.25	30.01	**29**	172.30	27.13	4	172.84	26.96
60	2	4h	0	207.09	30.37	**22**	182.12	29.28	10	183.09	29.13
60	2	8h	0	114.64	12.75	**26**	91.80	10.63	4	92.30	10.40
60	4	4h	0	126.42	15.11	**23**	99.24	11.39	7	99.85	11.54
90	1	8h	0	368.18	37.47	**19**	337.92	32.74	11	338.49	32.23
90	2	4h	0	380.50	38.80	**19**	349.98	33.97	11	351.05	34.74
90	2	8h	0	242.60	26.09	11	210.62	23.79	**19**	210.19	23.03
90	3	8h	0	168.99	16.31	11	135.97	15.10	**19**	135.40	15.06
90	4	4h	0	262.41	30.41	**17**	225.94	25.77	13	226.39	26.19
90	5	4h	0	216.53	23.76	**18**	182.03	21.82	12	182.20	22.21
		Total	8	2398.19	285.10	234	2086.11	253.63	130	2090.16	253.38

Table 3.8: Results of static VNS, dynamic VNS, and PILOT-GRASP.

the final cumulative user demand to the respective target value; negative values and values exceeding the station capacity C_v are replaced by zero and C_v, respectively. The idea is to neglect the timing aspects of station visits and check if this static VNS is able to find reasonable solutions also for the dynamic case. To assure always obtaining feasible solutions to DBBSS in the end, loading instructions for the finally best static solution are recalculated by the new greedy strategy of the dynamic case. We observe that GCH from Table 3.7 already performs a little bit better than the SVNS. DVNS and GRASP are able to compute results that are more than 10% better than those of SVNS. Therefore, we conclude that although it is possible to apply algorithms for the static case to the dynamic scenario, dedicated dynamic approaches taking time-dependent user demands into account are clearly superior. Among the dynamic approaches DVNS performs best on most of the considered instances. According to a Wilcoxon signed-rank test, all observed differences on the overall number of best solutions among any pair of compared approaches are statistically significant with an error level of less than 1%.

3.4.6 Conclusions and Future Work

In this work we showed how to extend the metaheuristics developed in our previous work for static BBSS to the significantly more complex dynamic variant. Starting from a model which can handle essentially arbitrary time-dependent expected user demand functions, we proposed an efficient way to calculate loading instructions for vehicle tours. We use an objective function where the weights of unfulfilled user demands and target fill levels can be adjusted in an easy way. Practically, this has a high relevance for the BSS operator. We also extended our previously introduced construction heuristics, VNS and GRASP, so that dynamic user demands are considered appropriately. Tests on practically realistic instances show that the dynamic approaches indeed make sense. Depending on the available time for optimization, greedy or PILOT construction heuristics are useful for fast runs, while VNS is most powerful for longer runs.

In future work it would be particularly interesting to also consider the impact of demand shifts among stations when their neighbors become either full or empty. Especially, when users want to return bikes and an intended target station is full, this demand obviously will not diminish but be shifted to some neighboring station(s). Considering this aspect might lead to an even more precise model, but also increases the model's complexity significantly. Furthermore, in cooperation with the Austrian Institute of Technology, we want to extend our techniques to a two-layer optimization approach where an upper layer is responsible for the long-term planning while our short-term algorithms handle the daily plans.

3.5 Cluster-First Route-Second Heuristic

This section describes a cluster-first route-second heuristic for BBSS which is extended in the next section. A logic-based Benders decomposition scheme is introduced and computational results for this algorithm are provided.

3.5.1 Introduction

Many major cities around the world already augment their public transport by Bike-Sharing Systems (BSS). They provide an ecofriendly way of traveling in the city and thus reduce emissions by avoiding some of the city's motorized traffic. Last but not least, BSS also motivate the population to do more sports and stay healthy [38]. An important factor to successfully run a BSS is the availability of bikes as well as empty parking slots at each station at any time. Especially this aspect is a major challenge and frequently a problem of existing BSS. Therefore, BSS operators need to continuously redistribute bikes among stations to increase user satisfaction.

Previous works aimed at providing a solution consisting of a route for every vehicle, and additionally, an accurate calculation of the loading operations at each stop of the vehicles. Although the computation of accurate loading instructions makes the problem more complex it seemed necessary to researchers and practitioners. We have also developed approaches for this problem definition in our previous work [86, 110, 109, 122, 124, 125] but after applying our algorithms to the system of *Citybike Wien* in Vienna we got valuable feedback from their technicians. They told us that every day they have more work than they can actually do. Thus, it seems most efficient for them to drive only with full vehicle loads. In this work we exploit this information and neglect loading operations which substantially simplifies the problem. Instead, we classify each station as either a pickup or a delivery station and only consider for each station how much vehicle loads we need to service it. Thus, we are using a Cluster-First Route-Second (CF-RS) approach where we first assign the stations to vehicles, and then, solve the according routing problem for every vehicle separately.

3.5.2 Problem Definition

Given is a set of nodes V representing the stations, and having associated a capacity C_v. Stations that need to be visited several times as multiple full loads need to be picked up or delivered are considered by including a respective number of copies of nodes in V. We define $V_{\text{pic}} \subseteq V$ as the set of all pickup stations and $V_{\text{del}} \subseteq V$ as the set of all delivery stations, i.e., $V = V_{\text{pic}} \cup V_{\text{del}}$. Furthermore, we add a node referred to as 0, representing the depot, and define $V_0 = V \cup \{0\}$. The arc set A represents the fastest connections between the nodes and every arc has associated a traveltime t_{uv}, and formally, $A = \{(u, v) \mid u \in V_{\text{pic}}, v \in V_{\text{del}}\} \cup \{(u, v) \mid u \in V_{\text{del}}, v \in V_{\text{pic}}\}$. Accordingly, we define the arc set $A_0 = A \cup \{(0, v) \mid v \in V_{pic}\} \cup \{(v, 0) \mid v \in V_{del}\}$. The corresponding bipartite graph G is defined as $G = (V, A)$, and graph G_0, including the depot, accordingly as $G_0 = (V_0, A_0)$. By assumption, we are given a homogeneous vehicle fleet L with a common time budget $\hat{t}$ wherein technicians have to finish their routes. Additionally, all vehicles have a common capacity Z defining how many bikes they can transport at the same time. We assume, that all vehicles have to start empty at the depot and return empty to the depot. Every station $v \in V$ is visited at most once.

Our aim is to service as many stations as possible within the time budgets of the drivers. Feasible solutions to the problem do not exceed the time budget for any driver, all routes start with a pickup station and end with a delivery station. Furthermore, the tour must alternate between pickup and delivery stations. Finally, the solution is represented by an alternating tour for every vehicle.

3.5.3 Logic-Based Benders Decomposition Scheme

The idea of classical *Benders Decomposition* is to split the original (compact) model into two parts, namely a *restricted master problem* and a *subproblem* by separating the variables and using LP duality. Logic-Based Benders Decomposition [72] aims at extending this approach to a "logical split" of the problem into a *master problem* and a *subproblem* where the subproblem yields Benders cuts through *logical deduction.* This means, Benders infeasibility cuts are added to the master problem whenever the solution of a subproblem is infeasible and the master problem is then resolved with these cuts.

The Assignment Problem (AP)

deals with assigning stations to vehicles and serves as the master problem of this approach. Much work on the AP in conjunction with *Vehicle Routing Problems (VRP)* is done with heuristic approaches (e.g., Genetic Algorithms) as it can be tough to find good routing costs approximation which can be formulated efficiently inside a MIP. In fact it turned out to be the biggest challenge to effectively encode the routing costs approximation into the MIP model. We need a lower bound on the TSP, because if we would overestimate the routing costs, obviously the subproblem would always be feasible as the optimal TSP tour would be lower than the estimation. Thus, our algorithm would terminate regardless whether the solution is optimal or not. Schuijbroek et al. [140] use a *Maximum Spanning Star* which they have proved to be an upper bound on the routing costs for their problem. However, in our case we need a lower bound, and thus, we ended up by using a single commodity flow formulation for computing a minimum spanning tree (MST) for approximating the routing costs in our MIP model. It has several advantages: It is a standard and well-known problem of graph theory, so it can be formulated very well inside the MIP and can be solved relatively efficiently. As preprocessing we calculate an upper bound ω on the assignment per cluster. Thus, we define a variable $t = 0$. Then, we seek for the pickup station with the least travel cost from the depot and the delivery station with the least travel cost to the depot and add these costs to t. Afterwards, we look for any lowest cost edges to add them also to t as long as $t < \hat{t}$. The number of nodes added during this procedure is an upper bound on the maximum assigned stations per cluster. This helps us to get some bound and our MIP model does not become quadratic (see Eq. 3.43 in the MIP).

Let $x_{vl}\,\forall v \in V_0, l \in L$ be 1 if v is assigned to vehicle l, 0 otherwise, $y^l_{uv}\,\forall (u,v) \in A_0, l \in L$ be 1 if there exists an edge from u to v in the MST for vehicle l, 0 otherwise, $f^l_{uv}\,\forall (u,v) \in A_0$ the flow between nodes u and v for the MST in cluster l and h_l be the routing costs approximation for vehicle l. Moreover, we define the constant τ as the scaling factor for the MST in the objective function. This scaling factor is used to neglect any influence of the MST in the objective function because we only want to maximize the total assigned stations. The MST in the objective function is only used for routing costs approximation. Then our MIP model is formally defined as follows:

$$\max \quad \sum_{l \in L} \sum_{v \in V} x_{vl} - \sum_{l \in L} \tau \cdot h_l \tag{3.37}$$

subject to

$$x_{0l} = 1 \qquad \forall l \in L \tag{3.38}$$

$$\sum_{l \in L} x_{vl} \le 1 \qquad \forall v \in V \tag{3.39}$$

$$\sum_{v \in V_{pic}} x_{vl} = \sum_{v \in V_{del}} x_{vl} \qquad \forall l \in L \tag{3.40}$$

$$\sum_{(0,v) \in A_0} f^l_{0v} = \sum_{v \in V} x_{vl} \qquad \forall l \in L \tag{3.41}$$

$$\sum_{(u,v) \in A_0} f^l_{uv} - \sum_{(v,u) \in A_0} f^l_{vu} = -1 \qquad \forall l \in L, u \in V \tag{3.42}$$

$$f^l_{uv} \le y^l_{uv} \cdot \omega \qquad \forall l \in L, (u,v) \in A_0 \tag{3.43}$$

$$\sum_{(u,v)\in A_0} y^l_{uv} = \sum_{v\in V} x_{vl} \qquad \forall l \in L \tag{3.44}$$

$$y^l_{uv} \leq x_{vl} \qquad \forall l \in L, (u,v) \in A_0 \tag{3.45}$$

$$y^l_{uv} \leq x_{ul} \qquad \forall l \in L, (u,v) \in A_0 \tag{3.46}$$

$$h_l \geq \sum_{(u,v)\in A} y^l_{uv} \cdot t_{uv} \qquad \forall l \in L \tag{3.47}$$

$$h_l \leq \hat{t} \qquad \forall l \in L \tag{3.48}$$

$$x_{vl}, y^l_{uv} \in \{0,1\} \qquad \forall v \in V_0, (u,v) \in A_0, l \in L \tag{3.49}$$

$$h_l, f^l_{uv} \in \mathbb{R}^+ \qquad \forall (u,v) \in A_0, l \in L \tag{3.50}$$

The objective function (3.37) maximizes the stations assigned to vehicles and minimizes the approximated costs by the spanning trees computed for each cluster. Note that not all stations may be assigned to a cluster as it is not possible to serve all stations within the given time limit. Inequalities (3.38)–(3.40) constitute the assignment of stations to the vehicles whereas inequalities (3.41)–(3.46) represent the spanning tree polytope. Equalities (3.38) state that the depot 0 is included in every cluster. Inequalities (3.39) ensure that each station is assigned to at most one vehicle, and equalities (3.40) state that the number of assigned pickup stations must be equal to the number of assigned delivery stations for each cluster. The spanning tree is modeled by a single commodity flow formulation where the outgoing flow of the root node for each cluster must be the number of nodes assigned to that cluster which is ensured by equalities (3.41). For all other nodes except the root node, equalities (3.42) define that every node must consume 1 flow. Inequalities (3.43) restrict the maximum flow value of each cluster. Furthermore, these inequalities define the flow to be 0, if the edge is not chosen by the MST (i.e., $y^l_{uv} = 0$). Equalities (3.44) state that the total number of selected edges for the MST must be the number of stations assigned to that cluster. Equalities (3.45) and (3.46) state that the edge (u, v) can only be chosen by the MST, if both nodes u and v are contained in the cluster l. Equalities (3.47) are used to assign the approximated routing costs for each vehicle $l \in L$ to the variable h_l. Inequalities (3.48) ensure that the approximated routing costs lie within the time budget of each vehicle.

We can extract the assignment of stations from the x-variables. Note that the assignment may not necessarily be feasible as we only approximate the routing costs. The MST is a lower bound on the TSP, and therefore, the optimal routing costs may be higher than the time budget.

Traveling Salesman Problems

have to be solved in our subproblems. After solving the AP we know which stations have to be serviced by which vehicle. Thus, we have to solve a TSP for each vehicle separately on the respective subgraph. As the TSP is already a well-studied problem, we use the State-Of-The-Art TSP solver *Concorde* [30]. However, Concorde is designed for solving only symmetric TSP instances on the complete graph. Thus, we transform our asymmetric instance into a symmetric one as shown in [77]. Infeasible edges (i.e., pickup to pickup and delivery to delivery stations) are modeled by very high routing costs so that these edges will not be used. Because of the maximum time limit of $480min$ in our experiments, we get rather small TSP instances with about 25 stations for each subproblem. *Concorde* is able to solve those instances within milliseconds.

By using *Concorde* and translating the result back, we retrieve an alternating tour through all stations of the cluster starting and ending at the depot with minimal costs. We have to check if the costs of the optimal TSP tour are within the common time limit $\hat{t}$ of the vehicle. If this is the case, we have found a feasible solution. If this is not the case, we know that we have an infeasible assignment, because it is not possible to traverse all stations within the given time limit. Thus, we add a cut for this assignment and resolve the AP.

If all subproblems are feasible which means that the tours of the vehicles lie within their time budget, we have found a feasible and optimal solution to our problem and the algorithm terminates.

Benders infeasibility cuts

are generated whenever we find a cluster which is not feasible due to the time limit of the vehicles. As we deal with a homogeneous vehicle fleet we can add this cut for each vehicle. Let C be the the set of all cuts, then the *Benders cut* is formally defined as: $\sum_{v \in c} x_{vl} \leq |c| - 1 \quad \forall c \in C, l \in L$. This means all sets $S \subseteq V_0$ where $\exists c \in C \ : \ S \supseteq c$ are discarded.

For improving the cuts we use the following additional heuristic: When we examine an infeasible assignment, we try to make the infeasible set smaller to cut off more infeasible assignments. Thus, we look for two stations, one pickup and one delivery station which have the least travel costs in the assignment. We remove these stations and try to resolve the TSP using *Concorde*. If the assignment stays infeasible we have found a better cut, and we can prune more infeasible assignments for the next run of the AP.

3.5.4 Computational Results

In this section we compare the Logic-Based Benders Decomposition with the effective and powerful *Variable Neighborhood Search (VNS)* from [125]. For comparison, the VNS gets also the pre-processed instances as input so that it produces alternating tours picking up as many bikes as possible and delivering as many bikes as possible at every station.

As benchmark set we chose $|V| \in \{10, 20, 30, 60\}$, $|L| \in \{1, 2\}$ and $\hat{t} \in \{120, 240, 480\}$. The instances have been taken from [125] but now we only consider the type of the station rather than the exact fill levels.

The algorithm was implemented in C++ using GCC 4.8.2. For running our tests we used a single core of an Intel XEON E5540 with 2.53 GHz and 3 GB of memory. As MIP solver for the AP and Concorde we used CPLEX 12.6.

We set $\tau = 0.001$ for our experiments. The time limit for the Logic-Based Benders Decomposition was 1 hour and for the VNS we used half an hour. To compare to the VNS, we count the visited stations at the end of rebalancing. For every benchmark type we used 30 different instances and calculated the average.

Table 3.9 shows that small instances can easily be solved to optimality by the Logic-Based Benders Decomposition. Furthermore, all instances up to 30 stations using 2 vehicles and 8 hours of time budget have been solved to optimality. In overall, we obtained optimal solutions for 66% of the selected instance sets. In the third column of the Benders Decomposition we show the *approximation quality* of the MST for the TSP. By *approximation quality* we measure how far off the estimated routing costs of the MST are compared to the optimal costs of the TSP retrieved by Concorde. Generally, it looks that approximation quality becomes better with bigger instances although we sometimes cannot solve all of the instances to optimality. In overall, we have achieved an approximation quality of 18% with respect to the TSP. The number of iterations and cuts are moderate for the selected instances.

When comparing the exact Logic-Based Benders Decomposition with the VNS, we noticed that differences between the exact algorithm and the metaheuristic become bigger when instance sets are larger although the VNS is most of the time not far off from the exact solution.

3.5.5 Conclusions and Future Work

We have proposed a novel problem definition for balancing BSS which is at least in some BSS practically highly relevant (i.e., Citybike Wien), but substantially simplifies the problem. To solve the problem by an exact approach we have come up with a Logic-Based Benders Decomposition scheme. The master problem is modeled as an Assignment Problem and the subproblems as Traveling Salesman Problems accordingly. It is possible to solve these subproblems with State-Of-The-Art methods, like *Concorde*, within milliseconds. Furthermore, we have shown that the Minimum spanning tree is a reasonable approximation for routing costs in the master problem. However, for bigger instances with more than 60 stations exact solutions are not always found within the time limit.

For future work it would be very interesting to investigate Branch-and-Check [150] as the subproblem can be solved very efficiently using *Concorde*. Furthermore, we would like to extend this approach to a heuristic Cluster-First Route-Second algorithm. If the AP would be solved heuristically, we could treat much bigger instances, probably even bigger instances than we have solved before (up to 700 stations in [125]). Moreover, as *Concorde* is very efficient for solving the TSP it would also be interesting to compare to

Instance set			Logic-Based Benders Decomposition						VNS	
$\|V\|$	$\|L\|$	$\hat{t}_{\max}$	Feasible [%]	$\overline{\text{obj}}$	qual [%]	$\overline{\#\text{iterations}}$	$\overline{\#\text{cuts}}$	$\widetilde{\text{t}_{\text{tot}}}$ [s]	$\overline{\text{obj}}$	*diff*
10	2	120	100.00	6.93	23.15	12.97	15.58	0.39	4.93	**-2.00**
10	2	240	100.00	8.20	24.56	1.20	0.76	0.06	8.07	**-0.13**
10	3	120	100.00	7.93	26.55	9.60	14.84	0.32	7.20	**-0.73**
20	2	120	86.67	7.92	22.97	71.88	149.04	48.98	5.37	**-2.56**
20	2	240	60.00	15.67	13.76	18.56	63.38	0.56	10.00	**-5.67**
20	3	120	56.67	11.76	20.82	65.59	163.84	197.22	8.13	**-3.63**
20	3	240	100.00	16.87	18.26	3.57	3.74	1.43	15.90	**-0.97**
30	2	120	40.00	8.00	22.98	14.58	14.27	829.56	5.10	**-2.90**
30	2	240	6.67	21.00	13.61	1.50	0.71	2851.00	9.53	**-11.47**
30	2	480	100.00	26.47	11.24	2.23	3.09	1.13	19.23	**-7.23**
30	3	120	10.00	12.00	24.14	1.00	0.00	3597.89	7.70	**-4.30**
30	3	240	30.00	23.78	14.39	8.00	5.96	7.79	17.53	**-6.24**
60	3	480	73.33	51.91	9.03	20.36	21.81	456.10	36.10	**-15.81**
			66.41	218.44	18.88				154.80	**-63.64**

Table 3.9: Results from our computational tests showing Logic-Based Benders Decomposition

a Route-First Cluster-Second algorithm as these approaches got very efficient during the last decade [115].

3.6 Full-Load Route Planning By Logic-Based Benders Decomposition

We propose three exact solution approaches: a compact MIP model, a logic-based Benders decomposition (LBBD), and a variant thereof, namely Branch-and-Check (BAC). Moreover, we compare with previously proposed and leading metaheuristics allowing flexible numbers of picked up and delivered bikes, concluding that the restriction to only full vehicle loads affects the finally achieved balance in practical scenarios indeed in only minor ways. We introduce the new problem formulation in detail and provide a compact MIP model there for. Moreover, a logic-based Benders decomposition algorithm is introduced and also a variant thereof, namely branch-and-check. It is shown how the cuts can be optimized in order to make the algorithm perform better. In the remainder computational results are given for the proposed algorithms.

3.6.1 Problem Statement

We first summarize aspects of existing problem formulations for *Balancing Bike Sharing Systems* (BBSS) and then state our new approach, giving respective formal definitions.

Generally, previous works distinguish two types of problem variants for BBSS, namely the *static* and the *dynamic* case.

In the *static scenario* we are *given an initial state* of the system, i.e., initial fill levels for all stations, and a *desired target state* of the system, i.e., target fill levels or demand intervals for all stations.

For the static case, a significant variety of different optimization goals has been considered in the literature, e.g., minimizing the traveling costs [21, 37] where balancing is modeled as a hard constraint, or minimizing the total number of expected shortages [126].

A quite challenging task is to determine best suited target fill levels for the optimization. This has to be done with caution because the final state at the end of rebalancing is the initial state for the next day(s) in the static model. The customer demand of renting and returning bikes is the crucial factor when target values for the rebalancing operations have to be determined. Thus, a sophisticated demand prognosis is necessary to estimate well-suited target values. Rudloff and Lackner [132] build such a prognosis model based on historical data of the system of Citybike Wien based on various impact factors like weather, day of the week, time of the day, temperature, etc. They also consider the influence of entirely full or empty neighboring stations. Han et al. [62] concentrate on the demand prediction for large-scale BSS. They describe the spatio-temporal correlation in BSS as an important factor for demand estimation. They verified their model on the record set they retrieved from the BSS *Vélib'* in Paris.

In general, the static problem variant neglects the dynamic interaction between the customers and the system as it does not consider the user demand during rebalancing and e.g., is appropriate for overnight rebalancing if the system is not in use during the night [126].

The *dynamic case* also considers user interactions during rebalancing. Only few works, however, exist in this direction. In [86] the user interactions and the demands are retrieved from historical data and implemented by a probabilistic model of Rudloff and Lackner [132], and the objective is to minimize *unsatisfied user demand* as well as to minimize deviation between initial and desired target fill levels. Contardo et al. [28] randomly generate demand values and try to minimize shortages and excesses of bikes over a prospective time horizon.

If the user demand is predicted reasonably well and the rebalancing takes place during the active times of the PBS, the dynamic case can thus in principle be more accurate than a static model but is also computationally much more demanding. Under the assumption that rebalancing should not primarily fulfill short-term needs and station capacities are reasonably large, static models are generally also accepted as a reasonably good approximation for systems where the rebalancing takes place during the operation hours. We therefore also concentrate on the static case here.

A BBSS Formulation Considering Full Vehicle Loads Only

As motivated already in the introduction, observations at *Citybike Wien* reveal that pickup and delivery of full vehicle loads clearly dominates practice. Due to economic reasons there is a financial limit on the labor costs, and rebalancing is done in such a way that a practically acceptable but usually not perfect balance of the stations' fill levels is achieved. Thus, the number of drivers respectively vehicles and their working times are a major limit, and the stations should be brought to specified target fill levels as far as possible, but reaching all of them exactly is (typically) out of question. The drivers are in principle daily faced with more work than can be feasibly done. Furthermore, many stations ideally require more than one, sometimes even several full vehicle loads to be delivered or picked up in order to achieve the desired target state. Most of the drivers' working time is consumed by traveling to the individual stations and parking somewhere nearby, however, required time for loading or unloading less or more bikes plays a comparably small role, and is frequently also neglected in existing models. In such a scenario, it becomes obvious that it is clearly most effective to move almost always approximately full vehicle loads from stations with a substantial excess of bikes to stations with a substantial demand.

Consequently, we assume in our new BBSS problem formulation that the vehicle is always either fully loaded with bikes or empty, dropping the consideration of moving only a certain number of bikes less than the vehicles full capacity. Concerning the objective function, our goal is to bring as many stations as far as possible to their specified target fill levels, respecting given working times, and the general constraints for feasible tours.

Considering only full vehicle loads simplifies existing models substantially. Typically, the consideration of the exact number of bikes to be moved requires an additionally embedded flow problem to be solved.

Of course, not dealing with partial vehicle loads comes along with a potential loss of accuracy, but the prediction of user demands which depends on, e.g., the weather, weekday, events in the stations' neighborhoods and the influence of neighboring stations involve in general uncertainties for the calculation of suitable target fill levels that can be safely assumed to dominate in practice.

Formal Problem Definition

We are given a set of stations S and a set of homogeneous vehicles L. For the vehicles we are given a common capacity Z and a common time budget $\hat{t}$ (drivers' shift times) within which the vehicles have to finish their routes. For each station $s \in S$ we are given the number of full vehicle loads f_s to be delivered ($f_s \leq -1$) or picked up ($f_s \geq 1$) such that the station achieves its (approximately) ideal target fill level. Stations that are already at their desired target fill level (or require less than a full vehicle load) are ignored from any further consideration.

A station, to which bikes shall be delivered is called a *delivery station*, while a station from which bikes should be removed is called a *pickup station*. At pickup stations, only pickups may be performed, while at delivery stations, only deliveries, and we never allow more than $|f_s|$ visits at each station. Thus, a kind of buffering bikes at some station and moving them further later is explicitly excluded. Especially in our context with the consideration of full vehicle loads only, such solutions would not make sense anyway when the triangle inequality is fulfilled by the traveling times between stations, what can safely be assumed for practice.

For modeling tours with up to $|f_s|$ visits at each station $s \in S$, we define a directed bipartite graph $G = (V, A)$ as follows. Let $V_{\text{pic}} = \{(s, i) \mid s \in S \wedge f_s \geq 1,\ i = 1, \ldots, |f_s|\}$ be a set of nodes representing up to $|f_s|$ visits at each pickup station, and let $V_{\text{del}} = \{(s, i) \mid s \in S \wedge f_s \leq -1,\ i = 1, \ldots, f_s\}$ denote the respective potential visits at the delivery stations. $V = V_{\text{pic}} \cup V_{\text{del}}$ then refers to the joined set of all potential visits, and the arc set of graph G is given by $A = \{(u, v), (v, u) \mid u \in V_{\text{pic}}, v \in V_{\text{del}}\}$.

We further extend the set of stations V by two nodes 0 and $0'$ representing the depot at the beginning and the end of each tour, respectively, obtaining $V_0 = V \cup \{0, 0'\}$. Node 0 is connected to all pickup nodes, while $0'$ is connected to all delivery nodes, i.e., $A_0 = A \cup \{(0, v) \mid v \in V_{\text{pic}}\} \cup \{(v, 0') \mid v \in V_{\text{del}}\}$, yielding bipartite graph $G_0 = (V_0, A_0)$. We explicitly omit here an arc $(0, 0')$ which might be used for representing a vehicle that stays at the depot and does not do any station visits due to the fundamental assumption in our modeling that more than enough rebalancing work exists for keeping all vehicles busy.

Each arc $(u, v) \in A_0$ represents an actual trip from the location represented by visit u to the location represented by visit v and has a corresponding traveling time $t_{uv} > 0$ associated. This time also includes an estimated time for parking at the destination and in case of $v \neq 0'$ for handling the station's electronic system and for loading or unloading the bikes.

A solution to our problem is a set of $|L|$ simple paths in G_0 from node 0 to node $0'$ visiting all vertices in their vehicle's $l \in L$ corresponding subgraph. Let $r_l = (r_l^1, r_l^2, \ldots, r_l^{\rho_l})$ be the successive station visits in the route of vehicle $l \in L$, with ρ_l being the number of visits and $V(r_l)$ corresponding to the set of station visits contained in route r_l. Due to the bipartite structure of G_0, as long as the path is not empty ($\rho_l > 0$) each odd stop must be performed at a pickup station, i.e., $r_l^1, r_l^3, \ldots, r_l^{\rho_l - 1} \in V_{\text{pic}}$, while each even stop takes place at a delivery station, i.e., $r_l^2, r_l^4, \ldots, r_l^{\rho_l} \in V_{\text{del}}$, and ρ_l always is even.

A non-empty route r_l is feasible with respect to the time budget $\hat{t}$ iff

$$t_{0 r_l^1} + \sum_{i=1}^{\rho_l - 1} t_{r_l^i r_l^{i+1}} + t_{r_l^{\rho_l} 0'} \ \leq\ \hat{t}. \tag{3.51}$$

By assumption all vehicles start empty at the beginning and have to return empty, which is implicitly guaranteed again by the bipartite graph.

By above definitions, we reduce the BBSS problem as introduced in [125] to a *selective unit-capacity one-commodity pickup and delivery problem with time budgets on a bipartite graph.*

As optimization goal, we consider in this work the maximization of the total number of station visits

$$\max \sum_{l \in L} \rho_l, \tag{3.52}$$

which corresponds to twice the number of moved full vehicle loads. By this objective function, we also minimize the sum of the deviations from the stations' target fill levels after the rebalancing, which is

$$\min \sum_{s \in S} |f_s| - \sum_{l \in L} \rho_l \tag{3.53}$$

and is the primary objective of previous work such as [41, 40, 70, 125, 126]. In [125] the objective is particularly given as follows:

$$\min\ \omega^{\text{bal}} \sum_{s \in S} \delta_s + \omega^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} |y^i_{l,r^i_l}| + \omega^{\text{work}} \sum_{l \in L} t_l, \tag{3.54}$$

where $\omega^{\text{bal}}, \omega^{\text{load}}, \omega^{\text{work}}$ are weighting factors, $\delta_s = |a_s - q_s|$ denotes the deviation of the final fill level a_s from the target fill level q_s at station $s \in S$, y^i_{l,r^i_l} denotes the number of bikes loaded (> 0) or unloaded (< 0) by vehicle $l \in L$ at station r^i_l, and t_l is the total working time of vehicle l.

Proposition 2. *When considering only balance optimization, the objective functions shown in equation (3.53) and (3.54) correspond to each other (except for rounding errors resulting from the fact that we now only consider full vehicle loads).*

Proof. As we only focus on the balance aspect here, i.e., minimizing the deviation between final and target fill levels, we set the weighting factors in equation (3.54) to $\omega^{\text{bal}} = 1$, $\omega^{\text{load}} = 0$, $\omega^{\text{work}} = 0$, effectively ignoring the second and third term. Equation (3.54) can then be rewritten as

$$\min \sum_{s \in S} |a_s - q_s| = \min \sum_{s \in S_{\text{pic}}} a_s - q_s + \sum_{s \in S_{\text{del}}} q_s - a_s. \tag{3.55}$$

Let p_s be the initial fill level for station $s \in S$, then in a static context, the final fill level can also be expressed as $a_s = p_s - \sum_{l \in L} \sum_{i=1}^{\rho_l} y^i_{l,r^i_l}$ which results in

$$\min \sum_{s \in S_{\text{pic}}} \left(p_s - q_s - \sum_{l \in L} \sum_{i=1}^{\rho_l} y^i_{l,s} \right) + \sum_{s \in S_{\text{del}}} \left(q_s - p_s + \sum_{l \in L} \sum_{i=1}^{\rho_l} y^i_{l,s} \right). \tag{3.56}$$

To show the correspondence of equation (3.53) to equation (3.56), equation (3.53) is multiplied by the vehicle capacity Z such that the deviation in full vehicle loads is transformed to the actual deviation in the number of bikes

$$\min \; Z \cdot \left(\sum_{s \in S} |f_s| - \sum_{l \in L} \rho_l \right) \tag{3.57}$$

Required full loads to balance station $s \in S$ can be calculated as follows

$$f_s = \left\lceil \frac{p_s - q_s}{Z} \right\rceil \forall s \in S_{\text{pic}}, \quad \text{and} \quad f_s = \left\lfloor \frac{p_s - q_s}{Z} \right\rfloor \forall s \in S_{\text{del}}, \tag{3.58}$$

which can be used in equation (3.57) to get

$$\min \; Z \cdot \left(\sum_{s \in S_{\text{pic}}} \left\lceil \frac{p_s - q_s}{Z} \right\rceil - \sum_{s \in S_{\text{del}}} \left\lfloor \frac{p_s - q_s}{Z} \right\rfloor - \sum_{l \in L} \rho_l \right). \tag{3.59}$$

Let $b_s = \sum_{l \in L} |\{r_l^i \mid r_l^i \in r_l, r_l^i = s\}| \; \forall s \in S$ the number of full vehicle loads delivered to or picked up at station $s \in S$, then we can rewrite equation (3.59) as

$$\begin{aligned}
&\min \; Z \cdot \left(\sum_{s \in S_{\text{pic}}} \left\lceil \frac{p_s - q_s}{Z} \right\rceil - \sum_{s \in S_{\text{del}}} \left\lfloor \frac{p_s - q_s}{Z} \right\rfloor - \sum_{s \in S} b_s \right) = \\
&\min \; Z \cdot \left(\sum_{s \in S_{\text{pic}}} \left\lceil \frac{p_s - q_s}{Z} \right\rceil - \sum_{s \in S_{\text{del}}} \left\lfloor \frac{p_s - q_s}{Z} \right\rfloor - \sum_{s \in S_{\text{del}}} b_s - \sum_{s \in S_{\text{pic}}} b_s \right) = \\
&\min \; Z \cdot \left(\sum_{s \in S_{\text{pic}}} \left(\left\lceil \frac{p_s - q_s}{Z} \right\rceil - b_s \right) - \sum_{s \in S_{\text{del}}} \left(\left\lfloor \frac{p_s - q_s}{Z} \right\rfloor - b_s \right) \right) = \\
&\min \; \sum_{s \in S_{\text{pic}}} \left(Z \cdot \left\lceil \frac{p_s - q_s}{Z} \right\rceil - Z \cdot b_s \right) + \sum_{s \in S_{\text{del}}} \left(Z \cdot \left\lceil \frac{q_s - p_s}{Z} \right\rceil + Z \cdot b_s \right)
\end{aligned} \tag{3.60}$$

Comparing equation (3.56) with equation (3.60) shows that the terms $Z \cdot b_s$ and $\sum_{l \in L} \sum_{i=1}^{\rho_l} y_{l,s}^i$ correspond to each other as both represent the number of moved bikes by the vehicles in the system. Moreover, the terms $Z \cdot \left\lceil \frac{p_s - q_s}{Z} \right\rceil, Z \cdot \left\lceil \frac{q_s - p_s}{Z} \right\rceil$ and $p_s - q_s, q_s - p_s$ correspond to each other except for rounding errors due to the consideration of only full vehicle loads. □

Compact Mixed Integer Linear Programming Model

We now formulate the problem as a compact MIP model using *assignment variables* $x_{vl} \in \{0,1\}$ to state the assignment of station visits $v \in V$ to vehicles $l \in L$ and *arc selection variables* $y_{uv}^l \in \{0,1\}$ to describe the tour for each vehicle. Subtours are eliminated via Miller-Tucker-Zemlin inequalities [96] utilizing further continuous variables a_v for the nodes $v \in V$.

$$\max \quad \sum_{l \in L} \sum_{v \in V} x_{vl} \tag{3.61}$$

$$\text{s.t.} \quad \sum_{l \in L} x_{vl} \leq 1 \qquad \forall v \in V \tag{3.62}$$

$$\sum_{v \in V_{\mathrm{pic}}} x_{vl} = \sum_{v \in V_{\mathrm{del}}} x_{vl} \qquad \forall l \in L \tag{3.63}$$

$$\sum_{l' \in L} x_{(s,i)l'} \geq x_{(s,i+1)l} \qquad \forall s \in S,\ l \in L,\ i = 1, \dots, f_s - 1 \tag{3.64}$$

$$\sum_{v \in V_{\mathrm{pic}}} y^l_{0v} = 1 \qquad \forall l \in L \tag{3.65}$$

$$\sum_{v \in V_{\mathrm{del}}} y^l_{v0'} = 1 \qquad \forall l \in L \tag{3.66}$$

$$\sum_{(u,v) \in A_0} y^l_{uv} = x_{ul} \qquad \forall l \in L,\ u \in V \tag{3.67}$$

$$\sum_{(u,v) \in A_0} y^l_{uv} = x_{vl} \qquad \forall l \in L,\ v \in V \tag{3.68}$$

$$\sum_{(u,v) \in A_0} y^l_{uv} = \sum_{(v,u) \in A_0} y^l_{vu} \qquad \forall l \in L,\ v \in V \tag{3.69}$$

$$a_u - a_v + |V| \cdot y^l_{uv} \leq |V| - 1 \qquad \forall l \in L,\ (u,v) \in A \tag{3.70}$$

$$\sum_{(u,v) \in A_0} t_{uv} \cdot y^l_{uv} \leq \hat{t} \qquad \forall l \in L \tag{3.71}$$

$$x_{vl} \in \{0,1\} \qquad \forall l \in L,\ v \in V \tag{3.72}$$

$$y^l_{uv} \in \{0,1\} \qquad \forall l \in L,\ (u,v) \in A_0 \tag{3.73}$$

$$1 \leq a_v \leq |V|, a_v \in \mathbb{R} \qquad \forall v \in V \tag{3.74}$$

The objective function (3.61) maximizes the number of full vehicle loads picked up and delivered to the stations and thus, the total balance increase in the PBS, compare equation (3.53). Inequalities (3.62) state that every station visit is performed by at most one vehicle. By equalities (3.63) we explicitly define that every tour contains the same amount of pickup visits as delivery visits. Note that these equations are in principle redundant but we include them nevertheless as they might be helpful from a computational point of view. Inequalities (3.64) are used for symmetry breaking among the visits of the same station: The $i+1$-th visit can only be performed when the i-th visit is performed, for $i = 1, \dots, f_s - 1$ and each station $s \in S$. For each vehicle the depot's starting node 0 has to have one outgoing arc (3.65), and similarly, the depot's target node $0'$ has to have one incoming arc (3.66). The arc selection variables are linked with the assignment variables as follows: Equalities (3.67) ensure that every node $u \in V$ has one outgoing arc iff it is assigned to vehicle l, i.e., $x_{ul} = 1$, while equalities (3.68) guarantee that each node $v \in V$ which is assigned to vehicle $l \in L$ has to have one corresponding ingoing arc. Equalities (3.69) express that the number of ingoing arcs has to be equal to the number of outgoing arcs for each node $v \in V, l \in L$. We eliminate subtours by inequalities (3.70) by computing an ordering of the nodes in variables a_v. Inequalities (3.71) guarantee that the routes for each vehicle lie within the allowed time budget $\hat{t}$. Finally, (3.72) to (3.74) define the domains of the decision variables.

For small instances, a state-of-the-art MIP solver such as CPLEX is able to directly yield proven optimal solutions by this model in reasonable time, see the experimental results in Section 3.6.3. The approach, however, does not scale well to larger instances.

3.6.2 Logic-based Benders decomposition

We first introduce the term *LBBD* and then describe the application of an LBBD scheme to BBSS.

Introduction

In 1962 Benders came up with his classical decomposition technique to solve large MIP problems [9]. This approach is in principle applicable if the problem can be split into a master problem making use of only a subset of the variables including the complicating integer variables, and an easier subproblem on the remaining continuous variables when the master problem variables are assumed to be fixed to certain values. The solution approach iterates by solving master problem instances and subproblems. After the master problem is solved, a corresponding subproblem is obtained by fixing the master problem's variables in the original formulation to the obtained values. From the solution of the subproblem's *linear programming (LP) dual* one derives feasibility and/or optimality cuts which are added to the master problem in each iteration. The whole process is repeated until no further Benders cuts can be derived and an optimal solution has been obtained.

Erdoğan et al. [47] propose a Benders decomposition scheme for solving the static rebalancing problem arising in BSS. When applying Benders decomposition to VRPs often the master problem, containing the complicating variables, is hard to solve. Thus, Lai et al. [89] came up with a hybrid of Benders decomposition and a genetic algorithm (GA). They solve the master problem by the GA and the subproblems via a MIP model by a commercial solver.

LBBD generalizes classical Benders decomposition by also allowing integer variables or even nonlinearities in the subproblem. This is achieved by replacing the LP dual by a more general concept called *inference dual* [72]. Typically, Benders cuts are here obtained via logical deduction. In several applications, in particular in the domain of scheduling, LBBD achieved remarkable results.

Hooker [71] presents a solution method applicable to generic scheduling problems where he models the master problem as a MIP and solves the subproblems by constraint programming (CP). Reported results on the LBBD outperform a pure MIP and and a pure CP approach. Harjunkoski and Grossmann [63] propose a decomposition approach for multistage scheduling problems. The master problem, an assignment problem, is modeled as a MIP whereas for the subproblems they employ two strategies for feasibility checking: One which utilizes a CP approach and another one where a MIP model is used for the feasibility check. They have shown that the hybrid decomposition approach by solving the master problem as a MIP and the subproblems with their CP approach has been superior to a pure MIP or pure CP approach. Furthermore, solving the subproblem, the sequencing of jobs, with the CP approach has been superior to the feasibility check by the MIP.

There are two types of Benders cuts, namely, infeasibility cuts and optimality cuts. Infeasibility cuts state that the current master solution is not feasible and avoid its generation in future iterations, whereas optimality cuts provide new bounds on the objective value for the current master problem solution. In every iteration except the last, one or more cuts are generated where every single cut reduces the master problem's search space, or more precisely its underlying LP polytope – the more the better in general. Thus, it should also be considered to strengthen obtained Benders cuts as far as possible, which is especially in case of the LBBD frequently done by heuristics or by constraint programming techniques, cf. the greedy algorithms proposed by Hooker [71].

We note that a technique which is similar to the principles of LBBD is called *combinatorial Benders cuts*, cf. Codato and Fischetti [27].

Application to BBSS

The problem consists of an assignment problem (AP) and multiple Hamiltonian path problems with time budgets that are interconnected. The AP is given in the proposed model by equations (3.62)–(3.64), the Hamiltonian path problems are represented by equations (3.69)–(3.71), and the connections between the AP and Hamiltonian path problems are given by equation (3.67) and (3.68). In the following we decompose the problem correspondingly by applying *LBBD*. In this approach, we iteratively solve a master problem, corresponding to the AP, and subproblems corresponding to the Hamiltonian path problems but are modeled as ATSPs. The solutions of the subproblems will yield Benders infeasibility cuts for restricting the master problem in the further iterations. The following section discusses this decomposition approach in detail.

In the following we show how LBBD is applied to our MIP for BBSS. Section 3.6.2 describes the master problem and states its MIP formulation, while Section 3.6.2 discusses the subproblem and proposes a corresponding solution approach. Section 3.6.2 shows how the master problem and subproblem interact and how the algorithm finally yields an optimal solution. Section 3.6.2 introduces the alternative to LBBD, namely BAC.

Master problem: We decompose the model (3.61)–(3.74) from Section 3.6.1 by focusing in the master problem on the clustering aspect, i.e., the AP, yielding multiple Hamiltonian path problems as subproblems. Our method was inspired by the cluster-first route-second method introduced by Fisher and Jaikumar [50] and also applied for BBSS by Schuijbroek et al. [140].

In order to strengthen the master problem such that a relatively meaningful clustering is determined from the beginning on, it is crucial to estimate the route durations for the cluster and exclude clusters that obviously cannot be handled by a single vehicle. Hooker [71] also reveals that it is important, for the success of the approach, to include a *relaxation of the subproblem within the master problem.* Ideally, this route duration estimation should come close to the real minimal Hamiltonian path durations and introduce only a reasonable overhead in the master problem's model. However, it is important that the determined approximate trip durations are guaranteed lower bounds for the real durations, as otherwise sets of station visits might be excluded from becoming clusters, despite feasible routes would actually exist for them.

A lower bound for a TSP that can relatively easily be expressed by a linear program is obtained from the minimum spanning tree relaxation of the TSP. As we can model the Hamiltonian path problem as an ATSP, we relax the problem of finding an optimal ATSP tour to the *minimum* 0-*arborescence problem*, i.e., a minimum, from the depot outgoing, arborescence.

The MIP formulation of our master problem primarily uses the assignment variables $x_{vl},\ v \in V,\ l \in L$ from the original problem. For determining the lower bounds for the vehicles' tour durations via the arborescence polytope, flow variables f^l_{uv} and arc selection variables $y^l_{uv} \in \{0,1\}$ for all vehicles $l \in L$ and arcs $(u,v) \in A_0$ are used.

Furthermore, we define β to be an upper bound on the maximal number of station visits per vehicle. This upper bound is derived by solving the single-vehicle case of the problem which is given as follows:

$$\max \quad \sum_{v \in V} x_v \tag{3.75}$$

$$\text{s.t.} \quad x_v \le 1 \qquad \forall v \in V \tag{3.76}$$

$$\sum_{v \in V_{\text{pic}}} x_v = \sum_{v \in V_{\text{del}}} x_v \tag{3.77}$$

$$x_{(s,i)} \ge x_{(s,i+1)} \qquad \forall s \in S, i = 1, \dots, f_s - 1 \tag{3.78}$$

$$\sum_{v \in V_{\text{pic}}} y_{0v} = 1 \tag{3.79}$$

$$\sum_{v \in V_{\text{del}}} y_{v0'} = 1 \tag{3.80}$$

$$\sum_{(u,v) \in A_0} y_{uv} = x_u \qquad \forall u \in V \tag{3.81}$$

$$\sum_{(u,v) \in A_0} y_{uv} = x_v \qquad \forall v \in V \tag{3.82}$$

$$\sum_{(u,v) \in A_0} y_{uv} = \sum_{(v,u) \in A_0} y_{vu} \qquad \forall v \in V \tag{3.83}$$

$$a_u - a_v + |V| \cdot y_{uv} \leq |V| - 1 \qquad \forall (u,v) \in A \qquad (3.84)$$

$$\sum_{(u,v)\in A_0} y_{uv} \cdot t_{uv} \leq \hat{t} \qquad (3.85)$$

$$x_v \in \{0,1\} \qquad \forall v \in V \qquad (3.86)$$

$$y_{uv} \in \{0,1\} \qquad \forall (u,v) \in A_0 \qquad (3.87)$$

$$1 \leq a_v \leq |V| \qquad \forall v \in V \qquad (3.88)$$

This single vehicle case is in practice much easier to solve than our complete problem. In our test discussed in Section 3.6.3, we typically obtained optimal solutions within seconds, and stopped the solving after a CPU-time limit of 5min and then took the obtained rounded down upper bound to the optimal solution value as β.

Given these decision variables, preprocessing values and parameters, the *master problem* (MP) is stated as follows:

$$\max \quad \sum_{l\in L}\sum_{v\in V} x_{vl} \qquad (3.89)$$

$$\text{s.t.} \quad \sum_{v\in V} x_{vl} \leq \beta \qquad \forall l \in L \qquad (3.90)$$

$$\sum_{l\in L} x_{vl} \leq 1 \qquad \forall v \in V \qquad (3.91)$$

$$\sum_{v\in V_{\text{pic}}} x_{vl} = \sum_{v\in V_{\text{del}}} x_{vl} \qquad \forall l \in L \qquad (3.92)$$

$$\sum_{l'\in L} x_{(s,i)l'} \geq x_{(s,i+1)l} \qquad \forall s \in S,\ l \in L,\ i = 1,\ldots,f_s - 1 \qquad (3.93)$$

$$\sum_{(0,v)\in A_0} y^l_{0v} = 1 \qquad \forall l \in L \qquad (3.94)$$

$$\sum_{(u,0')\in A_0} y^l_{u0'} = 1 \qquad \forall l \in L \qquad (3.95)$$

$$y^l_{uv} \leq x_{ul} \qquad \forall l \in L,\ u \in V,\ (u,v) \in A_0 \qquad (3.96)$$

$$y^l_{uv} \leq x_{vl} \qquad \forall l \in L,\ v \in V,\ (u,v) \in A_0 \qquad (3.97)$$

$$\sum_{(0,v)\in A_0} f^l_{0v} = \sum_{v\in V} x_{vl} + 1 \qquad \forall l \in L \qquad (3.98)$$

$$\sum_{(v,0')\in A_0} f^l_{v0'} = 1 \qquad \forall l \in L \qquad (3.99)$$

$$\sum_{(u,v)\in A_0} f^l_{uv} - \sum_{(v,w)\in A_0} f^l_{vw} = x_{vl} \qquad \forall l \in L,\ v \in V \qquad (3.100)$$

$$f^l_{uv} \leq \begin{cases} (\beta+1)\cdot y^l_{0v} & \text{if } u = 0 \\ \beta \cdot y^l_{uv} & \text{if } v \in V_{\text{pic}} \\ (\beta - 1)\cdot y^l_{uv} & \text{else} \end{cases} \qquad \forall l \in L,\ (u,v) \in A_0 \qquad (3.101)$$

$$\sum_{(u,v)\in A_0} y^l_{uv} = \sum_{v\in V} x_{vl} + 1 \qquad \forall l \in L \qquad (3.102)$$

$$\sum_{(u,v)\in A_0} t_{uv} \cdot y^l_{uv} \leq \hat{t} \qquad \forall l \in L \qquad (3.103)$$

$$x_{vl} \in \{0,1\} \qquad \forall l \in L,\ v \in V \qquad (3.104)$$

$$y^l_{uv} \in \{0,1\} \qquad \forall l \in L,\ (u,v) \in A_0 \qquad (3.105)$$

$$f^l_{uv} \in \mathbb{R}^+ \qquad \forall l \in L,\ (u,v) \in A_0 \qquad (3.106)$$

As in the compact model, the objective function (3.89) to be maximized is the total number of performed station visits. The maximum number of station visits per vehicle are bounded upwards by β (3.90), the optimal solution or rounded down upper bound of the single-vehicle case for which the MIP model is given in Section 3.6.2. Inequalities (3.91) state that any station visit can only be performed by at most one vehicle. Equations (3.92) ensure that for every vehicle the number of assigned delivery station visits corresponds to the number of assigned pickup station visits. Inequalities (3.93) ensure that the $i+1$-th visit of a station can only be performed when an i-th visit takes place. Equalities (3.94) and (3.95) state that each vehicle leaves node 0 once and arrives at $0'$ once, respectively. Assignment variables x_{vl} are linked with the arc selection variables y^l_{uv} by inequalities (3.96) and (3.97). It is ensured that an arc (u, v) can only be used in the arborescence if both $u \in V$ and $v \in V$ are assigned to vehicle l. Note that these inequalities are in principle redundant because it is also implicated by the constraints for the flow conservation but we include them nevertheless as they might be helpful from a computational point of view.

The arborescence is realized by the single commodity flow conservation equations (3.98)–(3.102). According to (3.98) the amount of flow sent out from the depot at node 0 corresponds to the number of nodes assigned to vehicle l plus one to also reach $0'$, i.e., to get back to the depot. The consumption of this last unit of flow at $0'$ is ensured by (3.99). Equalities (3.100) provide the flow conservation for all station visits $v \in V$, where one unit of flow is consumed by each station visit assigned to vehicle $l \in L$. Inequalities (3.101) link the flow variables with the arc selection variables y^l_{uv}, i.e., a positive flow may only occur on a selected arc. Equations (3.102) state for each arborescence that the total number of arcs must be one more than the total number of nodes, i.e., station visits assigned to vehicle $l \in L$. Inequalities (3.103) ensure that for each vehicle the approximated routing durations, i.e., the total times of the arborescence, lie within the allowed time budget $\hat{t}$. Finally (3.104)–(3.106) are the domain definitions of the decision variables. Variables x_{vl} and y^l_{uv} are binary whereas the flow variables f^l_{uv} are continuous.

Subproblems: A solution to the master problem yields an assignment of stations to vehicles in variables x_{vl}. Let $G_l = (V_l, A_l)$ with node set $V_l = \{v \mid v \in V, x_{vl} = 1\}$ and arc set $A_l = \{(u, v) \mid (u, v) \in A, x_{ul} = 1, x_{vl} = 1\} \cup \{(0, v) \mid v \in V^{\text{pic}}, x_{vl} = 1\} \cup \{(v, 0) \mid v \in V^{\text{del}}, x_{vl} = 1\}$ be the corresponding subgraph for vehicle $l \in L$. The *subproblem* (SP) in our LBBD corresponds then to the task of finding for each vehicle $l \in L$ in its corresponding subgraph G_l a Hamiltonian path from 0 to $0'$ visiting each node

$v \in V_l \cup \{0, 0'\}$ exactly once and having a total duration that does not exceed $\hat{t}$. Thus, our Benders subproblem decomposes into $|L|$ independent Hamiltonian path problems that are essentially decision variants of the ATSP, when considering that nodes 0 and $0'$ actually represent the same depot and might be further connected with an arc $(0', 0)$.

As sophisticated solvers for the TSP exist, we utilize one of them in our solution approach instead of implementing one on our own: *Concorde* [2, 30] is a state-of-the-art TSP solver for the symmetric traveling salesman problem (STSP) on complete graphs. We convert each of our directed ATSP instances into an STSP instance by employing the 2-node transformation described by Jonker and Volgenant [77, 78]. A symmetric auxiliary graph $G^{\text{aux}} = (V^{\text{aux}}, E^{\text{aux}})$ with associated costs $t^{\text{aux}} : E^{\text{aux}} \rightarrow \mathcal{R}^+$ is derived. Its set of vertices consists of two nodes for each one in V_l and two nodes 0 and $0'$ representing the depot: $V^{\text{aux}} = \{v \mid v \in V_l\} \cup \{v' \mid v \in V_l\} \cup \{0, 0'\}$. As Concorde works on a complete graph, we set $E^{\text{aux}} = V^{\text{aux}} \times V^{\text{aux}}$ and define the edge costs as follows:

$$t^{\text{aux}}_{vv'} = 0 \qquad \forall v \in V_l \tag{3.107}$$

$$t^{\text{aux}}_{uv} = t^{\text{aux}}_{u'v'} = \infty \qquad \forall u, v \in V_l,\ u \neq v \tag{3.108}$$

$$t^{\text{aux}}_{uv'} = t_{uv} + M \qquad \forall (u, v) \in A_l \tag{3.109}$$

$$t^{\text{aux}}_{uv'} = \infty \qquad \forall u, v \in V_l,\ u \neq v,\ (u, v) \notin A_l \tag{3.110}$$

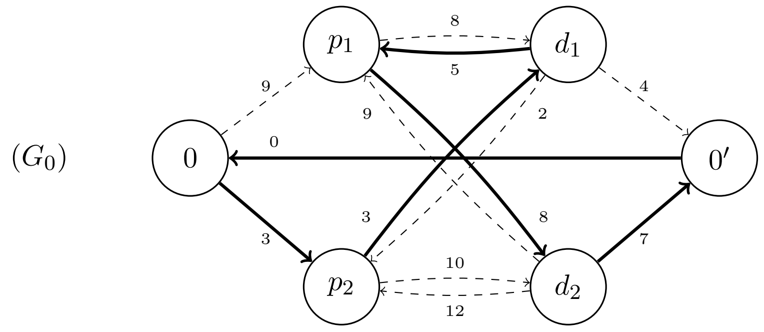
Figure 3.11 shows the derivation of the auxiliary graph on an example. Note that the big-M is needed to ensure that the zero-cost edges between all nodes and their duplicates (v, v') are always used in an optimal solution for the converted STSP. Thus, it has to be ensured that the big-M constant is large enough such that this property is guaranteed.

Proposition 3. *There is a one-to-one correspondence between optimal solutions to the converted STSP with finite objective and optimal solutions for the ATSP.*

Proof. Let $\mathcal{C}$ be the set of all Hamiltonian cycles in G^{aux} which contain (v, v') for all $v \in V_l$ and $C^{\text{aux}} \in \mathcal{C}$. We define the following corresponding subgraph of G_l, for which we will prove that it is a Hamiltonian cycle:

$$C = \{(u, v) \mid u, v' \in C^{\text{aux}}, u \neq v\} \cup \{(v, u) \mid u', v \in C^{\text{aux}}, u \neq v\}.$$

Due to the fact that the cost between all edges from the original graph (u, v) and all edges between duplicates (u', v') are set to infinity, they will never be part of C^{aux}. Thus, there are two types of edges contained in C^{aux}: $(u, v') \in E^{\text{aux}}$ representing an outgoing arc from node u in G_l and $(u', v) \in E^{\text{aux}}$ representing an ingoing arc to node u in the original graph G_l. As every node u has degree two in C^{aux} and is connected with u' there must be exactly one $v \neq u$ where v' is connected with u. The same way there exists exactly one $w \neq u$ which is connected to u'. Consequently, every node has exactly one ingoing and exactly one outgoing arc in C. Since the undirected version of C is exactly C^{aux} after merging all vertices v with v', it can be concluded that C is weakly connected. Since it was shown that every node has exactly one ingoing and exactly one outgoing arc and C is weakly connected, consequently, C has to be a Hamiltonian cycle



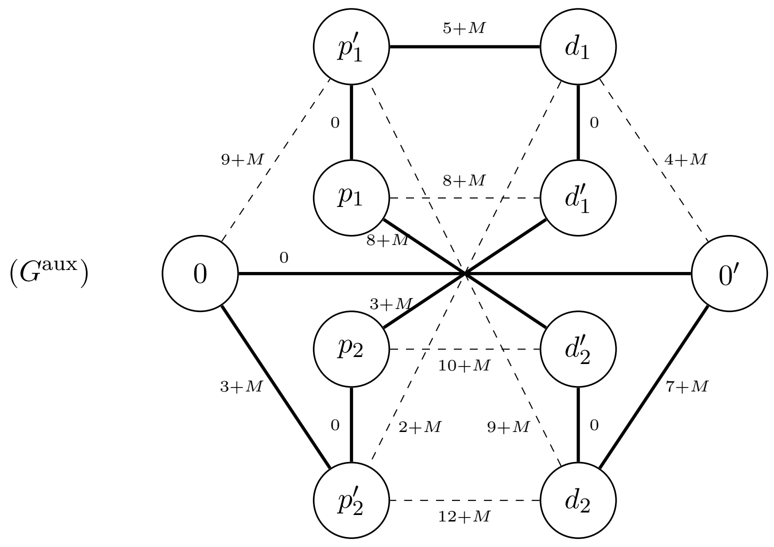


Figure 3.11: An example for the conversion of our subproblem on subgraph G_l into a symmetric traveling salesman problem instance on an auxiliary graph G^{aux}. Pickup stations are referred by p_1 and p_2, d_1 and d_2 denote delivery stations, 0 is the depot and $0'$ is the copy of the depot. Note, that G^{aux} actually is a complete graph. However, infeasible edges with $t_{uv} = \infty$, $\forall (u, v) \in G^{\text{aux}}$ are omitted for the sake of readability. The optimal solution in G^{aux} and the corresponding optimal solution in G_l are drawn as bold, green edges respectively arcs.

Algorithm 3.1: LBBD for BBSS

```
1:  repeat
2:    init: $r \leftarrow$ vector of $|L|$ empty routes, $cutsAdded \leftarrow$ false
3:    Solve MP to obtain subproblems
4:    for all $l \in L$ do
5:      $r_l \leftarrow$ solution of SP for vehicle $l$
6:      if $obj(r_l) > \hat{t}$ then
7:        $I \leftarrow V(r_l)$
8:        MP $\leftarrow$ MP $\cup$ $(\sum_{v \in I} x_{vl} \leq |I| - 1 \quad \forall l \in L)$
9:        $cutsAdded \leftarrow$ true
10:     end if
11:   end for
12: until not($cutsAdded$)
13: return $r$
```

in G_l. Moreover, if C is a Hamiltonian cycle in G_l we can construct the corresponding Hamiltonian cycle C^{aux} in G^{aux}. Therefore, we have a bijection between all Hamiltonian cycles in G_l and all Hamiltonian cycles in $\mathcal{C}$. The objectives of these Hamiltonian cycles is the same except a constant:

$$t_C = t_{C^{\text{aux}}} - (|V| + 2) \cdot M.$$

Therefore, if C^{aux} is optimal, the corresponding C also has to be optimal. Moreover, if C is optimal, it follows that C^{aux} has a minimum objective of all Hamiltonian cycles in $\mathcal{C}$. By construction all optimal Hamiltonian cycles of G^{aux} have to be in $\mathcal{C}$ and therefore, C^{aux} is optimal. $\square$

An optimal TSP solution on graph G^{aux} will always connect node 0 to a visit of a pickup station $v \in V^{\text{pic}}$ since the costs for traveling from the depot 0 to a delivery station is infinity. Moreover, when a pickup station has been visited the next visit can only be performed at a delivery station $v \in V^{\text{del}}$ since costs for traveling between two pickup stations is also infinity. The same condition holds for traveling between two delivery stations. Traveling to the copy of the depot $0'$ can only be performed from a delivery station since the costs for traveling from a pickup station to the copy of the depot is infinity. Finally, the Hamiltonian path can be obtained by simply excluding the arc between 0 and $0'$ which is performed at no cost.

Iterated Decomposition Procedure and Cut Generation: Algorithm 3.1 shows an *LBBD* scheme utilizing cut generation by Benders infeasibility cuts. Variable r denotes the current solution, i.e., the vector of $|L|$ routes, which are initially all empty. The function $obj(r_l)$ returns the objective value of a single subproblem solution, i.e., the actual routing costs when the TSP is solved to optimality.

Algorithm 3.2: Minimize the cutset

```
init: $r_l \leftarrow$ Hamiltonian Path for vehicle $l$ in $G_l$, $\mathcal{I} \leftarrow \{I\}$, $MinSize \leftarrow |I|$
function $minimizeCutSet(r_l, \mathcal{I}, MinSize)$
 1: for all $(u,v) \in r_l \mid u \notin \{0,0'\}, v \notin \{0,0'\}$ do
 2:   $T \leftarrow V(r_l) \setminus \{u,v\}$
 3:   $r'_l \leftarrow$ solution of SP for stations in $T$
 4:   if $obj(r'_l) > \hat{t}$ then
 5:     if $|T| = MinSize$ then
 6:       $\mathcal{I} \leftarrow \mathcal{I} \cup \{T\}$
 7:     else if $|T| < MinSize$ then
 8:       $\mathcal{I} \leftarrow \{T\}$
 9:       $MinSize \leftarrow |T|$
10:     end if
11:     $\mathcal{I} \leftarrow minimizeCutSet(r'_l, \mathcal{I}, MinSize)$
12:     $MinSize \leftarrow |I'|, I' \in \mathcal{I}$
13:   end if
14: end for
15: return $\mathcal{I}$
```

The MP is solved in Line 3 and the assignment of stations to vehicles is retrieved. We get our subproblems which are solved in the corresponding loop (4) for each vehicle separately. For every solution to a subproblem we utilize a solution cache. This means, that if a subproblem is feasible its corresponding Hamiltonian path and the routing costs are cached for later use. If the subproblem is infeasible it is not going to be cached because those subproblems result in a cut for the master problem. If we cannot find the subproblem in our solution cache, then a single subproblem is solved by Concorde (5) and added to the current solution r as r_l. In the subproblem the routing costs are minimized and if this objective value is greater than the maximal time budget of the vehicles, we found an infeasible assignment (6). Let $I = \{r_l^1, r_l^2, \ldots, r_l^{\rho_l}\}$ be a set of station visits for which the minimal Hamiltonian path from 0 to $0'$ is greater than the time budget $\hat{t}$. Then, we can build infeasibility cuts of the form

$$\sum_{v \in I} x_{vl} \leq |I| - 1 \quad \forall l \in L. \tag{3.111}$$

These cuts are created for each vehicle $l \in L$ and added to the MP. They imply that the simultaneous assignment of the station visits in I – and all supersets of I – to any of the vehicles is prohibited in subsequent master problem instances.

To make this cut as strong as possible, we try to *minimize* the *infeasible* set I of station visits, which is derived from all currently assigned stations (7) by Algorithm 3.2. Loop (1) iterates over all edges of a given Hamiltonian path $r_l = \{0, r_l^1, \ldots, r_l^{\rho_l}, 0'\}$ so that all possible options for minimizing the cutset are evaluated. We extract nodes u and v from the Hamiltonian path and refer the remaining set as T (2). Two station visits have to be removed because the number of pickup and delivery station visits have to be equal in oder to obtain a feasible route. As edges can only exist between alternating station types it is ensured that only one pickup and one delivery station visit is removed. Here again, we utilize the proposed solution cache so that previously evaluated sets of station visits may not be evaluated multiple times. If the subproblem cannot be found in the solution cache, the subproblem of finding a Hamiltonian path for the reduced set of station visits in T is solved (3) and the routing costs are checked for feasibility (4). If the set T is infeasible we either found an additional cut (5) with equal size of station visits as the previous found cut(s) or we found a new cut containing less station visits than all previously found cuts (7). If the routing costs are feasible we did not find any new cut and do not have to explore this branch of the search tree further. If the routing costs have been infeasible, we recursively call the function *minimizeCutSet* (11) to check all subsets of I which are candidates for a smaller cutset. At the end the set I contains the smallest possible cutset(s) based on the initial one. It is also possible that I contains more than one cut because multiple minimum cutsets may exist.

We can perform this algorithm because the subproblem is solved very efficiently by the Concorde TSP solver.

Vehicle-spanning cuts: Due to the following observation we came up with the idea of also computing vehicle-spanning cuts instead of only utilizing cuts only for a single vehicle: Throughout the algorithm, a Benders infeasibility cut is added to the MP whenever an infeasible vehicle assignment is generated. In a new iteration of the master problem the MIP solver often tries to move station visits among different vehicles – because then these new assignments constitute different vehicles than in the cutset – although it may not be possible to come toward a feasible solution this way. The total (optimal) routing costs over all vehicles may be larger than the total amount of time budget provided by all available vehicles.

The idea is to solve the LP relaxation of the following MIP formulation:

$$\min \quad \sum_{l\in L} \sum_{(u,v)\in A_0} y^l_{uv} \cdot t_{uv} \tag{3.112}$$

$$\text{s.t.} \quad \sum_{l\in L} x_{vl} = 1 \qquad \forall v \in V \tag{3.113}$$

$$\sum_{v\in V_{\text{pic}}} x_{vl} = \sum_{v\in V_{\text{del}}} x_{vl} \qquad \forall l \in L \tag{3.114}$$

$$\sum_{l'\in L} x_{(s,i)l'} \geq x_{(s,i+1)l} \qquad \forall s \in S, l \in L, i = 1,\ldots,f_s - 1 \tag{3.115}$$

$$\sum_{v\in V_{\text{pic}}} y^l_{0v} = 1 \qquad \forall l \in L \tag{3.116}$$

$$\sum_{v\in V_{\text{del}}} y^l_{v0'} = 1 \qquad \forall l \in L \tag{3.117}$$

$$\sum_{(u,v)\in A_0} y^l_{uv} = x_{ul} \qquad \forall l \in L, u \in V \qquad (3.118)$$

$$\sum_{(u,v)\in A_0} y^l_{uv} = x_{vl} \qquad \forall l \in L, v \in V \qquad (3.119)$$

$$\sum_{(u,v)\in A_0} y^l_{uv} = \sum_{(v,u)\in A_0} y^l_{vu} \qquad \forall l \in L, v \in V \qquad (3.120)$$

$$a_u - a_v + |V| \cdot y^l_{uv} \leq |V| - 1 \qquad \forall l \in L, (u,v) \in A \qquad (3.121)$$

$$x_{vl} \in \{0,1\} \qquad \forall l \in L, v \in V \qquad (3.122)$$

$$y^l_{uv} \in \{0,1\} \qquad \forall l \in L, (u,v) \in A_0 \qquad (3.123)$$

$$1 \leq a_v \leq |V| \qquad \forall v \in V \qquad (3.124)$$

This LP relaxation already provides reasonable lower bounds so that at least some of these vehicle-spanning cuts can be generated. Thus, we take the reduced set of station visits as solution from the MP but breakup the assignment to the vehicles and compute the minimal routing costs resulting from an optimal assignment. We therefore adjusted inequalities (3.91) from the master problem to the following equalities: $\sum_{l\in L} x_{vl} = 1 \; \forall v \in V$, and changed the objective function to minimize the total routing costs over all vehicles, i.e., min $\sum_{l\in L} \sum_{(u,v)\in A_0} y^l_{uv} \cdot t_{uv}$. If these routing costs are higher than the available time budget of all vehicles together, the set of station visits is not able to produce a feasible solution in any constellation of assignments. Let h_l denote the minimal computed routing costs for the reduced set of station visits of vehicle $l \in L$, then we can add a vehicle-spanning cut iff

$$\sum_{l\in L} h_l > |L| \cdot \hat{t}. \qquad (3.125)$$

Let I denote the set of stations used in the currently considered assignment, i.e., $I = \{v \mid x_{vl} = 1, \forall v \in V, l \in L\}$. Assume, that inequality (3.125) holds for this assignment. Then formally, the cut is defined as follows:

$$\sum_{l\in L} \sum_{v\in I} x_{vl} \leq |I| - 1. \qquad (3.126)$$

Branch-and-Check

As an alternative to the classical (logic-based) Benders decomposition method described in the previous section, we also consider a corresponding Branch-and-Check (BAC) approach. The term BAC has been originally proposed by Thorsteinsson in [150] and refers essentially to a classical branch-and-cut, which, however, is re-interpreted in terms of the Benders decomposition.

Instead of completely resolving the master problem after adding new Benders cuts in each iteration, BAC essentially solves the master problem only once and adds Benders cuts on the fly: BAC starts by solving the MIP model of the original master problem (possibly enhanced by a relaxation of the subproblem) in the usual branch-and-cut fashion.

When an optimal solution is identified, the subproblem is solved by an auxiliary method and corresponding Benders cuts are derived as in the original Benders decomposition method. Now in BAC, however, these Benders cuts are dynamically added to the currently considered master problem within the branch-and-cut, effectively cutting off the current solution, and the branch-and-cut process is continued. Any obtained so far optimal solution to the master problem is “checked” in this way, and the master problem correspondingly augmented by Benders cuts until at some point the obtained solution turns out to be feasible and optimal also for the original problem; no further Benders cuts need then to be added, and the whole BAC can be terminated.

In our implementation of BAC we use again the MIP model proposed in Section 3.6.2. Remember that this model already includes the 0-arborescence problems as a relaxation of the subproblems. The subproblems are solved as described before via Concorde and the derivation of Benders Cuts follows exactly the already described way.

As an improvement to basic LBBD, the Benders subproblem is solved not only for so-far optimal master problem solutions, but for any encountered feasible master problem solution resulting in more cuts than the LBBD-based approach. The larger number of earlier added cuts turned out to be beneficial in the sense that the overall number of required branch-and-bound nodes and the overall runtime were reduced.

A particular advantage of this BAC is that it is in general able to yield upper as well as lower bounds and corresponding feasible solutions to the BBSS problem already early during the optimization. In contrast, classical Benders decomposition with infeasibility cuts is not directly able to provide lower bounds earlier than at the very end, as the master problem variables will not get overall feasible assignments before.

Variable Neighborhood Search

For comparison purposes and to further study the impact of the BBSS problem simplification by only considering full loads, we use here the VNS proposed by Rainer-Harbach et al. [125]. This VNS uses remove-station, insert-unbalanced-station, intra-route-2-opt, replace-station, intra-or-opt, 2-opt*-inter-route-exchange and intra-route 3-opt neighborhood structures for local improvement within an embedded Variable Neighborhood Descent (VND), and for shaking move-sequence, exchange-sequence, destroy-&-recreate, and remove-stations operations. The only modification we applied concerns the objective function, in which we set the weighting factors ω^{work} and ω^{load} for the additional terms to consider tour lengths and loading instructions to zero, in order to follow the same single goal of maximizing the balance gain as we do in our new approaches. Furthermore, as the balance gain is expressed in the VNS in terms of the number of bikes and in our case here in station visits, we scale the VNS's objective values accordingly by dividing them by the vehicle capacity.

3.6.3 Computational Results

We have done our computational tests on a rigorous benchmark suite derived from [125] with instances up to 70 and 120 stations for the multi-vehicle and the single-vehicle cases respectively. An instance is primarily characterized by the tuple of the number of stations, the number of vehicles, and the time limit $(|V|, |L|, \hat{t})$. All algorithms have been implemented in C++ and have been compiled with g++ 4.8.4. As MIP solver we used CPLEX 12.6.3 branch-and-cut with default parameters except limiting the number of used threads to one for a better comparability and restricting the maximum size of the branch-and-cut tree to 3GB as this is the amount of memory available to one cluster node in our configuration. All tests have been performed as single threads on an Intel Xeon E5540 2.53GHz Quad Core processor.

For our tests with multiple vehicles we use instances of 10, 20, 30, 40, 50, 60 or 70 stations, two or three vehicles, and a time budget of either 120, 240, or 480 minutes. For all instances we employed a maximum CPU time limit of 1 hour. For every combination $(|V|, |L|, \hat{t})$, we considered 30 different randomly generated instances which are publicly available on the web[4]. These instances have been derived from the real-world scenario at Citybike Wien, Vienna's major PBS in the following way. Citybike Wien gave us historic data sets from year 2011 when the system consisted of 92 renting stations—now the system has already 120 stations. We enlarged the dataset by 664 artificial stations placed among the city of Vienna. We have derived travel times t_{uv} by an estimation of the real-world travel time and further including an estimation for the time needed to park at the station, i.e., some stations may be better reachable than others. For the existing stations we, of course, chose their original capacity C_s whereas for the artificial ones we have chosen the capacities at random. The initial fill level of stations p_s has been taken from a snapshot of 2011 for the existing stations. For the artificial stations we first chose for some of them initial fill levels at random according to a distribution we observed at the real stations so that geographically near stations have a similar fill level. For the other artificial stations we used an interpolation for determining their initial fill level. We set the target values q_s in such a way that there are multiple full vehicle loads needed in order to bring the system in a balanced state. All tests inhere have been performed by considering a full vehicle load consisting of $Z = 20$ bikes. Of course, our approach works with arbitrary sizes of full vehicle loads, and how an instance for our new problem formulation is derived by the given data, is explained in the next paragraph. For every instance set we have chosen a corresponding subset of the 756 available stations where we first chose a station at random and then selected the $|S| + 1$ nearest stations. One station was randomly selected as the depot for the instance.

Following our new approach, we only consider the movement of full vehicle loads, i.e., Z bikes from one station to another. Consequently, we derived each station's demand f_s of full vehicle loads as defined in equations (3.58). Remember that delivery stations S_{del} have negative demand values f_s, while pickup stations are given by positive values;

[4]`https://www.ac.tuwien.ac.at/files/resources/instances/bbss/benchs.tar.gz`

stations with $|p_s - q_s| < Z$ do not need to be considered in our model and are therefore discarded. By above definition, it is ensured that we never move more than $|p_s - q_s|$ bikes to a station and thus never exceed a station's capacity.

In the following we analyze the results. Table 3.6.3 shows for every instance configuration results for the analyzed approaches: Logic-based Benders decomposition (LBBD) and its variant Branch-and-Check (BAC), compact MIP (see Section 3.6.1), and the variable neighborhood search (VNS) and the hop-indexed MIP from [125]. The column #best shows the number of instances where the particular approach achieved the best objective value among the 5 considered approaches. For BAC, LBBD and the compact MIP we show also the number of instances for which the corresponding approach returned a proven optimal solution #opt according to the new problem formulation. Moreover, we also report the average objective value $\overline{obj}$ which corresponds to the station visits respectively twice the number of moved full vehicle loads. For BAC, LBBD and the compact MIP we take the optimal solution if available. If the optimal solution was not found we use the best integer solution found so far for BAC and the compact MIP. If also no integer solution was found we use 0 as the objective value for the corresponding instance. In the case of LBBD we take either the optimal solution or 0 because for this approach no bounds can be derived if the approach does not end up with a proven optimal solution. For VNS and miphop we use the original objective function as defined in [125] and shown in equation (3.54), only setting the weighting factors to $\omega^{\text{bal}} = 1$, $\omega^{\text{load}} = 0$, $\omega^{\text{work}} = 0$ accordingly, as already said we only consider balance optimization here. For comparing our new approaches with the VNS and miphop we use the following value:

$$\frac{\sum_{l \in L} \sum_{i=1}^{\rho_l} |y^i_{l,r^i_l}|}{Z}. \tag{3.127}$$

This value corresponds to twice the number of full vehicle loads moved among the stations and its average over a particular instance group is reported as $\overline{\text{obj}}$ in Table 3.6.3 for VNS and miphop.

In Section 3.6.3 we discuss the potential loss when considering only full vehicle loads. Then, in Section 3.6.3 we point out the advantages of our decomposition approaches and analyze the number of cuts and iterations of LBBD and its variant BAC and we analyze the approximation quality of the 0-arborescence in the master problem. Section 3.6.3 analyzes the single-vehicle case of our new problem formulation.

Analyzing the Impact of Considering Full Vehicle Loads Only

We first want to gain an approximate understanding of the loss in solution quality we obtain by moving from a previous *"detailed"* model, in which the loading and unloading of an arbitrary number of bikes is allowed, to our simplified model that considers only full vehicle loads.

For comparison in Table 3.6.3 we use a hop-indexed MIP model as well as one of the leading metaheuristic approaches, which is the VNS introduced in Section 3.6.2. Both approaches have been proposed in [125]. In the following we will concentrate on the VNS and BAC because these are the most competitive representatives of the two different problem formulations.

Remember that the VNS is not limited to full vehicle loads. It tries to find a tour together with best possible numbers of vehicles to load and unload at each stop. We just scaled the final overall balance gain, i.e., the sum of all loading instructions at each stop of every vehicle $\sum_{l \in L} \sum_{i=1}^{\rho_l} |y^i_{l,r^i_l}|$, by dividing it by the vehicle capacity Z, cf. equation (3.127), to make it comparable to the number of full vehicle loads by which we measure the balance gain with respect to the number of moved bikes in our new model.

When analyzing the results in Table 3.6.3 one can see that the average objective values obtained by BAC, which provided the best results for our new problem formulation, and the VNS correspond closely. Obviously, the simplification of considering only full vehicle loads has on these instances only a very minor impact. In fact, we could observe that also the solutions identified by the VNS also almost always transported only full vehicle loads from one station to another. Obviously, this only holds under our fundamental assumption that a complete balance with objective value zero, i.e., where all station demands are fulfilled, is not achievable within the limited working time – and also not necessary in practice. It can be observed that BAC yields more often the overall best solution—among the five different approaches—than the VNS (693 versus 618), and a Wilcoxon signed-rank test comparing BAC with the VNS on each instance set shows significant advantages with error probabilities of less than 5% for 12 of the 30 classes for BAC whereas the VNS has significant advantages on 5 of the 30 classes.

Instance set			BAC				LBBD				Compact				VNS			miphop		
$\|S\|$	$\|L\|$	$\hat{t}$	#best	#opt	$\overline{\text{obj}}$	$\widetilde{\text{time}}$ [s]	#best	#opt	$\overline{\text{obj}}$	$\widetilde{\text{time}}$ [s]	#best	#opt	$\overline{\text{obj}}$	$\widetilde{\text{time}}$ [s]	#best	$\overline{\text{obj}}$	$\widetilde{\text{time}}$ [s]	#best	$\overline{\text{obj}}$	$\widetilde{\text{time}}$ [s]
30	2	120	**30**	30	8.00	0.6	**30**	30	8.00	0.5	**30**	29	8.00	0.3	29	7.93	3600.0	**30**	8.00	10.4
30	2	240	**26**	27	18.80	23.2	23	27	17.00	13.8	24	25	18.67	19.2	14	17.87	3600.0	21	18.47	3600.0
30	2	480	11	30	26.47	1.0	11	30	26.47	0.5	2	30	25.33	0.2	**23**	28.87	3600.0	8	25.93	486.0
30	3	120	**29**	30	11.93	1.4	**29**	30	11.93	1.1	**29**	22	11.93	670.5	26	11.67	3600.0	**29**	11.93	35.7
30	3	240	**22**	24	25.27	35.5	18	24	20.33	28.0	17	30	24.80	4.1	20	25.00	3600.0	3	19.80	3600.0
30	3	480	11	30	26.47	3.6	11	30	26.47	0.4	2	30	25.33	0.4	**23**	28.87	3600.0	10	26.52	466.6
40	2	120	**30**	30	8.07	1.5	**30**	30	8.07	1.5	**30**	22	8.07	188.3	29	8.00	3600.0	29	8.00	33.4
40	2	240	23	25	19.60	44.8	23	26	17.27	82.4	**24**	20	19.67	43.6	11	18.53	3600.0	17	18.87	752.8
40	2	480	21	26	35.00	25.1	18	26	30.27	4.7	8	29	33.87	0.9	**24**	35.47	3600.0	0	7.38	3600.0
40	3	120	**30**	29	12.07	3.3	29	29	11.60	2.8	**30**	9	12.07	1146.8	25	11.73	3600.0	29	12.00	85.1
40	3	240	**24**	11	27.60	3600.0	8	8	7.93	3600.0	21	15	27.33	737.1	12	26.33	3600.0	5	20.07	3600.0
40	3	480	9	30	35.73	12.0	9	30	35.73	1.6	0	30	34.20	0.9	**23**	38.33	3600.0	5	23.67	3600.0
50	2	120	28	30	7.80	3.5	28	30	7.80	3.4	**29**	16	7.93	1108.8	**29**	7.93	3600.0	**29**	7.93	82.8
50	2	240	**23**	21	19.60	295.3	19	20	13.40	302.8	22	12	19.53	1383.6	21	19.27	3600.0	15	18.27	1518.9
50	2	480	**22**	5	38.40	3600.0	8	9	11.80	3600.0	21	23	38.40	114.2	17	37.87	3600.0	0	2.73	3600.0
50	3	120	28	29	11.67	7.4	28	29	11.40	10.8	**29**	9	11.87	1852.7	28	11.80	3600.0	**29**	11.87	198.5
50	3	240	**23**	9	28.13	3600.0	7	8	7.80	3600.0	21	7	27.93	1015.2	14	27.07	3600.0	4	14.33	3600.0
50	3	480	3	30	44.07	22.2	3	30	44.07	5.5	0	30	42.67	2.3	**30**	48.53	3600.0	0	4.33	3600.0
60	2	120	**29**	30	8.00	6.1	28	29	7.67	4.8	**29**	13	8.00	2362.9	**29**	8.00	3600.0	28	7.93	111.9
60	2	240	26	24	20.07	336.4	26	26	17.67	328.7	**27**	10	20.13	3589.9	14	19.00	3600.0	17	18.40	2410.3
60	2	480	**23**	2	40.00	3600.0	1	1	1.33	3600.0	20	17	39.80	604.0	12	38.33	3600.0	0	0.00	3600.0
60	3	120	**29**	28	11.93	17.1	28	28	11.20	16.6	**29**	5	11.93	3592.0	26	11.80	3600.0	28	11.60	269.2
60	3	240	**24**	11	28.67	3600.0	10	10	10.00	3600.0	22	3	28.53	1479.7	10	26.93	3600.0	0	5.93	3600.0
60	3	480	11	20	52.00	246.7	5	17	28.80	974.6	6	21	51.20	21.9	**21**	53.20	3600.0	0	0.00	3600.0
70	2	120	**30**	30	8.00	10.2	**30**	30	8.00	9.0	**30**	13	8.00	1255.9	**30**	8.00	3600.0	**30**	8.00	197.2
70	2	240	**28**	24	20.40	344.2	23	23	15.40	308.2	**28**	10	20.40	1826.4	17	19.60	3600.0	11	16.80	3326.7
70	2	480	20	3	41.17	3600.0	1	1	1.33	3600.0	**23**	10	41.33	928.6	9	39.53	3600.0	0	0.00	3600.0
70	3	120	**30**	30	12.00	27.9	**30**	30	12.00	25.6	**30**	2	12.00	3586.2	29	11.93	3600.0	**30**	12.00	439.8
70	3	240	**27**	16	29.60	2155.5	14	14	14.00	3600.0	24	1	29.40	2559.9	11	27.93	3600.0	0	1.00	3600.0
70	3	480	**23**	3	56.48	3600.0	1	3	5.47	3600.0	15	5	55.73	1103.5	12	55.13	3600.0	0	0.00	0.0
		Total	**693**	667			529	658			622	498			618			407		

Table 3.10: This table provides a full comparison among the various discussed approaches.

Instance set	**BAC**			**LBBD**			**Compact**		
$\lvert S \rvert$	#best	#opt	$\widetilde{\text{time}}\,[s]$	#best	#opt	$\widetilde{\text{time}}\,[s]$	#best	#opt	$\widetilde{\text{time}}\,[s]$
10	180	**180**	0.1	180	**180**	< 0.1	92	177	< 0.1
20	180	**180**	0.4	179	**180**	0.2	110	**180**	0.1
30	178	**171**	2.2	170	**171**	0.9	135	166	0.8
40	176	**151**	12	149	149	4	133	125	36
50	174	124	89.8	124	**126**	62.2	151	97	558.9
60	171	**115**	343	111	111	352.4	155	69	1422.7
70	169	**106**	459.4	101	101	336.5	159	41	1673.6
Total	1228	**1027**		1014	1018		935	855	

Table 3.11: Additional aggregation provided by aggregating only over the number of nodes for logic-based Benders decomposition, its variant Branch-and-Check, and the compact mixed-integer linear programming model.

However, we have to note that the VNS often finds very fast high-quality solutions whereas, at least for the larger instances, the MIP-based approaches, namely LBBD, BAC, and the compact formulation often need a lot of time for finding an integer solution or even solve the problem instance to proven optimality. Smaller instances or mid-size instances, according to our test suite, are however, often solved to proven optimality within a small amount of time.

We conclude that the disadvantages of the simplified model are very well compensated by the much better solvability of the new approach.

Analyzing Logic-Based Benders Decomposition and the Compact Model

In Table 3.6.3 additional aggregation of the results is given by providing results for each individual number of stations. The best results are achieved by utilizing BAC. This is also the approach which yields the highest number of proven optimal solutions among the analyzed dataset. As expected the median computation times are better by applying LBBD than solving the full compact model at once. This further emphasizes the importance of the decomposition approach we are introducing with this approach.

In Table 3.6.3 we measure various statistics about LBBD and BAC. Column $\overline{\text{approx}}$ shows the average approximation quality of the 0-arborescence as percentage value which is computed by $(t^{\text{est}} - t^{\text{opt}})/t^{\text{opt}}$, where t^{est} are the estimated routing costs by the 0-arborescence and t^{opt} are the optimal routing costs as obtained by the optimal solution for a subproblem. The average over all computed routes is taken. Column $\overline{\#\text{iter}}$ shows the average number of iterations performed per instance set whereas $\overline{\#\text{cuts}}$ shows the average number of generated cuts. For BAC we use a different nomenclature, namely $\overline{\#\text{calls}}$ as for this variant of LBBD we do not have iterations but we measure the calls to the LazyConstraintCallback where we generate the Benders infeasibility cuts. Moreover, columns $t^{\max}_{\text{master}}$ and $\overline{t_{\text{master}}}$ state the maximum time used to solve the master problem

Instance set			**LBBD**						**BAC**			
$\lvert S\rvert$	$\lvert L\rvert$	$\hat{t}$	$\overline{\text{approx}}$ [%]	$\overline{\#\text{iter}}$	$\overline{\#\text{cuts}}$	$t^{\max}_{\text{master}}$ [s]	$\overline{t}_{\text{master}}$ [s]	$t^{\max}_{\text{sub}}$ [s]	$\overline{\text{approx}}$ [%]	$\overline{\#\text{calls}}$	$\overline{\#\text{cuts}}$	$t^{\max}_{\text{sub}}$ [s]
10	2	120	1.1	1.1	0.3	0.2	0.1	< 0.1	1.8	1.4	0.3	< 0.1
10	2	240	1.3	1.0	0.0	0.1	< 0.1	< 0.1	1.4	1.0	0.0	< 0.1
10	2	480	1.3	1.0	0.0	< 0.1	< 0.1	< 0.1	1.6	1.0	0.0	< 0.1
10	3	120	0.5	1.1	0.4	0.3	0.1	< 0.1	1.3	1.2	0.1	< 0.1
10	3	240	1.0	1.0	0.0	< 0.1	< 0.1	0.1	1.3	1.1	0.0	< 0.1
10	3	480	1.0	1.0	0.0	< 0.1	0.1	0.1	1.4	1.0	0.0	< 0.1
20	2	120	1.5	1.4	1.0	4.5	0.2	< 0.1	2.1	1.8	0.7	< 0.1
20	2	240	2.3	5.4	13.1	37.2	1.2	0.1	2.3	6.2	11.7	0.2
20	2	480	3.4	1.0	0.0	0.1	< 0.1	0.1	2.5	1.1	0.0	0.6
20	3	120	1.4	1.8	4.3	15.3	0.8	< 0.1	2.0	2.4	3.3	< 0.1
20	3	240	2.0	1.2	0.5	0.5	0.2	< 0.1	1.9	1.1	0.0	< 0.1
20	3	480	2.4	1.0	0.0	0.1	0.1	< 0.1	2.2	1.0	0.0	< 0.1
30	2	120	1.2	1.4	0.9	1.4	0.4	< 0.1	1.8	1.7	0.6	0.1
30	2	240	1.8	11.6	44.6	1925.7	37.5	0.9	1.8	21.2	63.9	0.6
30	2	480	4.8	1.0	0.0	0.6	0.2	0.1	4.2	1.1	0.0	0.1
30	3	120	1.3	2.2	4.5	39.6	2.1	< 0.1	3.2	2.5	2.0	< 0.1
30	3	240	1.9	4.6	21.5	3600.0	298.6	0.8	2.0	13.7	59.5	0.5
30	3	480	3.4	1.0	0.0	0.9	0.2	0.2	3.2	1.1	0.0	0.3
40	2	120	1.8	1.3	0.6	20.8	1.6	< 0.1	3.0	2.2	1.1	< 0.1
40	2	240	1.8	20.0	77.2	3407.2	81.0	0.6	1.7	47.8	159.1	0.5
40	2	480	3.3	4.3	11.0	3012.2	69.0	0.9	2.7	6.1	12.0	1.4
40	3	120	1.7	1.9	3.6	3600.0	122.2	< 0.1	3.3	2.5	2.7	0.1
40	3	240	2.2	6.6	40.4	3600.0	1423.7	0.8	2.1	53.5	274.1	1.7
40	3	480	5.4	1.0	0.0	3.1	0.8	0.6	4.4	1.2	0.0	0.7
50	2	120	2.2	1.7	1.7	112.3	3.7	< 0.1	2.4	2.1	1.1	0.1
50	2	240	2.5	27.1	115.2	3600.0	379.4	1.0	2.2	95.3	306.8	1.2
50	2	480	2.9	14.9	64.3	3600.0	704.0	1.1	2.7	97.2	406.0	3.2
50	3	120	1.9	2.6	7.8	1679.5	44.2	< 0.1	2.8	3.5	7.2	0.1
50	3	240	2.6	6.8	44.8	3600.0	1171.4	0.3	2.4	49.9	267.8	0.7
50	3	480	5.4	1.0	0.0	5.2	2.8	0.5	4.0	2.1	0.1	0.3
60	2	120	1.9	2.1	2.6	988.7	25.3	< 0.1	4.6	2.7	2.1	0.2
60	2	240	2.1	12.0	38.7	2510.5	80.8	0.3	2.2	53.8	176.0	0.9
60	2	480	3.0	11.4	49.4	3600.0	1530.3	1.9	2.8	55.8	195.0	2.0
60	3	120	2.2	2.0	4.9	3600.0	168.1	< 0.1	3.4	4.6	9.3	0.2
60	3	240	2.6	6.7	40.0	3600.0	1036.4	0.5	2.1	36.0	173.9	0.8
60	3	480	3.5	1.6	4.4	3600.0	1006.9	0.5	2.9	18.4	77.4	4.0
70	2	120	1.6	1.3	0.6	17.7	6.8	0.1	5.3	2.0	0.9	0.2
70	2	240	1.6	2.6	5.9	3600.0	652.0	0.6	1.4	16.7	30.0	1.5
70	2	480	3.1	11.5	48.8	3600.0	1758.4	1.1	2.2	60.1	188.3	2.3
70	3	120	1.5	1.7	2.6	33.2	13.8	< 0.1	4.2	4.7	8.6	0.4
70	3	240	2.0	4.2	22.7	3600.0	1053.2	0.5	1.8	22.6	97.9	1.3
70	3	480	4.3	1.0	8.3	3600.0	2429.6	1.7	2.7	15.7	60.4	1.6
Average			2.3	4.5	16.3				2.6	17.1	61.9	

Table 3.12: Comparing number of cuts, iterations, approximation quality and the time needed to solve master problems as well as subproblems.

and the average time respectively. Column $t_{\text{sub}}^{\max}$ shows the maximum time which was needed to solve a single subproblem per instance set.

A result of this comparison is that subproblems are relatively easy to solve compared to the master problems. As seen in Table 3.6.3 the most difficult subproblem needed only 4 seconds to be solved whereas some of the the master problem instances could not be solved to optimality within 1 hour. What can also be seen is that the approximation via the 0-arborescence is tight. In many cases we have only 1 to 2 percent deviation from the optimal routing costs. Furthermore, as expected, subproblems for instances with low time budget, i.e., 2 hours, are very easy to solve and most often even do not need 0.1 seconds to be solved. BAC generates much more cuts as LBBD evaluates subproblems only after optimal solutions to the master problem have been found whereas BAC evaluates subproblems each time a feasible integer solution has been found. Although, BAC generates more cuts than LBBD, the quantity of cuts is no guarantee for success at all, but the quality of the cuts can improve solvability of the problem instance. A phenomenon we could observe is that often instances with 4 hours time limit are more difficult to solve and also produce more cuts/iterations. This is because the objective of the problem is only to maximize the number of station visits. Thus, multiple optimal solutions may exist, even if stations are near to another. As already said, as the approximation quality of the 0-arborescence is relatively tight one of the optimal solutions to an instance with a higher time limit, i.e., 8 hours can be obtained very fast. On the other hand there are instances with 4 hours time limit that are hard to solve because there may not be as many optimal solutions as for instances with 8 hours time limit, even for the smaller instances. What has been observed by the authors and which was the reason to introduce vehicle-spanning cuts (see also Section 3.6.2) is that the approach often needs many iterations to prove that there does not exist a solution with a given number of station visits but there could be many assignments with 2 visits less that are optimal.

Single-Vehicle Case

We have also performed computational tests on the single-vehicle case of the problem which we have solved by the compact MIP model introduced in Section 3.6.2. This is also of practical importance for cases where the whole service area is more statically partitioned into districts and individual drivers/vehicles are then solely responsible for dedicated districts. Computational results are shown in Table 3.6.3. As the problem becomes simpler when reducing the number of vehicles we have also provided results for instances with 90 and 120 stations. For the most difficult instances with 120 stations and a shift time of 4 hours for the driver we have been able to solve 13 out of 30 instances to proven optimality. For the remaining instances we have been able to provide results with an average optimality gap of about 5%.

3.6.4 Conclusions and Future Work

We have introduced and investigated a new problem formulation for BBSS derived from practical considerations as they appear, e.g., at Citybike Wien, which is computationally

Instance set		**Compact**					
$\vert V\vert$	$\hat{t}$	#opt	$\overline{\text{obj}}$	$\overline{\text{LB}}$	$\overline{\text{UB}}$	$\overline{\text{gap}}\,[\%]$	$\widetilde{\text{time}}\,[s]$
40	120	30	4.07	4.07	4.07	0.00	0.1
40	240	30	10.13	10.13	10.13	0.00	0.2
40	480	29	20.93	20.93	20.98	0.24	0.4
50	120	30	4.00	4.00	4.00	0.00	0.1
50	240	28	10.33	10.33	10.43	1.11	0.4
50	480	29	21.53	21.53	21.59	0.30	1.9
60	120	30	4.07	4.07	4.07	0.00	0.2
60	240	23	10.33	10.33	10.66	3.29	1.6
60	480	27	22.00	22.00	22.16	0.77	7.1
70	120	30	4.00	4.00	4.00	0.00	0.3
70	240	22	10.53	10.53	10.91	3.76	6.8
70	480	21	22.47	22.47	22.90	2.03	15.2
90	120	29	4.00	4.00	4.05	1.13	0.5
90	240	18	10.53	10.53	11.12	5.84	98.3
90	480	17	22.73	22.73	23.34	3.10	23.1
120	120	29	4.00	4.00	4.05	1.13	1.3
120	240	13	10.60	10.60	11.51	9.06	3120.2
120	480	19	23.27	23.27	23.88	2.76	369.6
	Average		12.20	12.20	12.44	1.92	

Table 3.13: Computational results for the single-vehicle case

substantially simpler to solve than previous BBSS formulations. The key observation is that an economic maintenance of a PBS rarely allows to bring all stations into perfect balance w.r.t. precisely specified target fill levels. In practice, usually "more rebalancing work actually exists than can be typically achieved" with the given number of vehicles and limited working time of the drivers. Therefore, vehicles almost always move full vehicle loads among the stations. Moving just very few bikes from one station to another is basically meaningless in practice.

While previous BBSS models always considered a rather fine-grained planning allowing the movement of arbitrary numbers of bikes, we restrict our rebalancing tours to the movement of full vehicle loads from the beginning. This restriction yields substantial simplifications in the overall model, and consequently, the model can be solved computationally significantly easier.

For solving this new model, we developed a compact MIP model, an LBBD approach, and a BAC variant of the latter. The LBBD was inspired by the cluster-first route-second method because the problem can naturally be split in an assignment part and a routing part. The integral subproblems turned out to essentially correspond to asymmetric TSPs, which we solve by the state-of-the-art TSP solver Concorde. From these, Bender's infeasibility cuts are derived and iteratively added to the assignment master problem. In the BAC, we modified the LBBD approach by solving the master problem only once and solving corresponding subproblems for any encountered feasible solution. This modification turned out to further improve the performance in many cases.

Experimental comparisons with a state-of-the-art VNS for a previous fine-grained BBSS model have shown that our new problem formulation has only a minor impact on the achievable solution quality. In fact, the advantages of the easier solvability clearly outweigh the theoretical restrictions introduced by allowing only full vehicle loads. A Wilcoxon signed-rank test showed that BAC compared to VNS has significant advantages with an error probability of less than 5% for 12 of the 30 instance sets.

With our LBBD we could solve instances up to 70 stations to proven optimality, which is a substantial step forward in comparison to previous work with exact approaches. For the single-vehicle problem, we have solved even larger instances up to 120 stations.

Clearly, the proposed compact MIP, LBBD, and BAC are not the only meaningful methods to approach the new simplified BBSS problem formulation exactly. State-of-the-art branch-and-cut solvers for diverse vehicle routing problem variants based on subtour-elimination constraints can likely be adapted and are then presumably strong if not superior competitors. Also column generation approaches based on some set covering formulation appear meaningful.

But also in a purely heuristic context for addressing larger problem instances with possibly thousands of stations, the simplified modeling approach appears very meaningful and opens diverse existing methods for vehicle routing problem variants to be adapted with moderate effort.

CHAPTER 4

Bike-Sharing Station Planning Problem

In this chapter, we discuss two works regarding the bike-sharing station planning problem (BSSPP). The first one computes optimal station locations with arbitrary (continuous) slot count whereas the second work takes into account predefined station configurations for each cell of the input. Moreover, the instances used in the second approach are based on real-world data.

4.1 Introduction

Finding a good combination of station locations and building these stations in the right size is crucial when planning a PBS as these stations obviously directly determine the satisfied customer demand in terms of bike trips, the arising rebalancing effort, and the resulting fixed and variable costs. Stations close to public transport, business parks, or large housing developments will likely face a high demand whereas stations in sparser inhabited areas will probably face a lower demand. However, also the station density and connectedness of the actual regions to be covered play crucial roles. Some solitary station that is far from any other station will most likely not fulfill much demand. Moreover, a clever choice of station locations might also exploit the natural demands and customer flows in order to keep the rebalancing effort and associated costs reasonable.

As PBSs are usually implemented in rather large cities the problem of finding optimal locations for rental stations and sizing these stations appropriately is challenging and manually hardly comprehensible. Thus, there is the need for computational techniques supporting this decision-making. Besides fixed costs for building the system, an integrated approach should also estimate maintenance and rebalancing costs over a certain time horizon such that overall costs for the operator can be approximated more precisely. It is further important to consider the customer demands in a time-dependent way because there usually exists a morning peak and an afternoon peak which is due to commuters, people going to work, and students. Between these peaks, the demand of the system is usually a bit lower. We refer to this problem as *Bike Sharing Station Planning Problem* (BSSPP). The objective we consider here is to determine for a specified total-cost budget and a separate fixed-cost budget a selection of locations where rental stations of an also to be determined size should be erected in order to maximize the actually fulfilled customer demand.

4.2 Related Work

There already exists some work which tries to find optimal station locations for BSSs, although mostly considering different aspects. Most of the previous approaches are not entirely coherent and consider different constraints and sometimes also different optimization goals which makes direct comparisons impossible. Many approaches utilize mixed integer (non-)linear programming (MIP) as core technology. However, for larger instances it is usually impossible to solve such models exactly. However, MIP technology can also be used inside a heuristic method.

To the best of our knowledge, Yang et al. [161] were the first who considered the problem in 2010. They relate the problem to *hub location problems*, a special variant of the well-known *facility location problem*, and propose a mathematical model for it. The considered objective is to minimize the walking distance by prospective customers, fixed costs, and, a penalty for uncovered demands. The authors solve the problem by a heuristic approach in which a first part of the algorithm tries to identify the location of rental stations and a second, inner part tries to find shortest paths between origin and destination pairs. The authors illustrate their approach by a small example consisting of 11 candidate cells for bike stations.

Lin et al. [91] propose a mixed integer non-linear programming model and solve a small example instance with 11 candidate stations by the commercial solver LINGO, and furthermore provide a sensitivity analysis.

Martinez et al. [95] propose a hybrid approach consisting of a hourly MIP model which is embedded into a heuristic approach. They aim to maximize the net revenue of the system and also consider rebalancing costs but the latter are not explicitly modeled but only estimated beforehand. The authors aim to solve a real-world problem in Lisbon but only consider a relatively small part of the city with 565 potential station locations. Results for four different scenarios with various parameters (e.g., willingness to use electric bikes, different fee systems) are shown.

Lin et al. [92] again use the same problem formulation as given in [91] and state a mixed-integer non-linear programming model for it. As they aim to compute a minimum inventory to fulfill a certain service level the model is non-linear and they conclude that it is not exactly solvable in practice and thus, propose a heuristic approach for solving the problem. The authors introduce a greedy algorithm which starts by opening all possible station candidates and building all possible bike lanes between stations. Then, they alternatively close bike stations and bike lanes that result in a largest cost reduction but are still possible to close with respect to the minimum service level requirement. This procedure is iterated until some termination criterion is met. For evaluating a solution Dijkstras shortest path algorithm is used to compute paths between all origin/destination pairs. Again an illustrative example on a small test instance with 11 possible station candidates is given and different scenarios are evaluated to provide a sensitivity analysis.

Saharidis et al. [134] propose a pure MIP formulation to the BSSPP in a case study for the city of Athens. They present a time-discretized model in one hour steps and aim to minimize the total walking time of the users of the PBS and the unfulfilled demand of the system. Demands in the city are estimated by analyzing the usage patterns of the Vélib' system in Paris. The considered instance for the case study is small and only considers 50 prospective candidate stations. The authors have been able to solve the provided instance with CPLEX and considered two case studies: one which assumes that the new PBS will be used heavily by the Athenians, and a second one, which assumes that the PBS is not that popular among the Athenian population.

Chen et al. [22] provide a mathematical non-linear programming model and solve the problem utilizing an improved immune algorithm. They define three different types of rental stations depending on their location (e.g., near a metro station, supermarkets). Their aim is that stations in the residential area have enough bikes available such that the morning peak can be managed and that stations near metro lines or important places have enough free parking slots available to manage incoming bikes during the morning peak. They provide a case study for a particular metro line of Nianjing city including 10 district stations and 31 residential stations.

Chen and Sun [23] aim at minimizing the total travel time of the users of the system which includes times for walking and cycling. They propose a MIP formulation which they solve by the LINGO solver. They consider multiple time periods and traffic zones corresponding to origins and destinations of the users of the system and prospective candidate sites for PBS stations. In the end the authors present a small numerical study with 20 candidate bike stations and ten traffic zones in detail. Despite the small instance size, the obtained model is huge and a long time (over a CPU month) was needed to solve it.

Frade et al. [52] present an exact approach based on MIP modeling. They aim at maximizing the total satisfiable demand of the system under budget constraints. Their model is rather simple and they consider building costs per planned slot. However, they evaluated their model only on a case study for the city center of Coimbra, Portugal, which is only a small instance with 29 potential stations. The authors evaluated four different scenarios with various input parameters and solved the proposed model using the XPRESS solver. As they are utilizing an exact approach, it is not going to scale well to much larger instances.

Hu et al. [73] present a small case study for establishing a PBS along a metro line. Different to most other work they aim at cost minimization, whereas other work aims at maximizing net revenue, fulfilled customer demand or minimizing unfulfilled demand. The MIP model is rather simple, and the only considered constraints in their MIP model are the minimum number of stations that have to be built. The authors show results for different scenarios based on a dataset of ten possible station candidates.

Ouyang et al. [105] model the problem using a graph theoretic pproach. Their aim is to ensure reachability and richness. By reachability they define a percentage threshold of the population that must be covered by the PBS, and by richness hey try to maximize the number of scenic spots covered by the PBS. The PBS is said to be reachable or a scenic spot is covered when the entry point or the scenic spot lies within a 200 meter range of a rental station. This is measured by finding the intersection between circles with a radius of 200 meters. The authors propose a station decision algorithm and test their algorithm in two simulations. In [106] Ouyang et al. refined the work and present the algorithmic components in more detail.

Straub et al. [146] present a semi-automated planning tool based on the optimization algorithm by Kloimüllner and Raidl [84] and give full details on the development procedure from the requirement analysis to the frontend of the semi-automated planning tool.

There exists also literature related to this topic. In particular, the problem belongs the class of *facility location problems* [129] and more generally to *hub location problems* [29, 49]. Moreover, a related and currently hot topic is the optimal placement of stations in car sharing systems [12]. Gavalas et al. [56], for instance, summarized diverse algorithmic approaches for the design and management of vehicle-sharing systems (e.g., car sharing and PBS).

We conclude that all previous works on computational optimization approaches for designing BSS only consider rather small scenarios. Most previous work accomplishes the optimization with compact mathematical models that are directly solved by a MIP solver. Such methods, however, are clearly unsuited for tackling large realistic scenarios of cities with up to 2000 cells or more. In the following, we therefore propose a novel multilevel refinement heuristic based on a hierarchical clustering of the demand data.

4.3 Problem Definition

We consider a geographical area which is partitioned into discrete cells. Let S denoted as the set of possible candidate cells where stations may be located, and let V be the set of cells containing some positive demand of prospective customers. Furthermore, let $m = |S|$ and $n = |V|$. In the simplest case $S = V$ holds, meaning that in each station with positive customer demand a station might be located, however, this does not necessarily need to be the case.

As it is meaningful to define cells with about 150×150 meters, the input size grows rapidly in larger cities. In particular, it is not meaningful and not practicable anymore

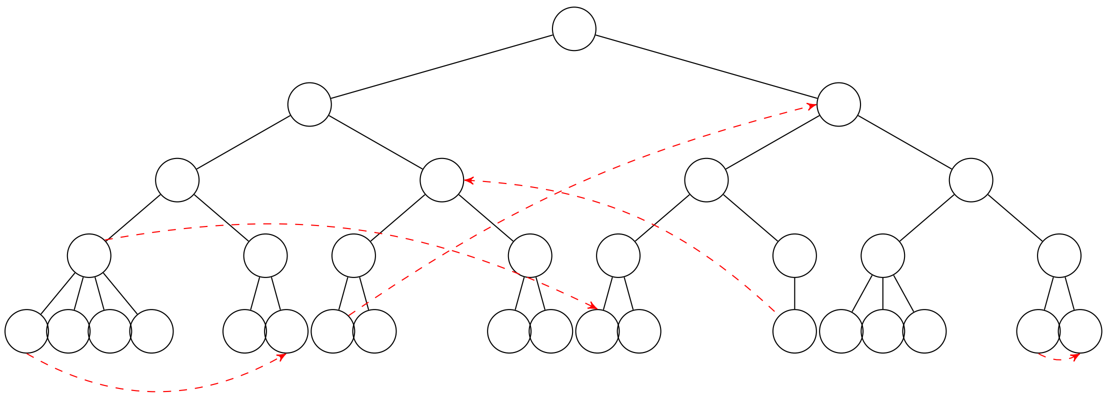


Figure 4.1: Example of a hierarchical clustering and the corresponding graph G^t

to consider the full origin-destination demand matrix. Instead we use a hierarchical abstraction as data structure to be able to have a chance for solving such instances and assume to this end that an hierarchical clustering of the cells is given as input.

This hierarchical clustering is given as a rooted tree where the leaves of the tree correspond to single cells and all other cells correspond to clusters of cells of the geographical area. Let h denote the height of the tree and let $C = C_0 \cup \ldots \cup C_h$ be the set of all nodes of the tree. By C_d we denote all nodes at height $d = 0, \ldots, h$ of the tree. Set C_0 only contains the root node representing the whole considered area and C_h is the set of original all cells of the geographical area, i.e., $C_h = V$. For convenience, we define $\text{super}(p) \in C$ to return the parent cluster of node $p \in C \setminus \{0\}$, and $\text{sub}(p) \subset C$ to return the children of cluster $p \in C \setminus V$.

To model varying demand throughout the day, we consider by $T = \{1, \ldots, \tau\}$ a set of time periods. We derive a weighted directed graph for each time period $t \in T : G^t = (C^t, A^t)$ of the full demand matrix by Algorithm 1. An example of the graph G^t on the hierarchically clustered input data is shown in Figure 4.1. The demand from node $v \in V$ to cluster $p \in C$ in time period $t \in T$ is denoted by $d^t_{v,p} > 0$. For convenience, we define $V(p) \subseteq V$ to be subset of all leaf nodes rooted under subtree $p \in C \setminus V$. Moreover, the set $C(p)$ contains all nodes which are part of the subtree rooted under p, also including $V(p)$ and p itself. The derivation of the arc set A^t of graph $G^t, \forall t \in T$ can be found in Algorithm 4.1 having parameters G^{in}, the full demand matrix input graph and θ, a parameter which defines the minimum demand for an arc in the newly constructed graph.

To keep this demand graph as sparse in order to be able to solve large instances we define a set of rules for the demands in the input graph. It is not allowed that there exists demand from a node v to one of its predecessors p, i.e., $d^t_{v,p} > 0 \mid v \in \text{sub}(p)$ is not possible. Self-loops, however, are a special case, and are explicitly allowed to model trips of customers within a cell or cluster. There must not exist arcs with negligible demand,

Algorithm 4.1: Derivation of A^t from original graph $G^{in} = (V^{in}, A^{in})$

```
Require: graph of full demand matrix $G^{in} = (V^{in}, A^{in}) \mid V^{in} = V^t$, threshold $\theta$
1: $A^t = A^{in}$
2: for all $l \in \{h-1, \dots, 0\}$ do
3:   for all $c \in C_l$ do
4:     for all $v \notin V(c)$ do
5:       $w^+ = 0, w^- = 0$
6:       for all $(v,p) \in A^t \mid p \in C(c) \cap C_{l+1}$ do
7:         if $d_{v,p} < \theta$ then
8:           $w^- = w^- + d^t_{v,p}$
9:           $A^t = A^t \setminus (v,p)$
10:        else
11:          $A^t = A^t \cup (v,p)$
12:        end if
13:      end for
14:      for all $(p,v) \in A \mid p \in C(c) \cap C_{l+1}$ do
15:        if $d_{p,v} < \theta$ then
16:          $w^+ = w^+ + d^t_{p,v}$
17:          $A^t = A^t \setminus (p,v)$
18:        else
19:          $A^t = A^t \cup (p,v)$
20:        end if
21:      end for
22:      if $w^- > 0$ then
23:        $A^t = A^t \cup (v,c)$ with $d^t_{v,c} = w^-$
24:      end if
25:      if $w^+ > 0$ then
26:        $A^t = A^t \cup (c,v)$ with $d^t_{c,v} = w^+$
27:      end if
28:    end for
29:  end for
30: end for
```

i.e., arcs with $d^t_{v,p} < \theta$ where θ is a predefined threshold, are not allowed. These “low” demands have to be subsumed to a bigger demand at a higher level of the clustering.

Proposition 4. *If there already exists an arc $(v,p) \in A^t$ with weight $d^t_{v,p} \geq \theta$, there cannot exist an arc $(v,q) \in A^t \mid p \in \mathrm{sub}(q) \vee p \in \mathrm{super}(q)$.*

This follows from Algorithm 4.1:

Proof. Given $(v,p) \in A^t$ with $d^t_{v,p} \geq \theta$ we distinguish the following two cases:

Case 1: Let $(v,q) \in A^t \mid p \in \mathrm{sub}(q)$. Since $p \in \mathrm{sub}(q)$ and $d^t_{v,p} \geq \theta$ no arc (v,q) is generated according to Line 7 of Algorithm 4.1.

Case 2: Let $(v,q) \in A^t \mid p \in \mathrm{super}(q)$. An arc $(v,p) \in A^t$ can only be generated if $d^t_{v,q} < \theta$ according to Line 7. In this case, however, arc $(v,q) \in A^t$ would be removed according to Line 9 of Algorithm 4.1. Moreover, if $d^t_{v,q} \geq \theta$, then arc (v,p) is not generated according to Line 7. Thus, only one of these arcs, either (v,p) or (v,q), can exist in the final arc set A^t.

□

An important condition that needs to be ensured is that incoming and outgoing demands must be consistent. Therefore, for any $p \in C \setminus V$ the following two conditions must hold:

$$\sum_{(v,q)\in A^t \mid v\in V(p), q\notin C(p)} d^t_{v,q} \geq \sum_{(q,v)\in A^t \mid q\in C(p), v\notin V(p)} d^t_{q,v} \tag{4.1}$$

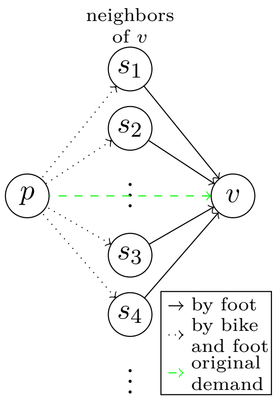
and

$$\sum_{(q,v)\in A^t \mid q\notin C(p), v\in V(p)} d^t_{q,v} \geq \sum_{(v,q)\in A^t \mid v\notin V(p), q\in C(p)} d^t_{v,q}. \tag{4.2}$$

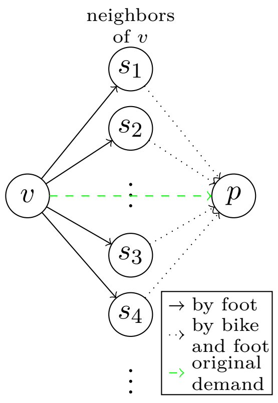
Inequality (4.1) ensures that the total demand originating at the leaves of the subtree rooted at p and leading to a destination outside of the tree is never less than the total incoming demand at all the cells outside the tree originating from some cluster inside the tree. Inequality (4.2) provides a symmetric condition for the total incoming demand at all the leaves of the tree. Furthermore, for the root node equations $p = 0$ the former inequalities and the latter inequalities must hold with equality.

An important fact for bike-sharing systems (PBS) is that demand of prospective customers may not only be fulfilled by its own cell but also by neighbor cells within a reasonable walking distance. The modeling of these neighbor cells is shown in Figure 4.2. Thus, we define for each leaf node $v \in V$ a set of station cells $S(v) \subseteq S$ which are in neighborhood of v and with which v's demand can at least be partly fulfilled. To model partly fulfilled demand, we introduce an *attractiveness value* $a_{v,s} \in (0,1]$, $\forall v \in V,\ s \in S(v)$. This value determines the percentage of demand of customer cell v which can be maximally fulfilled at a station at location s. Note, that if $v = s$, then $a_{v,v} = 1$ will hold.

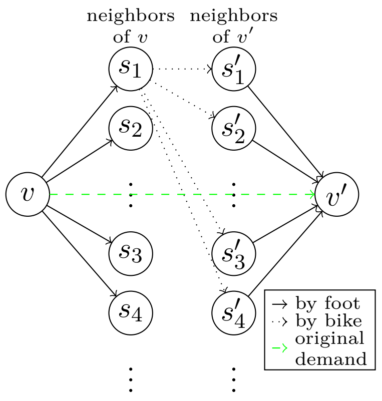
For each station cell $s \in S$, a set of possible station configurations $K_s = \{0,\dots,\gamma_s\}$ is specified. The special configuration 0 always corresponds to the case that no station is built or an existing station is removed. Each other configuration $i \in K_s$ has associated the following values: The number of parking slots is defined by $b^{\mathrm{ps}}_{s,i} \in \mathbb{N}$, the fixed costs, i.e., the costs for constructing the station including the purchase of a corresponding number of bikes are denoted by $b^{\mathrm{cfix}}_{s,i} \geq 0$, and the variable costs, i.e., total maintenance and operating costs over the whole planning horizon, including the maintenance of a respective number of bikes are denoted by $b^{\mathrm{cvar}}_{s,i} \geq 0$. In case the cell contains an already



(a) An arc from a cluster node to a leaf node



(b) An arc from a leaf node to a cluster node



(c) An arc from a leaf node to another leaf node

Figure 4.2: By modeling the neighbor stations for each customer cell, a separate network is created for every arc of graph G^t

existing station, $b^{\text{cfix}}_{s,i}$ corresponds to the costs for rebuilding or removing (if $i = 0$) the station.

Finally, we are given the following global parameters. Parameter $b^{\text{reb}} \geq 0$ represents the average costs for rebalancing a single bike per day over the whole planning horizon, whereas $B^{\text{tot}}_{\max}$ and $B^{\text{fix}}_{\max}$ represent the total maximum costs and the maximum fixed costs respectively. These maximum cost parameters are also denoted as *budget*. Basically, when considering two different kind of budgets, possible solution candidates may be excluded which could achieve better solution quality according to the objective function, but having two different budget types is a requirement from practice.

4.4 Hierarchical Clustering and Multilevel Refinement

In this work, we first concentrate on how to efficiently model the BSSPP such that we can also deal with very large instances with thousands of considered geographical cells for customers and potential station locations. To this end we propose to utilize a hierarchical clustering to express the estimated potential customer demand on it. We will then describe a *linear programming* (LP) based method to evaluate candidate solutions, and finally present a first novel *multilevel refinement heuristic* (MLR), based on mixed integer linear programming (MIP), to approach the optimization problem.

In Section 4.4.1 we who the solution representation while Section 4.4.2 shows how the objective is modeled. Sections 4.4.3 and 4.4.4 describe LP models for determining the actually fulfilled customer demands for a candidate solution and estimating the required rebalancing effort, respectively. The MLR is then described in Section 4.4.5. Computational results on randomly generated instances are shown in Section 4.4.6.

4.4.1 Solution Representation

A solution $x = \{x_s \in \mathbb{N} \mid s \in S\}$ assigns each station cell $s \in S$ an amount of parking slots to be built, possibly also 0 which would mean that no station is going to be built in cell s.

4.4.2 Objective

The goal is to maximize the expected total number of journeys in the system, i.e., the total demand that actually can be fulfilled at each day over all time periods, considering the available budgets $B^{\text{tot}}_{\max}$ and $B^{\text{fix}}_{\max}$.

Let $D(x,t)$ be the total demand fulfilled by solution x in time period $t \in T$, and let $Q_x(s)$ be the required rebalancing effort arising at each station $s \in S \mid x_s \neq 0$ in terms of the number of bikes to be moved to some other station. The calculation of these values will be considered separately in Sections 4.4.3 and 4.4.4. The BSSPP can then be stated as the following MIP.

$$\max \sum_{t \in T} D(x,t) \tag{4.3}$$

$$\sum_{s \in S} (b^{\text{fix}} \cdot x_s + b^{\text{var}} \cdot x_s + b^{\text{reb}} \cdot Q_x(s)) \leq B^{\text{tot}}_{\max} \tag{4.4}$$

$$\sum_{s \in S} b^{\text{fix}} \cdot x_s \leq B^{\text{fix}}_{\max} \tag{4.5}$$

$$x_s \in \{0, \dots, z_s\} \qquad s \in S \tag{4.6}$$

Inequality (4.4) calculates the total costs over all stations and ensures that the total budget is not exceeded, while inequality (4.5) restricts only the fixed costs over all stations by the respective budget.

4.4.3 Calculation of Fulfilled Customer Demand

To determine the overall fulfilled demand for a specific, given solution x and a certain time slot $t \in T$, we first make the following local definitions. Let $S' = \{s \in S \mid x_s \neq 0\}$ correspond to the set of cells where a station actually is located, $V' = \{v \in V \mid S(v) \cap S' \neq \emptyset\}$ be the set of customer cells whose demand can possibly (partly) be fulfilled as at least one station exists in the neighborhood. Moreover, let $C' = \{p \in C \mid V(p) \cap V' \neq \emptyset\}$ be the set of all nodes in the hierarchical clustering representing relevant customer cells, i.e., cells whose demand can possibly be fulfilled. The set $S'(v) = S(v) \cap V'$, $\forall v \in V'$ refers to the existing stations that might fulfill part of v's demand, and $V'(p) = V(p) \cap V'$, $\forall p \in C'$ denotes the existing customer cells contained in cluster p. $C'(p)$ refers to the subset of all the nodes $q \in C'$ that are part of the subtree rooted at p, including p and $V'(p)$, and $G' = (C', A')$ with $A' = \{(p, q) \in A^t \mid p, q \in C'\}$ is then the correspondingly reduced demand graph.

In the following we use variables u, v, w for referencing customer cells in V', variables p, q for referencing cluster nodes in C' (which might possibly also be customer cells), variable s for station cells in S', and α, β for arbitrary nodes in $C' \cup S$.

We further define for each arc in A' corresponding to a specific demand an individual flow network depending on the kind of the arc:

- Arcs $(u, v) \in A'$ with $u, v \in V'$, including the case $u = v$:
  $G_{\mathrm{f}}^{u,v} = (V_{\mathrm{f}}^{u,v}, A_{\mathrm{f}}^{u,v})$ with node set $V_{\mathrm{f}}^{u,v} = \{u\} \cup S'(u) \cup S'(v) \cup \{v\}$ and arc set $A_{\mathrm{f}}^{u,v} = (\{u\} \times S'(u)) \cup (S'(u) \times S'(v)) \cup (S'(v) \times \{v\})$.
- Arcs $(v, p) \in A'$ with $v \in V', p \in C' \setminus V'$:
  $G_{\mathrm{f}}^{v,p} = (V_{\mathrm{f}}^{v,p}, A_{\mathrm{f}}^{v,p})$ with node set $V_{\mathrm{f}}^{v,p} = \{v\} \cup S'(v) \cup \{p\}$ and arc set $A_{\mathrm{f}}^{v,p} = (\{v\} \times S'(v)) \cup (S'(v) \times \{p\})$.
- Arcs $(p, v) \in A'$ with $p \in C' \setminus V', v \in V'$:
  $G_{\mathrm{f}}^{p,v} = (V_{\mathrm{f}}^{p,v}, A_{\mathrm{f}}^{p,v})$ with node set $V_{\mathrm{f}}^{p,v} = \{p\} \cup S'(v) \cup \{v\}$ and arc set $A_{\mathrm{f}}^{p,v} = (\{p\} \times S'(v)) \cup (S'(v) \times \{v\})$.

All arcs $(\alpha, \beta) \in A_{\mathrm{f}}^{p,q}$ of all flow networks have associated corresponding flow variables $0 \leq f_{\alpha,\beta}^{p,q} \leq d_{p,q}^{t}$. The fulfilled demands can be modeled within these networks as maximum flows. Furthermore, we utilize variables H_p^{in}, H_p^{out} $\forall p \in C' \setminus V'$, for the total inflow/outflow at all customer cells $V'(p)$ originating at/targeted to cluster nodes from outside cluster p, i.e., $C' \setminus C'(p) \setminus V'$. Variables $F_p^{\mathrm{in}}, F_p^{\mathrm{out}}$, $\forall p \in C' \setminus V'$, represent the total ingoing/outgoing flows at all cluster nodes q within cluster p originating at/targeted to customer cells outside cluster p, i.e., $V' \setminus V'(p)$, respectively. The flow variables, however, depend on each other and the stations' capacities. A weighting factor ω is used to adjust the number of trips which can be performed in time period t by using only a single bike. The following LP is used to compute the total satisfied demand $D(x,t) =$

$$\begin{aligned}
\max \quad & \sum_{(v,p)\in A' | v \in V'} \sum_{(v,s)\in A_\mathrm{f}^{v,p}} f_{v,s}^{v,p} && && (4.7)\\
\text{s.t.} \quad & \sum_{(v,s)\in A_\mathrm{f}^{v,p}} f_{v,s}^{v,p} \leq d_{v,p}^t && (v,p) \in A' \mid v \in V' && (4.8)\\
& \sum_{(s,v)\in A_\mathrm{f}^{p,v}} f_{s,v}^{p,v} \leq d_{p,v}^t && (p,v) \in A' \mid v \in V' && (4.9)\\
& f_{u,s}^{u,v} = \sum_{s'\in S'(v)} f_{s,s'}^{u,v} && (u,v) \in A' \mid u,v \in V', && (4.10)\\
& && s \in S'(u) &&\\
& \sum_{s'\in S'(u)} f_{s',s}^{u,v} = f_{s,v}^{u,v} && (u,v) \in A' \mid u,v \in V', && (4.11)\\
& && s \in S'(v) &&\\
& f_{v,s}^{v,p} = f_{s,p}^{v,p} && (v,p) \in A' \mid v \in V', && (4.12)\\
& && p \in C' \setminus V', s \in S'(v) &&\\
& f_{p,s}^{p,v} = f_{s,v}^{p,v} && (p,v) \in A' \mid v \in V', && (4.13)\\
& && p \in C' \setminus V', s \in S'(v) &&\\
& -x_s \leq \sum_{(p,q)\in A'} \sum_{(\alpha,s)\in A_\mathrm{f}^{p,q}} f_{\alpha,s}^{p,q} && s \in S' && (4.14)\\
& \qquad - \sum_{(p,q)\in A'} \sum_{(s,\alpha)\in A_\mathrm{f}^{p,q}} f_{s,\alpha}^{p,q} && &&\\
& \qquad - \omega \cdot \frac{\delta^\mathrm{rent} \cdot \sum_{(p,q)\in A'} \sum_{(\alpha,s)\in A_\mathrm{f}^{p,q}} f_{\alpha,s}^{p,q}}{\delta_t^\mathrm{period}} && &&\\
& x_s \geq \sum_{(p,q)\in A'} \sum_{(\alpha,s)\in A_\mathrm{f}^{p,q}} f_{\alpha,s}^{p,q} && s \in S' && (4.15)\\
& \qquad - \sum_{(p,q)\in A'} \sum_{(s,\alpha)\in A_\mathrm{f}^{p,q}} f_{s,\alpha}^{p,q} && &&\\
& \qquad + \omega \cdot \frac{\delta^\mathrm{rent} \cdot \sum_{(p,q)\in A'} \sum_{(s,\alpha)\in A_\mathrm{f}^{p,q}} f_{s,\alpha}^{p,q}}{\delta_t^\mathrm{period}} && &&\\
& H_p^\mathrm{in} = \sum_{(q,v)\in A' | q \notin C'(p) \cup V', v \in V'(p)} && p \in C' \setminus V' && (4.16)\\
& \qquad \sum_{(s,q)\in A_\mathrm{f}^{q,v}} f_{s,q}^{q,v} && &&
\end{aligned}$$

$$F_p^{\text{in}} = \sum_{(v,q)\in A' \mid v\notin V'(p), q\in C'(p)\setminus V'} \sum_{(s,q)\in A_{\text{f}}^{v,q}} f_{s,q}^{v,p} \qquad p \in C' \setminus V' \tag{4.17}$$

$$H_p^{\text{in}} \geq F_p^{\text{in}} \qquad p \in C' \setminus V' \setminus \{0\} \tag{4.18}$$

$$H_0^{\text{in}} = F_0^{\text{in}} \tag{4.19}$$

$$H_p^{\text{out}} = \sum_{(v,q)\in A' \mid v\in V'(p), q\notin C'(p)\cup V'} \sum_{(q,s)\in A_{\text{f}}^{q,v}} f_{q,s}^{q,v} \qquad p \in C' \setminus V' \tag{4.20}$$

$$F_p^{\text{out}} = \sum_{(q,v)\in A' \mid q\in C'(p)\setminus V', v\notin V'(p)} \sum_{(p,s)\in A_{\text{f}}^{q,v}} f_{q,s}^{q,v} \qquad p \in C' \setminus V' \tag{4.21}$$

$$H_p^{\text{out}} \geq F_p^{\text{out}} \qquad p \in C' \setminus V' \setminus \{0\} \tag{4.22}$$

$$H_0^{\text{out}} = F_0^{\text{out}} \tag{4.23}$$

$$0 \leq f_{v,s}^{v,p} \leq a_{v,s} \cdot d_{v,p}^t \qquad (v,p) \in A' \mid v \in V',\ (v,s) \in A_{\text{f}}^{v,p} \tag{4.24}$$

$$0 \leq f_{s,v}^{p,v} \leq a_{s,v} \cdot d_{p,v}^t \qquad (p,v) \in A' \mid v \in V',\ (s,v) \in A_{\text{f}}^{p,v} \tag{4.25}$$

$$0 \leq f_{\alpha,\beta}^{p,q} \leq d_{p,q}^t \qquad (p,q) \in A',\ (\alpha,\beta) \in A_{\text{f}}^{p,q} \mid \alpha,\beta \notin V' \tag{4.26}$$

$$F_p^{\text{in}}, F_p^{\text{out}} \geq 0 \qquad p \in C' \setminus V' \tag{4.27}$$

$$H_p^{\text{in}}, H_p^{\text{out}} \geq 0 \qquad p \in C' \setminus V' \tag{4.28}$$

Objective function (4.7) maximizes the total outgoing flow over all $v \in V'$, i.e., the fulfilled demand. Note that this also corresponds to the total ingoing flow over all v. Inequalities (4.8) limit the total flow leaving $v \in V'$, for each demand $(v,p) \in A' \mid v \in V'$ to $d_{v,p}^t$. Inequalities (4.9) do the same w.r.t. ingoing demands. Equalities (4.10) and (4.11) provide the flow conservation at source and destination stations s for $(u,v) \in A'$ with $u, v \in V'$. Equalities (4.12) provide the flow conservation at the source station in case of an arc $(v,p) \in A'$ towards a cluster node p, while (4.13) provide the flow conservation at the destination station in case of an arc $(p,v) \in A'$ originating at a cluster node p. Inequalities (4.14) and (4.15) provide the capacity limitations at each station $v \in V'$. It is the accumulated demand occurring at the particular station including a "compensation term" for large values of ingoing as well as outgoing demand. The fraction $\delta_t^{\text{period}}/\delta^{\text{rent}}$ represents the number of trips which can ideally be performed in period t using a single bike. The weighting factor ω is used to adjust this value such that it better reflects reality as the bike trips are not likely to be performed "optimally" with respect to the distribution over the whole time period in real world. Equalities (4.16) compute the total outgoing flow for the leaves of the subtree rooted at p to any cluster which is not part of the subtree rooted at p. Equalities (4.17) compute the total ingoing flow for each cluster node p by considering the ingoing flow from any $v \in V$ for which p is not a predecessor to every cluster of the subtree rooted at p. Inequalities (4.18) ensure that there must not be more ingoing flow to clusters of the subtree rooted at p as there is outgoing flow from the leaves contained in the subtree rooted at p. Equality (4.19) ensures that at the top level, i.e., at the root node 0, the outgoing flow from leaf nodes to cluster nodes and the ingoing flow from cluster nodes to leaf nodes is balanced, i.e, the same amount. Inequalities (4.21)–(4.23) state the corresponding constraints for the outgoing flow instead of the ingoing flow. Equations (4.24) and (4.25) provide the domain definitions for the flow variables from/to a cell v to/from a neighboring station s by considering the demand weighted by factor $a_{v,s}$. For all remaining flow variables, (4.26) provide the domain definitions based on the demands. The remaining variables are just restricted to be non-negative in (4.27) and (4.28).

4.4.4 Calculation of Rebalancing Costs

We state an LP for minimizing the total rebalancing effort over all time periods T at each station $s \in S'$ by choosing an appropriate initial fill level for each period, ensuring that the whole prospective customer demand is fulfilled. We estimate the rebalancing effort by considering the necessary changes in the fill levels inbetween the time periods. The LP uses the following decision variables. By $y_{t,s}$ we refer to the initial fill level of station $s \in S'$ at the beginning of time period $t \in T$, and by $r^+_{t,s}$ and $r^-_{t,s}$ we denote the number of bikes which need to be delivered to, respectively picked up from, station $s \in S'$ at the end of period $t \in T$ to achieve the fill levels $y_{t+1,s}$ (or $y_{1,s}$ in case of $t = \tau$).

The accumulated demand $D^{\mathrm{acc}}_{t,v}$ can be calculated by utilizing the solution of the previous model from Section 4.4.3, c.f. inequalities (4.14) and (4.15). The following LP is solved for each station $s \in S' \mid x_s \neq 0$ independently. For station cells $s \in S \setminus S'$, i.e., where no station is actually built in solution x, $Q_x(s) = 0$.

$$Q_x(s) = \min \quad \sum_{t \in T} r^+_{t,s} + r^-_{t,s} \tag{4.29}$$

$$\text{s.t.} \quad y_{t,s} + r^+_{t,s} \geq D^{\mathrm{acc}}_{t,s} \qquad t \in T \tag{4.30}$$

$$x_s - y_{t,s} + r^-_{t,s} \geq -D^{\mathrm{acc}}_{t,s} \qquad t \in T \tag{4.31}$$

$$y_{t+1,s} = y_{t,s} - D^{\mathrm{acc}}_{t,s} + r^+_{t,s} - r^-_{t,s} \qquad t \in T \setminus \{\tau\} \tag{4.32}$$

$$y_{1,s} = y_{\tau,s} - D^{\mathrm{acc}}_{\tau,s} + r^+_{\tau,s} - r^-_{\tau,s} \tag{4.33}$$

$$0 \leq y_{t,s} \leq x_s \qquad t \in T \tag{4.34}$$

$$0 \leq r^+_{t,s} \leq D^{\mathrm{acc}}_{t,s} \qquad t \in T \tag{4.35}$$

$$0 \leq r^-_{t,s} \leq -D^{\mathrm{acc}}_{t,s} \qquad t \in T \tag{4.36}$$

Objective function (4.29) minimizes the number of rebalanced bikes, i.e., number of bikes that have to be delivered $r^+_{t,s}$ and number of bikes that have to be picked up

$r^{-}_{t,s}$. Inequalities (4.30) compute the number of bikes that have to be delivered to the corresponding station in order to meet the given demand. Inequalities (4.31) compute the number of bikes that have to be picked up from the corresponding station in order to meet the given demand. Inequalities (4.32) state a recursion in order to compute the fill level for the next time period. Inequalities (4.33) state that for each station the fill level for the next day has to be again the initial fill level of the first period. Inequalities (4.34)–(4.36) are the domain definitions for the number of bikes to be moved and the fill level for each time period.

4.4.5 Multilevel Refinement Approach

Clearly, practical instances of the problem are far too large to be approached by a direct exact MIP approach. However, also basic constructive techniques or metaheuristics with simple, classical neighborhoods are unlikely to yield reasonable results when making decisions on a low level without considering crucial relationships on higher abstraction levels, i.e., a more global view. Classical local search techniques on the natural variable domains concerning decisions for individual stations may only fine-tune a solution but are hardly able to overcome bad solutions in which larger regions need to be either supplied with new stations or where many stations need to be removed. We therefore have the strong need of some technique that exploits also a higher-level view, deciding for larger areas about the supply of stations in principle. Multilevel refinement strategies can provide this point-of-view.

In multilevel refinement strategies [159] the whole problem is iteratively coarsened (aggregated) until a certain problem size is reached that can be reasonably handled by some exact or heuristic optimization technique. After obtaining a solution at this highest abstraction level, the solution is iteratively extended to the previous lower level problem instance and possibly refined by some local search, until a solution to the original problem at the lowest level, i.e., the original problem instance, is obtained. For a general discussion and the generic framework we refer to the work of Walshaw [158].

To apply multilevel refinement to BSSPP we essentially have to decide how to realize the procedures for coarsening an instance for the next higher level, solving a reasonably small instance, and extending a solution to a solution at the next lower level. In the following, we denote all problem instance data at level l by an additional superscript l. By P_l we generally refer to the problem at level l of the MLR algorithm described here.

Coarsening

We have to derive the more abstract problem instance P_{l+1} from a given instance P_l. Naturally, we can exploit the already existing customer cell cluster hierarchy for the coarsening. Remember that all customer cells appear in the cluster hierarchy always at the same level. We coarsen the problem by considering the customer cells and the station cells separately.

Coarsening of customer cells: The main strategy for coarsening the customer cells is to merge cells having the same parent cluster together with their parent. This means $V^{l+1} = C^l_{h^l-1}$ or simply $V^{l+1} = C_{h-l-1}$, i.e., each cluster node at depth $h - l - 1$ corresponds to a customer cell at level $l + 1$ representing the merged set of customer nodes contained in C_{h-l-1}. The hierarchical clustering of P_l becomes $C^{l+1} = C_0 \cup \ldots \cup C_{h-l}$. Remember that we already defined the function super(p) to return the parent cluster of some node p, and therefore super$(p^l) : C^l \to C^{l+1}$ also returns the cluster from C^{l+1} in which cluster $p^l \in C^l$ is merged into. The new demand graph $G^{t,l+1} = (C^{t,l+1}, A^{t,l+1})$ consists of the arc set $A^{t,l+1} = \bigcup_{(p^l,q^l)\in A^{t,l}}(\text{super}(p^l), \text{super}(q^l))$. This demand graph may again contain self-loops, but it is still simple, i.e., multiple arcs from $A^{t,l}$ may map to the same single arc in $A^{t,l+1}$ and the respective demand values are merged. Considering an arc $(p^{l+1}, q^{l+1}) \in A^{t,l+1}$, its associated demand is thus

$$d^{t,l+1}_{p^{l+1},q^{l+1}} = \sum_{(p^l,q^l)\in A^{t,l} | p^{l+1}=\text{super}(p^l), q^{l+1}=\text{super}(q^l)} d^{t,l}_{p^l,q^l}. \tag{4.37}$$

Note that the conditions for a valid demand graph and valid demand values stated in inequalities (4.1) and (4.2) will still hold when aggregating in this way, since the total ingoing and outgoing demand at each cluster $p \in C^{l+1}$ (including the demands from and to all existing subnodes) stays the same.

Coarsening of station cells: To coarsen the station cells we need to define a hierarchical clustering for them as well. For simplicity we assume from now on that $S = V$ holds, i.e., there is a one-to-one correspondence of considered station cells and customer cells. This also appears reasonable in a practical setting. We can then apply the hierarchical clustering defined for the customer cells also to the station cells. Maximum station capacities for aggregated stations $s^{l+1} \in S^{l+1}$ are naturally calculated by the sum of the respective maximum capacities of the underlying station cells, i.e., $z^{l+1}_{s^{l+1}} = \sum_{s^l\in\text{sub}(s^{l+1})} z^l_{s^l}$.

Coarsening of neighborhoods: A coarsened neighborhood mapping $S^{l+1}(v^{l+1})$ for each customer cell $v^{l+1} \in V^{l+1}$ and respective attractiveness values $a_{v^{l+1},s^{l+1}}$ for station cells $s^{l+1} \in S^{l+1}(v^{l+1})$ are determined as follows. The neighborhood mapping is retained as long as the attractiveness value in the coarsened problem instance does not fall below a certain threshold $\lambda \in (0, 1)$:

$$S^{l+1}(v^{l+1}) = \left\{ s^{l+1} \in \bigcup_{v^l\in\text{sub}(v^{l+1})} \text{super}(S^l(v^l)) \mid a_{v^{l+1},s^{l+1}} \geq \lambda \right\} \tag{4.38}$$

with the aggregated attractiveness values being

$$a_{v^{l+1},s^{l+1}} = \begin{cases} 1 & \text{if } v^{l+1} = s^{l+1} \\ \frac{\sum_{v^l\in\text{sub}(v^{l+1})} \sum_{s^l\in\text{sub}(s^{l+1})\cap S^l(v^l)} \left(a_{v^l,s^l}\right)}{|\text{sub}(v^{l+1})|\cdot|\text{sub}(s^{l+1})|} & \text{if } v^{l+1} \neq s^{l+1}. \end{cases} \tag{4.39}$$

Initialization

The initial problem becomes coarsened until we reach some level l where it can be reasonably solved as it is then small enough. In our experiments with binary clustering trees here we are stopping the coarsening when the clustering tree has no more than $2^5 = 32$ leaf nodes, or in other words, at a height of five. For initializing the solution at the coarsest level we utilize a MIP model. In this model, the objective stated in Section 4.4.2, the demand calculation for every time period stated in Section 4.4.3, and the rebalancing LP model stated in Section 4.4.4 are put together. By solving this model we obtain an optimal solution for the coarsest level, which forms the basis for proceeding with the next step of the algorithm, the *extension* to derive step-by-step a more detailed solutions.

Extension

In the extension step we derive from a solution x^{l+1} at level $l+1$ a solution x^l at level l, i.e., we have to decide for each aggregated station $s^{l+1} \in S^{l+1}$ with $x^{l+1}_{s^{l+1}} > 0$ slots how they should be realized by the respective underlying station cells $\text{sub}(s^{l+1})$ at level l. We do this in a way so that the globally fulfilled demand is again maximized by solving the following MIP.

$$\max \quad \sum_{t \in T} D(x^l, t) \tag{4.40}$$

$$\text{s.t.} \quad \sum_{s^l \in S^l} \left(b^{\text{fix}} \cdot x^l_{s^l} + b^{\text{var}} \cdot x^l_{s^l} + b^{\text{reb}} \cdot Q_{x^l}(s^l) \right) \leq B^{\text{tot}}_{\max} \tag{4.41}$$

$$\sum_{s^l \in S^l} b^{\text{fix}} \cdot x_{s^l} \leq B^{\text{fix}}_{\max} \tag{4.42}$$

$$\sum_{s^l \in \text{sub}(s^{l+1})} x^l_{s^l} \leq x^{l+1}_{s^{l+1}} \qquad s^{l+1} \in S^{l+1} \tag{4.43}$$

$$x^l_{s^l} \in \{0, \ldots, z^l_s\} \qquad s^l \in S^l \tag{4.44}$$

The objective (4.40) maximizes the total satisfiable demand. Inequalities (4.41) restrict the maximum total budget whereas inequalities (4.42) restrict the maximum fixed budget. Inequalities (4.43) are the bounds on the total number of slots for the station nodes $s^l \in \text{sub}(s^{l+1})$. The number of parking slots in each cell $x^l_{s^l}$ is restricted by the maximum number of parking slots allowed in this cell (4.44).

4.4.6 Computational Results

For our experiments we created seven different benchmark sets[1], each one containing 20 different, random instances. We consider instances with 200, 300, 500, 800, 1000, 1500, and 2000 customer cells, where each customer cell is also a possible location for a station to be built. Customer cells are aligned on a grid in the plane and euclidean distances

[1] `https://www.ac.tuwien.ac.at/files/resources/instances/bsspp/lion17.bz2`

Instance				**MLR**				
name	#runs	$B^{\text{tot}}_{\max}$ [€]	$B^{\text{fix}}_{\max}$ [€]	$\overline{\text{obj}}$	$\overline{\text{\#coarsen}}$	$\widetilde{\text{time}}$ [s]	$\overline{\text{totcost}}$ [€]	$\overline{\text{fixcost}}$ [€]
BSSPP_200	20	200,000.00	130,000.00	9,651.98	3	46.2	198,000.00	126,000.00
BSSPP_300	20	350,000.00	250,000.00	10,951.79	5	60.8	349,250.00	222,250.00
BSSPP_500	20	500,000.00	350,000.00	16,057.78	6	121.6	497,750.00	316,750.00
BSSPP_800	20	850,000.00	550,000.00	28,862.21	6	263.9	849,750.00	540,750.00
BSSPP_1000	20	1,000,000.00	700,000.00	28,967.58	8	346.7	998,250.00	635,250.00
BSSPP_1500	20	1,500,000.00	1,000,000.00	41,208.19	8	574.5	1,498,475.00	953,575.00
BSSPP_2000	20	2,000,000.00	1,300,000.00	55,892.06	8	803.4	1,999,250.00	1,272,250.00
			Average	27,370.22	6.3		912,960.71	580,975.00

Table 4.1: Results for the multilevel refinement heuristic (MLR).

have been calculated based on which a hierarchical clustering with the complete-linkage method was computed. Demands among the leaf nodes were chosen randomly, considering the pairwise distance between customer cells, and demands below a certain threshold have been aggregated upwards in the clustering tree such that the demand graphs get sparser. Only cells within 200 meters walking distance are considered to be in the vicinity of a customer cell and respective attractiveness values are chosen randomly but in correlation with the distances. We set the maximum station size to $z_s = 40$ for all cells in all test cases. For slot costs we set $b^{\text{fix}} = 1750$ €, and $b^{\text{var}} = 1000$ €, which are reasonable estimates in the Vienna area gathered from real BSSs. The costs for rebalancing a single bike for one day have been estimated with 3 € per bike and per day. When projecting this cost to the optimization horizon, e.g., 1 year, we get $b^{\text{reb}} = 365 \cdot 3 = 1095$ €. For coarsening of attractiveness values, we set the corresponding parameter $\lambda = 0$ and for adjusting the number of trips which can be performed in a particular time period $t \in T$ by using only a single bike we set $\omega = 1.2$. Each instance contains four time periods which we selected as follows: 4:30am to 8:00am, 8:00am to 12:00 Noon, 12:00 Noon to 6:15pm, and 6:15pm to 4:30am. The duration for each time period $t \in T$ has been set accordingly and the average trip duration has been set to $t^{\text{rent}} = 10$ minutes.

All algorithms are implemented in C++ and have been compiled with gcc 4.8. For solving the LPs and MIPs we used Gurobi 7.0. All experiments were executed as single threads on an Intel Xeon E5540 2.53GHz Quad Core processor.

Table 4.1 summarizes obtained results. For every instance set we state the name containing the number of nodes, the number of different instances we have tested on (#runs), the maximum total budget ($B^{\text{tot}}_{\max}$), and the maximum fixed budget ($B^{\text{fix}}_{\max}$). For the proposed MLR, we list the average objective value ($\overline{\text{obj}}$), i.e., the expected fulfilled demand in terms of the number of journeys, the average number of coarsening levels ($\overline{\text{\#coarsen}}$), the median time ($\widetilde{\text{time}}$), and the average total costs ($\overline{\text{totcost}}$) as well as the average fixed costs ($\overline{\text{fixcost}}$) for building the number of slots in the solution. Most importantly, it can be seen that the proposed MLR scales very well to large instances up to 2000 customer cells.

4.5 Solving Large-Scale Instances Based on Real-World Data

In this section we consider real-world instances of the Bike-Sharing Station Planning Problem (BSSPP) from the city of Vienna. It is an integrated approach where we do not only compute optimal station locations but also considering the rebalancing effort and compute suggestions for initial inventory at the bike-sharing stations. We introduce a refinement step based on local-search techniques and are given possible station configuration and the prices therefore. Results are shown for up to 4000 prospective station candidates and customer cells.

The remainder of this section is structured as follows. In Section 4.5.1, the solution representation of the problem is given. Then, in Section 4.5.2, the optimization goal is formalized. The multilevel refinement approach is explained in Section 4.5.3 and computational results are presented in Section 4.5.4.

4.5.1 Solution Representation

A solution $x = (x_s)_{s \in S}$ with $x_s \in K_s$ assigns each station cell $s \in S$ a valid configuration $x_s \in K_s$ (which might also be the "no-station" configuration 0).

4.5.2 Optimization Goal

The goal is to maximize the total number of trips in the system, i.e., the total demand that can be fulfilled at each day over all time periods, considering a maximum total budget $B^{\mathrm{tot}}_{\max}$ as well as a maximum budget for the overall fixed costs $B^{\mathrm{fix}}_{\max}$ alone.

Let $D(x, t)$ be the total demand fulfilled by solution x in time period $t \in T$, and let $Q_x(s)$ be the required rebalancing effort arising at each station $s \in S \mid x_s \neq 0$ in terms of the number of bikes to be moved to some other station. The calculation of these terms via MIP models was already presented in detail in [84].

The corresponding optimization problem can then be stated as follows.

$$\max \sum_{t \in T} D(x, t) \tag{4.45}$$

$$\sum_{s \in S} \left(b^{\mathrm{cfix}}_{s,x_s} + b^{\mathrm{cvar}}_{s,x_s} + b^{\mathrm{reb}} \cdot Q_x(s) \right) \leq B^{\mathrm{tot}}_{\max} \tag{4.46}$$

$$\sum_{s \in S} b^{\mathrm{cfix}}_{s,x_s} \leq B^{\mathrm{fix}}_{\max} \tag{4.47}$$

$$x_s \in K_s \qquad s \in S \tag{4.48}$$

The objective function (4.45) maximizes the total satisfiable demand for each time period $t \in T$. The left side of inequality (4.46) calculates the total costs by summing up the fixed and variable costs resulting from a change in station configurations and the rebalancing costs. Inequality (4.47) restricts the maximum fixed costs of the new system and in (4.48) domain definitions for the decision variables are given.

4.5.3 Multilevel Refinement Approach

In our opinion, for this type of problem, single construction heuristics based on greedy principles are not the ultimate choice as solution technique and neither is a direct application of classical local-search based metaheuristics like variable neighborhood search and iterated local search. We think that these strategies are not able to grasp the connections and interactions between all the clusters and leaf nodes in large instances. In this context many local decisions are not the way to go. One has to find a solution method which is able to overlook the complete and complex problem more as a whole. An intuitive way of achieving this, is to apply the so called *multilevel refinement approach* which is stated in pseudo-code in Algorithm 2.8.

Initially, it was thought as an additional ingredient to any metaheuristic to improve its solutions. However, we are going to use the approach as the main solution technique. This technique fits to the already given hierarchical clustering of the geographic cells, as we can do the coarsening accordingly, level by level, until we reach a problem size that can be reasonably well solved. We, then compute an initial solution by the *initialize* procedure and finally, *extend* and *refine* the solution until we obtain a solution to the original input instance.

In the following the *coarsen*, *initialize*, *extend*, and *refine* functions are explained in detail.

Coarsening

In the coarsening, we iteratively merge neighboring clusters into larger clusters according to the already given hierarchical clustering.

Coarsening of customer cells: As illustrated in Figure 4.3, in the coarsening step, customer cells are merged together into their parent cluster, so that the problem becomes smaller and gets easier solvable.

The outgoing demand of a node p that corresponds to the merging of nodes $V(p)$ is the demand $\sum_{(v,q)\in A^t|v\in V(p),q\notin C(p)} d^t_{v,q}$ and the ingoing demand of p is the total demand of nodes $V(p)$, i.e., $\sum_{(q,v)\in A^t|v\in V(p),q\notin C(p)} d^t_{q,v}$.

Coarsening of station cells: Merging a set of station cells, each representing possible station configurations with different associated costs, is, however, not as straight-forward. Considering all possible combinations of station configurations appears not meaningful, since the resulting number of these combinations would grow exponentially with the number of merged original station configurations. Furthermore, in particular on higher abstraction levels, individual station configurations do not play a practically significant

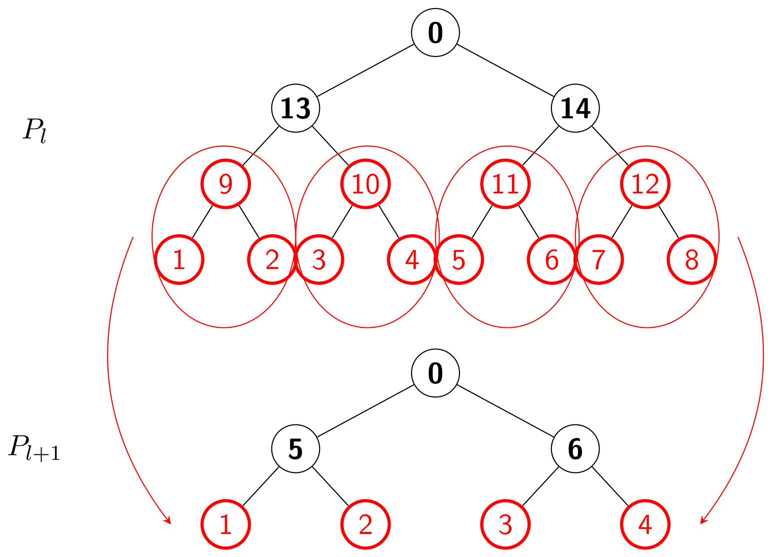


Figure 4.3: Example of a coarsening step in the hierarchical clustering of customer cells. The leaf nodes are merged with their parent nodes to form new leaf nodes in the coarsened tree.

role anymore. A simpler approximate model for the number of possible parking slots and corresponding costs appears thus reasonable for practice.

We apply the following continuous linear model as approximation for the possibilities at each station cell $s \in S^l$. Let b_s^{ps} be the maximum number of bike parking slots at prospective station s. The possible number of bike parking slots at s can be chosen freely from the (continuous) interval $[0, b_s^{\text{ps}}]$ where the upper bound b_s^{ps} will be chosen as explained below. Consequently, a solution to P^l is a vector containing for each station cell the selected number of bike parking slots, i.e., $x = (x_s)_{s \in S^l}$ with $x_s \in [0, b_s^{\text{ps}}]$. Costs are calculated in dependence of x_s as follows:

$$b_s^{\text{cfix}}(x_s) = b_s^{\text{cfix,a}} \cdot x_s + b_s^{\text{cfix,b}} \tag{4.49}$$

$$b_s^{\text{cvar}}(x_s) = b_s^{\text{cvar,a}} \cdot x_s + b_s^{\text{cvar,b}} \tag{4.50}$$

For original station cells $s \in S^0$ we assume $b_s^{\text{ps}} = \max_{i \in K_s} b_{s,i}^{\text{ps}}$ and the model parameters $b_s^{\text{cfix,a}}$, $b_s^{\text{cfix,b}}$, $b_s^{\text{cvar,a}}$, and $b_s^{\text{cfix,b}}$ are determined as follows:

$$b_{s^{l+1}}^{\text{cfix,a}} = \sum_{s^l \in \text{sub}(s^{l+1})} b_{s^l}^{\text{cfix,a}} \quad \text{and} \quad b_{s^{l+1}}^{\text{cfix,b}} = \frac{1}{|\text{sub}(s^{l+1})|} \sum_{s^l \in \text{sub}(s^{l+1})} b_{s^l}^{\text{cfix,b}} \tag{4.51}$$

$$b_{s^{l+1}}^{\text{cvar,a}} = \sum_{s^l \in \text{sub}(s^{l+1})} b_{s^l}^{\text{cvar,a}} \quad \text{and} \quad b_{s^{l+1}}^{\text{cvar,b}} = \frac{1}{|\text{sub}(s^{l+1})|} \sum_{s^l \in \text{sub}(s^{l+1})} b_{s^l}^{\text{cvar,b}} \tag{4.52}$$

The aggregation of station nodes with their approximate cost models is then done as follows. Let $\mathrm{sub}(s^{l+1}) \subseteq S^l$ denote the set of all station cells at level l which are to be aggregated into $s^{l+1} \in S^{l+1}$. The maximum number of parking slots naturally is the sum of the maximum values over all station nodes to be aggregated:

$$b^{\mathrm{ps}}_{s^{l+1}} = \sum_{s^l \in \mathrm{sub}(s^{l+1})} b^{\mathrm{ps}}_{s^l} \tag{4.53}$$

The model parameters are aggregated as follows:

Attractiveness values: As we coarsen the problem, we also have to aggregate the attractiveness values of merged stations for the merged customer cells. Hence, we take a weighted average value of all attractiveness values of all respective pairs of customer cells and station cells. In more detail, let $p^l, q^l \in C_{h-l-1}$ and v^{l+1} the node arising from merging all the nodes contained in the subtree rooted at p^l and let s^{l+1} be the resulting station from merging the stations in the subtree rooted at q^l. Then, we compute the attractiveness values as follows

$$a_{v^{l+1},s^{l+1}} = \begin{cases} 1 & \text{if } v = s \\ \frac{\sum_{v\in V(p)}\sum_{s\in V(q)}\left(a_{v,s}\cdot \max_{i\in K_s}\{b^{\mathrm{ps}}_{s,i}\}\right)}{\sum_{v\in V(p)}\sum_{s\in V(q)}\max_{i\in K_s}\{b^{\mathrm{ps}}_{s,i}\}} & \text{if } l = 0, v \neq s \\ \frac{\sum_{v^l\in V^l(p^l)}\sum_{s^l\in V^l(q^l)}\left(a_{v^l,s^l}\cdot \hat{b}^{\mathrm{ps}}_{s^l,l}\right)}{\sum_{v^l\in V^l(p^l)}\sum_{s^l\in V^l(q^l)}\hat{b}^{\mathrm{ps}}_{s^l,l}} & \text{if } l > 0, v \neq s \end{cases} \tag{4.54}$$

It is impossible to keep the original attractiveness values when coarsening the problem. Of course, we have to find a method to keep deviation to original attractiveness values to a minimum. We take the average between all pairs of customer cells and station cells weighted by the maximum possible parking slots at the particular stations. At level 0 we have to compute this value and find the maximum through all configurations K_s for the station s, and for all other levels we simply take the maximum parking slots as computed by the coarsening of the stations. If the customer cell v and the station cell s refer to the same cell, we set the attractiveness value to 1 which implies that every demand within this cell, i.e., self loops, are always satisfiable.

Initialization

Looking at Algorithm 2.8 Line 6 we have to initialize the solution at some reasonably coarsened level. We solve it via a MIP model when an appropriate level l, where the problem is well solvable, is reached. Parts of the model have been already shown, like calculating fulfillable demands or rebalancing costs in [84] and the optimization goal in Section 4.5.2.

As it is not important and also not meaningful to make decisions about exact station configurations in all levels $l > 0$ we use continuous variables $x^{\mathrm{flex}}_{s,l}$ for computing a completely flexible number of slots per station. The value 0 means no station is going to be built. Note, that variables are also indexed by the level $l \geq 0$ for which we compute the number of slots for the various stations. However, it is important that all stations which are going to be planned have a minimum number of slots which we denote as $x^{\text{flex}}_{\text{min}}$. We need this condition because it is not meaningful to plan a number of slots for a specific station under a particular minimum slot count. If the minimum slot count of some station configuration at level $l = 0$ is greater than the slot count planned at level $l = 1$ the information obtained through the coarsening and extension steps would be useless since no station can be built in this scenario. Due to this constraint, we introduce additional variables $g_s \in \{0, 1\}$ that decide whether a station is to be built in cell $s \in S$ or not.

The calculation of the demand is done by the linear program $D(x, t)$ defined in [84] but extended with an additional index for modeling the time periods.

Rebalancing effort is determined by the linear program $Q_x(s)$ is also defined in [84]. For the initialization it is also enriched by an additional index regarding the various prospective station candidates.

By $A^{p,q}_{\text{f},l}$ we denote the flow network of nodes/clusters (p, q) in level l. The set S^l corresponds to a set of possible station candidates at level l.

The MIP model for initialization of the solution at the coarsest level $l^{\max}$ is finally defined as follows.

$$\max \quad \sum_{t \in T} \left(\sum_{(v,p) \in A^{t,l^{\max}} | v \in V^{l^{\max}}} \quad \sum_{(v,s) \in A^{v,p}_{\text{f},l^{\max}}} f^{t,v,p}_{v,s} \right) \tag{4.55}$$

$$\text{s.t.} \quad \text{inequalities from } D(x,t) \text{ hold, see [84].} \tag{4.56}$$

$$\text{inequalities from } Q_x(s) \text{ hold, see [84].} \tag{4.57}$$

$$\sum_{s \in S} \left(b^{\text{cfix}} \cdot x^{\text{flex}}_{s,l^{\max}} + b^{\text{cvar}} \cdot x^{\text{flex}}_{s,l^{\max}} + b^{\text{reb}} \cdot \left(\sum_{t \in T} \left(r^{+}_{t,s} + r^{-}_{t,s} \right) + r^{\text{o}}_{s} \right) \right) \leq B^{\text{tot}}_{\max} \tag{4.58}$$

$$\sum_{s \in S} b^{\text{cfix}} \cdot x^{\text{flex}}_{s,l^{\max}} \leq B^{\text{fix}}_{\max} \tag{4.59}$$

$$x^{\text{flex}}_{s,l^{\max}} \geq g_s \cdot x^{\text{flex}}_{\text{min}} \quad \forall s \in S_{l^{\max}} \tag{4.60}$$

$$x^{\text{flex}}_{s,l^{\max}} \leq g_s \cdot x^{\text{flex}}_{\text{min}} \quad \forall s \in S_{l^{\max}} \tag{4.61}$$

$$x^{\text{flex}}_{s,l^{\max}} \geq 0 \quad \forall s \in S_{l^{\max}} \tag{4.62}$$

$$g_s \text{ binary} \quad \forall s \in S_{l^{\max}} \tag{4.63}$$

$$f^{t,v,p}_{\alpha,\beta} \geq 0 \quad \forall t \in T, (p,q) \in A_{l^{\max}}, (\alpha,\beta) \in A^{p,q}_{\text{f},l^{\max}} \tag{4.64}$$

$$r^{+}_{t,s}, r^{-}_{t,s}, y_{t,s} \geq 0 \quad \forall s \in S_{l^{\max}}, t \in T \tag{4.65}$$

$$r^{\text{o}}_{s} \geq 0 \quad \forall s \in S_{l^{\max}} \tag{4.66}$$

The objective (4.55) is to maximize the overall prospective demand. All inequalities defined in the maximum flow polytope (4.56) and all inequalities of the rebalancing polytope (4.57) must hold. The total budget (4.58) and the maximum allowed fixed budget (4.59) have to be respected. Inequalities (4.60) and (4.61) are used to ensure that a minimum number of stations is going to be built, if a station should be built in the particular station cell $s \in S$. Domain definitions are given in (4.62)–(4.66). All variables are continuous except the $g_s \mid s \in S$ variables which are binary decision variables.

At an appropriate level of the coarsening, the proposed MIP model is able to yield near optimal solutions in a reasonable runtime when solved with a state-of-the-art MIP solver such as Gurobi. As the underlying mathematical model is difficult to solve because of numerical issues and precision, it is useful to give the MIP solver a reasonable optimality gap. The actual optimality gap could vary from problem to problem and should be set according to practical observations. This initial solution is then further extended using the extension algorithm proposed in the next section.

Extension

The extension is done heuristically as in the lower levels the instance size gets larger and utilizing a MIP formulation becomes time consuming. We use a simple heuristic which is fast as after the extension step the solution is refined using local search.

For each station in the upper level we try to divide the number of slots to the corresponding children of this cluster node in the lower level. We make use of a priority queue which is sorted according to the highest possible demand which can be fulfilled by each node in the lower level. Thus, we compute the demand LP $D(x, t)$ with setting the solution vector x to the highest possible slot count so that we know how much demand this node would fulfill in the ideal case. At the end, the new solution for the lower level may be infeasible due to violations of the budget restrictions. If this is the case, we try to iteratively remove stations until we again reach a valid solution. For removal, we choose the station with the lowest satisfied demand.

At the end of the algorithm we obtain a valid solution for the lower level.

Refinement

As the solution evaluation is done by executing two linear-programming (LP) models, namely $D(x, t)$ to calculate the demand and $Q_x(s)$ to estimate the rebalancing costs from [84], it is expensive. Therefore, refinement is done by local search, but only a limited amount of time. The following neighborhoods are iterated in the given, static order until a predefined time limit or local optimum has been reached.

Change station: This neighborhood removes a station, i.e., assigns a station the 0 configuration and assigns another customer cell which has currently a 0 configuration some other configuration.

Add station: This neighborhood assigns a station, which has currently the 0 configuration, a new one. Here, also to concentrate on promising stations, the station among the 0 configuration stations is chosen, which has the highest demand.

Change configuration: This neighborhood changes the configuration of an arbitrary station candidate.

We consider only moves that construct valid solutions as it does not make sense to work with infeasible moves/solutions as repairing infeasible solutions consumes too much time due to the expensive evaluation function. Fixed and variable costs are checked before solution evaluation to avoid calling the LP for infeasible solutions. However, it can happen that newly computed solution is still infeasible when considering rebalancing costs. This information is retrieved from the LP and if this is the case the solution is simply discarded.

4.5.4 Computational Results

We derived our test instances[2] from real-world data of the city of Vienna. The whole city was partitioned into cells and a full demand matrix on these cells is given. Different instance sizes have been generated by picking some cells from the whole instance. The clustering on these instances was computed using the Nearest Point Algorithm. Consider cluster p_u and cluster p_v, where $V(p_u) = \{u_1, \ldots, u_n\}, V(p_v) = \{v_1, \ldots, v_m\}$. Among the remaining clusters, the nearest neighbor is chosen as follows:

$$d_{p_u,p_v} = \min_{u \in V(p_u), v \in V(p_v)} \{dist(u, v)\} \tag{4.67}$$

The graph $G^t = (C^t, A^t)$ was derived by utilizing Algorithm 4.1. Therefore, we set $\theta = 0.01$. The attractiveness values have also been given as input as a full matrix of attractiveness values between the different cells. However, to have computationally tractable instances we consider only subset of neighbors for a single cell. For instances with 20 and 50 customer cells we consider 10 neighbors for each cell, for instances with 300, 500 and 750 cells, 7 neighbors, for instances with 1,000 and 2,000 cells, 5 neighbors and for instances with 2,500, 3,000, 3,500, and 4,000 cells 3 neighbors. The neighbors with highest attractiveness values are taken. Configurations have been set as follows for all customer cells of the instances:

$$\forall s \in S : K_s = \{(0, 0, 0), (10, 17500, 10000), (20, 25000, 20000), (40, 30000, 40000)\}$$

These tuples represent the number of slots to be built for the particular configuration, its fixed and variable costs. Rebalancing a single bike for a single day has been estimated with 3 € which is $b^{\text{reb}} = 365 \cdot 3 = 1095$ € for a whole year which is equal to the planning horizon. The parameter for adjusting the attractiveness values is set accordingly to $\lambda = 0$.

[2] `https://www.ac.tuwien.ac.at/files/resources/instances/bsspp/lion19.bz2`

		Instance	**Multilevel Refinement Heuristic**				
#cells	$B^{\text{tot}}_{\max}$ [€]	$B^{\text{fix}}_{\max}$ [€]	obj	#coarsen	time [s]	totcost [€]	fixcost [€]
20	100,000.00	80,000.00	0.79	2	167.03	82,598.27	52,500.00
50	200,000.00	160,000.00	45.85	2	3,775.28	200,000.00	122,500.00
300	1,200,000.00	960,000.00	160.18	4	4,635.18	1,200,000.00	752,500.00
500	2,200,000.00	1,760,000.00	451.90	5	8,720.94	2,115,602.79	1,155,000.00
750	3,350,000.00	2,680,000.00	665.62	7	8,588.95	2,428,067.58	1,365,000.00
1,000	4,500,000.00	3,600,000.00	876.77	6	6,532.26	3,343,472.61	1,872,500.00
2,000	9,000,000.00	7,200,000.00	1,125.29	8	11,460.01	6,565,538.53	3,920,000.00
2,500	10,000,000.00	8,000,000.00	1,214.90	8	12,434.51	7,707,661.10	4,462,500.00
3,000	12,000,000.00	9,600,000.00	1,315.58	9	12,464.25	9,321,026.50	5,460,000.00
3,500	15,000,000.00	12,700,000.00	1,435.98	9	13,255.25	11,277,185.57	6,702,500.00
4,000	17,000,000.00	13,600,000.00	1,213.81	9	13,518.10	10,841,911.99	6,510,000.00
		Average	773.33	6.27	8,686.52	3,265,499.73	3,101,000.00

Table 4.2: Results for the multilevel refinement heuristic.

As the visualization component (see later) uses only one time interval, we also use only one time interval for our computational tests and set its duration to 1440 minutes. We introduced a flexible number of time intervals as it is important for practice. In our first experiments, however, we use only one time interval, as the visualization component is currently only able to visualize one time interval. This time interval uses an average prospective user demand over a whole day. We stop the coarsening step when we reached a maximum number of 128 customer cells. We set the time limit for the local search component to 15 seconds.

All algorithms are implemented in C++ and have been compiled with gcc 5.5.0. For solving the LPs and MIPs we used Gurobi 7.0. All experiments were executed as single threads on an Intel Xeon E5540 2.53GHz Quad Core processor. We executed a single run per instance which seems meaningful in the context of a long-term planning problem.

Result of the proposed method are shown in Table 4.2. For each instance we show the number of customer cells/station candidates (#cells), the maximum total budget ($B^{\text{tot}}_{\max}$), the maximum budget for the fixed costs ($B^{\text{fix}}_{\max}$), the objective value of the solution, i.e., number of trips (obj), the number of levels to be coarsened in the clustering tree such that we reach our goal number of nodes to initialize the solution (#coarsen), the time needed to find the solution (time), the total and the fixed costs of the found solution (totcost) and (fixcost). As shown in Table 4.2, the approach is able to solve instances derived from real-world data with up to 4,000 customer cells. Interesting is that for the instance with 1,000 customer cells we need less coarsening steps than for the instance with 750 customer cells, but this also strongly depends on the clustering and the depth of the original cluster tree. The second thing which looks interesting is that the fulfilled demand of the instance with 4,000 cells is less than the fulfilled demand for some instances with fewer customer cells. This happens because the refinement, i.e., local search, has fewer iterations for the instance with 4,000 customer cells and the *addStation* neighborhood is often successful. A detailed visualization of the solution found for the instance with 50 customers cells is given in Figure 4.4.

4.6 Conclusions and Future Work

We presented an innovative approach to the BSSPP. Previous work only considers very small instances and case studies to small parts of a city whereas we aim at solving more realistic large-scale scenarios arising in large cities. As we have to cope with thousands of customer cells and potential station cells it is most fundamental to model the potential demands efficiently. To this end, we proposed to use a hierarchical clustering and defining the demand graph on it. This approach can drastically reduce the data in comparison to a complete demand matrix with only a very reasonable information loss. Moreover, we provided MIP formulations to compute the satisfiable demand by given configurations and to compute the prospective rebalancing costs. Putting them together under the objective of maximizing the expected satisfied total demand and adding further constraints for complying with given monetary budget constraints, we obtained a MIP model that solves our definition of the BSSPP exactly. Because this MIP model can in practice still only be solved for rather small instances, we further suggested a multilevel refinement heuristic utilizing the same hierarchical clustering we are given as input.

Regarding the hierarchical clustering, we introduced an algorithm which constructs a sparse graph on clustered input data from the original full demand matrix. This drastically reduces the input size of the problem instance. We developed a refinement method based on local search for the multilevel refinement approach which improves the solutions after each extension step in the algorithm. Wee considered two problem variants, one considering flexible/fractional number of slots and one considering particular station configurations . Obviously, considering stations configurations increases the complexity of the algorithm and makes practical solvability more difficult but nevertheless is practically much more relevant.

In our first approach by solving the extension phase with a MIP model we solved randomly generated instances with 2000 stations. In the second approach we derived instances which are based on real-world data from the city of Vienna. Results seem reasonable and we were able to solve instances with 4000 prospective station candidates which demonstrate scalability of the proposed approach. Using the provided visualization the solutions can also visually be verified.

In future work we want to extend our benchmark suite with also other cities like e.g., Linz in Austria. Furthermore, the whole Vienna instance consists of 7216 prospective stations cells and it is interesting which instance sizes the proposed multilevel refinement can solve. Moreover, it is also interesting to study alternatives and improvements to the extension heuristic and the refinement part of the algorithm. Last but not least tests on small instances can be performed where exact solutions are compared to solutions of the multilevel refinement heuristic.

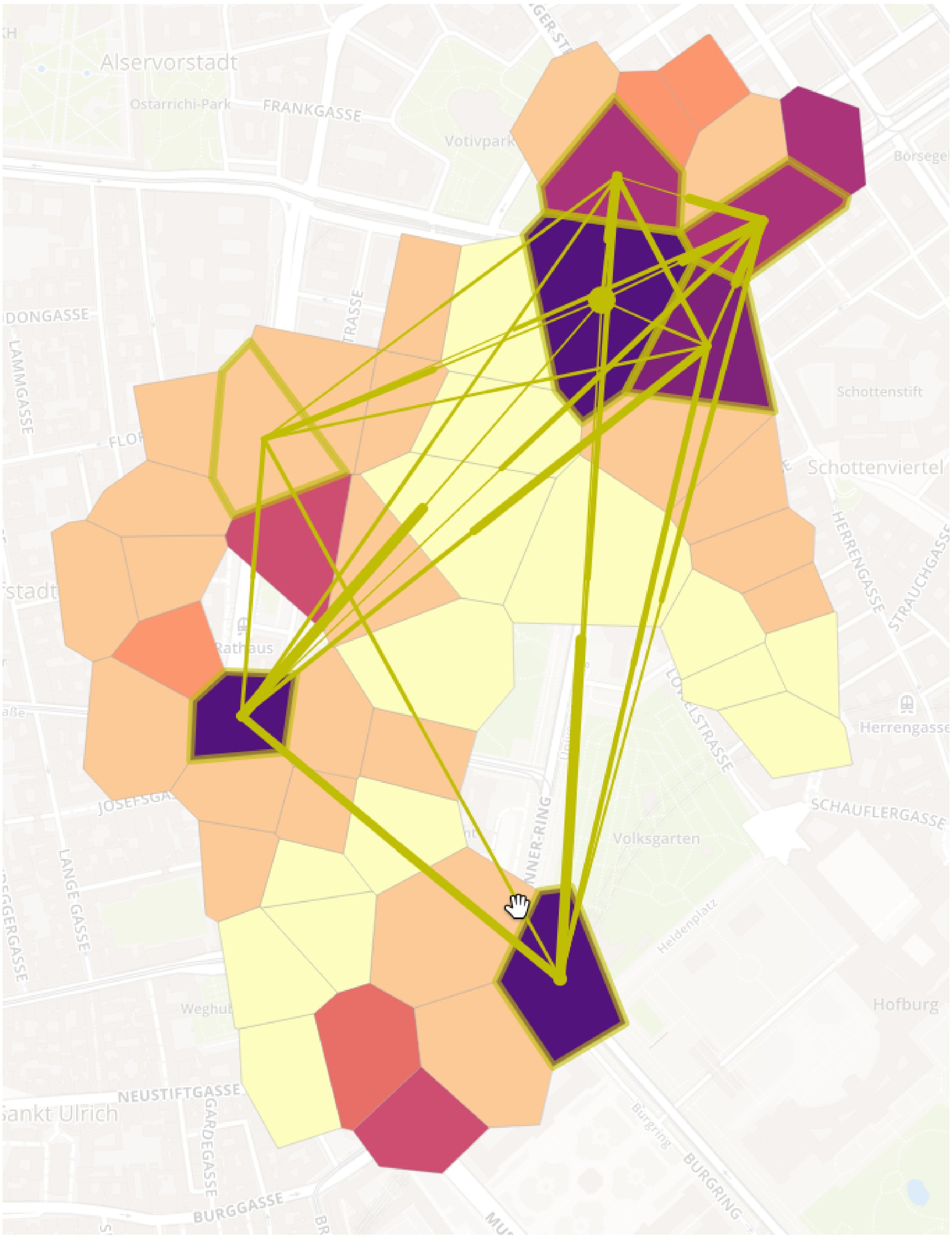


Figure 4.4: A solution to the instance for 50 customer cells is visualized. Station cells are marked by green rectangles. The darker a customer cell, the more demand is fulfilled for this cell. Flows between station cells are represented by green lines. The thicker a line, the more demand flows across the line. The visualization frontend tool has been engineered by Markus Straub and can be found under `https://bit.do/planbiss`.

CHAPTER 5

Districting and Routing Problem for Security Control

The last combinatorial optimization problem considered in this thesis is the Districting and Routing Problem for Security Control (DRPSC). First, we propose an approach inspired from the route elimination algorithm of Nagata and Bräysy [101] and then we present a variant of the problem with soft time windows and propose an efficient method for computing optimal arrival times.

5.1 Introduction

As in the area of private security control constant surveillance of an object might not be economically viable or even necessary, security firms face the problem of sending security guards to visit a large number of sites multiple times over the course of a day in order to fulfill their custodial duty.

Security companies have to schedule tours for their employees in order to cover all needed visits of all objects under their guardianship. The complexity of this task leaves a high potential for solving this problem by algorithmic techniques to minimize the number of needed tours. Thus, we propose the *Districting and Routing Problem for Security Control* (DRPSC) which consists of a districting part and a routing part. In the districting part all objects should be partitioned into a minimum number of disjoint districts, such that a single district can be serviced by a single security guard within each working day of a planning horizon. Given such a partitioning a routing problem has to be solved for each combination of district and day. We seek for a tour starting and ending at a central location which satisfies a maximum tour duration and the time window constraints for each requested visit. In case multiple visits are required at an object in the same period, there typically has to be a separation time between consecutive visits to ensure a

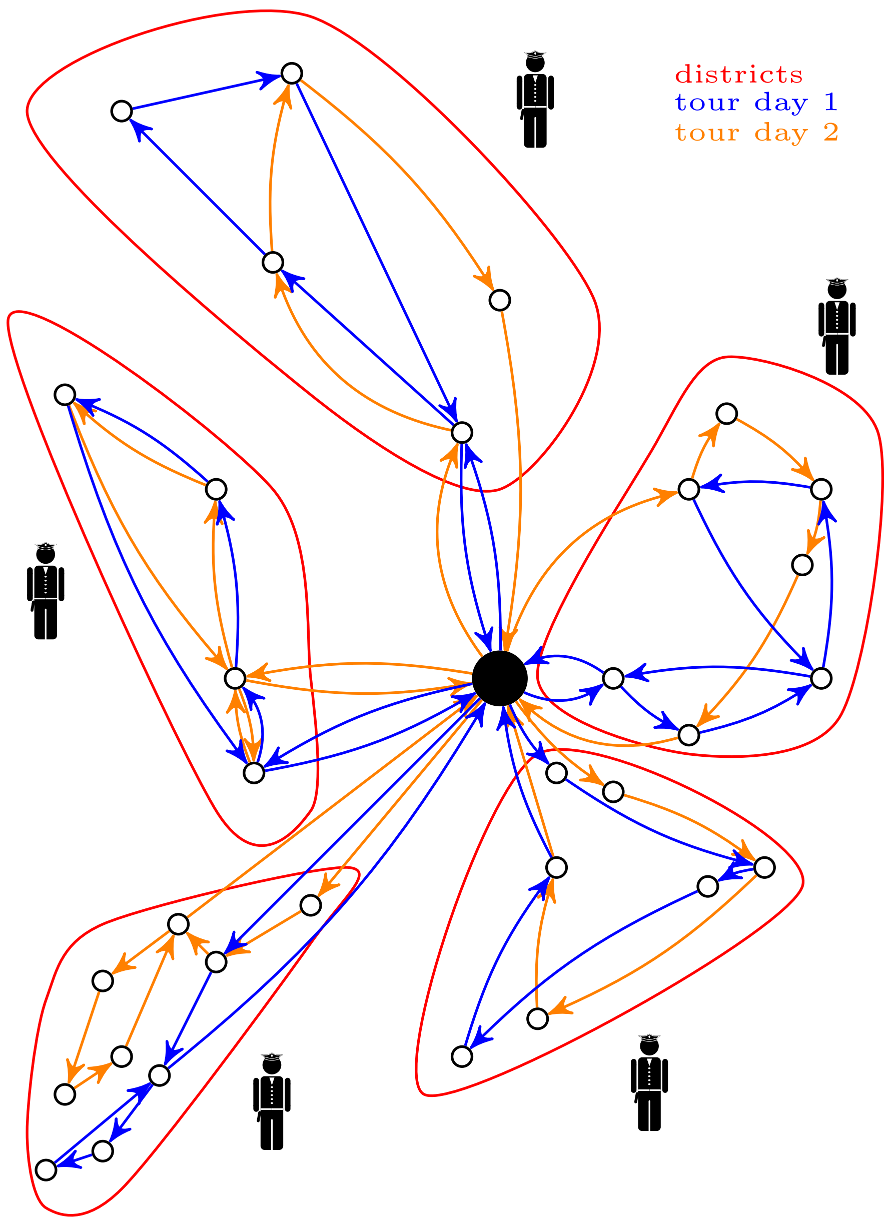


Figure 5.1: Example for a solution to the DRPSC [116]. This solution consists of four districts given in red. Blue and orange lines show tours for different days as visits of buildings may vary depending on the particular day. All tours for all districts and days must not exceed a given makespan. The objective value, which should be minimized, is the number of districts.

better distribution over time. To minimize the number of districts, it is important to minimize the duration of the planned tours in order to incorporate as many objects into the resulting districts as possible, which shows the inseparability of the districting and routing parts. A possible solution to the DRPSC is shown in Figure 5.1.

5.2 Solving the DRPSC by a Route Elimination Algorithm

This section provides construction heuristics for the DRPSC and also proposes an iterative destroy & recreate algorithm which is able minimize the routes needed for doing all visits of all objects.

We address the routing part by an exact *mixed integer linear programming formulation* (MIP) and a *routing construction heuristic* (RCH) with a subsequent *variable neighborhood descent* (VND). For the districting part we propose an *iterative destroy & recreate* (IDR) approach based on an initial solution identified by a *districting construction heuristic* (DCH).

This section is structured as follows. In Section 5.2.1 a formal problem definition of the DRPSC is given followed by a survey of related work in the literature in Section 5.2.2. The proposed algorithms for solving the routing subproblem and the districting problem are described in Sections 5.2.3 and 5.2.4, respectively. Computational results are shown and discussed in Section 5.2.5.

5.2.1 Problem Definition

This section formalizes the DRPSC. We are given a set of objects $I = \{1, \dots, n\}$ and a starting location, which we call in relation to the usual terminology in vehicle routing depot 0. There are p planning periods (days) $P = \{1, \dots, p\}$, and for each object $i \in I$ a set of visits $S_i = \{i_1, \dots, i_{|S_i|}\}$ is defined. Not all visits, however, have to take place in each period. The visits requested in period $j \in P$ for object $i \in I$ are given by subset $W_{i,j} \subseteq S_i$.

For each visit $i_k \in S_i$, $i \in I$, $k = 1, \dots, |S_i|$, we are given its duration $t^{\text{visit}}_{i_k} \geq 0$ and a time window $T_{i_k} = [T^{\text{e}}_{i_k}, T^{\text{l}}_{i_k}]$, during which the whole visit has to take place. The time windows of successive visits of an object may also overlap but visit i_k always has to take place before a visit $i_{k'}$ with $k, k' \in W_{i,j},\ k < k'$ and they must be separated by a minimum duration of t^{sep}. The maximum duration of each planned tour must not exceed a global maximum duration $t^{\max}$.

Next, we define underlying graphs on which our proposed algorithms operate. For each period $j \in P$ we define a directed graph $G^j = (V^j, A^j)$ where V^j refers to the set of visits requested at corresponding objects, i.e., $V^j = \bigcup_{i \in I} W_{i,j}$, and the arc set is: $A^j = \{(i_k, i'_{k'}) \mid i_k \in W_{i,j}, i'_{k'} \in W_{i',j}\} \setminus \{(i_k, i_{k'}) \mid i_k, i_{k'} \in W_{i,j}, k' \leq k\}$. We have arc weights associated with every arc in A^j which are given by $t^{\text{travel}}_{i,i'}$, the duration of the fastest connection from object i to object i'. We assume that the triangle inequality holds among these travel times. Let us further define the special nodes 0_0 and 0_1 representing the start and end of a tour and the augmented node set $\hat{V}^j = V^j \cup \{0_0, 0_1\}$, $\forall j \in P$. Accordingly, we add outgoing arcs from node 0_0 to all visits $i_k \in V^j$ and arcs from all visits $i_k \in V^j$ to node 0_1, formally, $\hat{A}^j = A^j \cup \{(0_0, i_k) \mid i_k \in V^j\} \cup \{(i_k, 0_1) \mid i_k \in V^j\}$. Consequently, we define the augmented graph $\hat{G}^j = (\hat{V}^j, \hat{A}^j)$.

The goal of the *DRPSC* is to assign all objects in I to a smallest possible set of districts $R = \{1, \ldots, \delta\}$, i.e., to partition I into δ disjoint subsets I_r, $r \in R$, with $I_r \cap I_{r'} = \emptyset$ for $r, r' \in R$, $r \neq r'$ and $\bigcup_{r \in R} I_r = I$, so that a feasible tour $\tau_{r,j}$ exists for each district I_r, $r \in R$ and each planning period $j \in P$. A tour $\tau_{r,j} = (\tau_{r,j,0}, \tau_{r,j,1}, \ldots, \tau_{r,j,l_{r,j}}, \tau_{r,j,l_{r,j}+1})$ with $\tau_{r,j,0} = 0_0, \tau_{r,j,l_{r,j}+1} = 0_1$, $l_{r,j} = \sum_{i \in I_r} |W_{i,j}|$, and $\tau_{r,j,1}, \ldots, \tau_{r,j,l_{r,j}} \in \bigcup_{i \in I_r} W_{i,j}$ has to start at the depot node 0_0, has to perform each visit $i_k \in W_{i,j}$ in the respective sequence for each object $i \in I_r$ exactly once, and finally has to return back to the depot, i.e., reach node (0_1). A tour $\tau_{r,j}$ is feasible if each visit $\tau_{r,j,u}$, $u = 0, \ldots, l_{r,j} + 1$ takes place in its time window T_{i_k}, where waiting before a visit is allowed, the minimum duration t^{sep} between visits of the same object is fulfilled, and the total tour duration does not exceed $t^{\max}$.

Note that the routing part can be solved for a given district I_r and each period $j \in P$ independently and consists of finding a feasible tour $\tau_{r,j}$.

5.2.2 Related Work

To the best of our knowledge there is no work covering all the aspects of the DRPSC as considered here. The similarity of the DRPSC to the vehicle routing problem with time windows (VRPTW), however, leads to a huge amount of related work. A majority of the approaches in the literature aim at minimizing the total route length without taking the makespan into account [32, 93, 98, 102, 103, 114, 143, 155]. As the practical difficulty usually increases when makespan minimization is considered, specialized algorithms have been developed for the traveling salesman problem with time windows (TSPTW) [24, 54], which is the specialization of the VRPTW to just one tour. The routing part of the DRPSC is similar to the TSPTW as the aim is to find a feasible tour of duration less than a prespecified value which is related to the minimization problem of the TSPTW. In the TSPTW, however, multiple visits of the same objects and a separation time between them are not considered. Interestingly, López-Ibáñez et al. [94] showed that by adapting two state-of-the-art metaheuristics for travel time minimization of the TSPTW [93, 103] to makespan minimization it is possible to outperform the specialized algorithms. Many of the proposed approaches focus on first minimizing the number of needed routes and only in a second step minimizing the travel time or makespan, e.g., by using a hierarchical objective function [102, 114]. Nagata and Bräysy [101] propose a route minimization heuristic which in particular tries to minimize the number of routes needed to solve the VRPTW. They also rely on a destroy-and-recreate heuristic which iteratively tries to delete one route while maintaining an ejection pool (EP). The EP stores all objects which are yet to be inserted. The algorithm tries to identify objects which are difficult to insert in one of the current routes and utilizes this information for choosing objects to be removed and re-inserted. As this approach produced excellent results we adopt this basic idea of the destroy-and-recreate heuristic here.

Exact solution approaches for the VRPTW were proposed by Ascheuer et al. [4] who developed a branch-and-cut algorithm using several valid inequalities and were able to solve most instances with up to 50–70 nodes. Baldacci et al. [6] introduce the ngL-tour relaxation. By using column generation as well as dynamic programming they are able to solve instances with up to 233 nodes to optimality and report new optimal solutions that have not been found previously. A current state-of-the-art method for heuristically solving several variants of the VRPTW is a hybrid genetic algorithm (GA) by Vidal et al. [155]. As many other approaches described in the literature [102, 114] they use a penalty function for handling infeasible routes, which is described in [102]. In the GA the initial solutions are created randomly but there are also more elaborate construction heuristics available: Solomon [143] proposes several algorithms for constructing only feasible solutions by extending the well-known savings heuristic, a nearest neighbor heuristic, and insertion heuristics using different criteria. Numerous simple construction heuristics for the asymmetric TSPTW are also proposed by Ascheuer et al. [4].

5.2.3 Routing Problem

An important factor when approaching the DRPSC is a practically efficient approach to the underlying routing problems. This part is embedded in the whole approach for optimizing the districting as a subcomponent which is called when the feasibility of a district needs to be checked. As already mentioned, this subproblem is similar to the well-known TSPTW which has been exhaustively studied in the literature. There is, however, one substantial and significant difference: objects have to be visited several times per period and between every two visits of the same object there has to be a specific separation time. Nevertheless, many fruitful ideas of the literature can be adopted to our problem.

As a single routing problem is solved for each period $j \in P$ and each district $r \in R$ independently, we are given one graph $G_r^j = (V_r^j, A_r^j)$. The node set is defined as $V_r^j = V^j \cap \bigcup_{i \in I_r} W_{i,j}$ and the arc set as $A_r^j = A^j \setminus \{(i_k, i'_{k'}) \mid i_k \notin V_r^j \vee i'_{k'} \notin V_r^j\}$. Similarly, we define the augmented graph containing the tours' start and end nodes 0_0 and 0_1 as $\hat{G}_r^j = (\hat{V}_r^j, \hat{A}_r^j)$ where $\hat{V}_r^j = \hat{V}^j \cap \bigcup_{i \in I_r} W_{i,j}$ and $\hat{A}_r^j = \hat{A}^j \setminus \{(i_k, i'_{k'}) \mid i_k \notin \hat{V}_r^j \vee i'_{k'} \notin \hat{V}_r^j\}$.

For computing the duration of a tour τ we first define the arrival and waiting times for each visit of the tour. Moreover, let us define the auxiliary function $\kappa : V_r^j \mapsto I$ which maps the visit $i_k \in V_r^j$, to its corresponding object $i \in I_r$, and the auxiliary function $\gamma : V_r^j \mapsto \mathbb{N}$ which maps visit $i_k \in V_r^j$, to its corresponding index in the set of visits for this particular object. For every visit $i_k \in V_r^j$, a_{i_k} denotes the arrival time at the object, whereas a_{0_0} and a_{0_1} denote the departure and arrival time for the depot nodes 0_0 and 0_1, respectively. Let $t_{\tau_u}^{\text{wait}} = \max(0, T_{\tau_u}^e - \max(a_{\tau_{u-1}} + t_{\tau_{u-1}}^{\text{visit}} + t_{\kappa(\tau_{u-1}),\kappa(\tau_u)}^{\text{travel}}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t_{\kappa(\tau_u)_{\gamma(\tau_u)-1}}^{\text{visit}} + t^{\text{sep}}))$ denote the waiting time before a visit τ_u can be fulfilled. We aim at finding a feasible tour $\tau = (0_0, \tau_1, \dots, \tau_l, 0_1)$, $\tau_1, \dots, \tau_l \in V_r^j, l = |V_r^j|$ through all visits starting and ending at the depot such that the total tour duration $T(\tau) = a_{0_1} - a_{0_0}$ does not exceed $t^{\max}$.

Exact Mixed Integer Linear Programming Model

The following compact mixed integer programming (MIP) model operates on the previously defined and reduced graph G_r^j and is based on Miller-Tucker-Zemlin (MTZ) [96] constraints. We use binary decision variables $y_{i_k,i'_{k'}}$ $\forall (i_k, i'_{k'}) \in A_r^j$ which are set to 1 if the arc between the k-th visit of object i and the k'-th visit of object i' is used in the solution, and 0 otherwise. We model arrival times by additional continuous variables a_{i_k} $\forall i_k \in V_r^j$ and ensure by these variables compliance with the time windows and the elimination of subtours. For each district $r \in R$ and each period $j \in P$ we solve the following model:

$$\min \quad \sum_{i_k \in V_r^j} (t_{i_k}^{\text{wait}} + t_{i_k}^{\text{visit}}) + \sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} (y_{i_k,i'_{k'}} \cdot t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})}) \tag{5.1}$$

$$\text{s.t.} \quad \sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} y_{i_k,i'_{k'}} = \sum_{(i'_{k'},i_k) \in \hat{A}_r^j} y_{i'_{k'},i_k} \qquad \forall i_k \in V_r^j \tag{5.2}$$

$$\sum_{(0_0,i_k) \in \hat{A}_r^j} y_{0_0,i_k} = 1 \tag{5.3}$$

$$\sum_{(i_k,0_1) \in \hat{A}_r^j} y_{i_k,0_1} = 1 \tag{5.4}$$

$$a_{i_k} - a_{i'_{k'}} + t^{\max} \cdot (1 - y_{i'_{k'},i_k}) \geq t^{\text{travel}}_{\kappa(i'_{k'}),\kappa(i_k)} + t^{\text{visit}}_{i'_{k'}} \qquad \forall i_k \in \hat{V}_r^j, (i_k, i'_{k'}) \in \hat{A}_r^j \tag{5.5}$$

$$a_{i_k} + t^{\text{travel}}_{0,\kappa(i_k)} \cdot (1 - y_{0_0,i_k}) \geq t^{\text{travel}}_{0,\kappa(i_k)} \qquad \forall (0_0, i_k) \in \hat{A}_r^j \tag{5.6}$$

$$t^{\text{wait}}_{i_k} + t^{\max} \cdot (1 - y_{i_k,i'_{k'}}) \geq a_{i'_{k'}} - a_{i_k} - t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})} - t^{\text{visit}}_{i_k} \qquad \forall i_k \in \hat{V}_r^j, (i_k, i'_{k'}) \in \hat{A}_r^j \tag{5.7}$$

$$a_{i_{k-1}} \leq a_{i_k} - t^{\text{sep}} \qquad \forall i_k, i_{k-1} \in V_r^j \tag{5.8}$$

$$\sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} y_{i_k,i'_{k'}} = 1 \qquad \forall i_k \in V_r^j \tag{5.9}$$

$$T^e_{i_k} \leq a_{i_k} \leq T^l_{i_k} - t^{\text{visit}}_{i_k} \qquad \forall i_k \in V_r^j \tag{5.10}$$

$$y_{i_k,i'_{k'}} \in \{0,1\} \qquad \forall (i_k, i'_{k'}) \in \hat{A}_r^j \tag{5.11}$$

The objective function (5.1) minimizes the total makespan within which all object visits take place by summing up all visit times, travel times, and waiting times. Equalities (5.2) ensure that the number of ingoing arcs is equal to the number of outgoing arcs for each node $i_k \in V_r^j$. Equalities (5.3) and (5.4) ensure that there must be exactly one ingoing and outgoing arc for the depot in each period $j \in P$. Inequalities (5.5) are used to recursively compute the arrival times for every visit. If an edge $(i_k, i'_{k'})$ is not used, then the constraint is deactivated. These inequalities can be individually lifted by using $(T^{\mathrm{l}}_{i'_{k'}} - t^{\mathrm{visit}}_{i'_{k'}}) + (t^{\mathrm{travel}}_{\kappa(i'_{k'}),\kappa(i_k)} + t^{\mathrm{visit}}_{i'_{k'}})$ instead of t^{tmax}, which is also done in our implementation. Inequalities (5.6) set the start time at the depot for each period. Inequalities (5.7) compute the waiting time at the k-th visit of object i before traveling to the k'-th visit of object i'. We need these waiting times $t^{\mathrm{wait}}_{i_k}\ \forall i_k \in V^j_r$ for the objective function to minimize the makespan of the route. These inequalities can also be lifted by replacing $t^{\max}$ with the term $(T^l_{i'_{k'}} - t^{\mathrm{visit}}_{i'_{k'}}) - T^l_{i_k} - t^{\mathrm{travel}}_{\kappa(i_k),\kappa(i'_{k'})} - t^{\mathrm{visit}}_{i_k}$. Inequalities (5.8) model the minimum time required between two different visits of the same object, i.e., ensure the separation time t^{sep}. Inequalities (5.9) state that there must exist an ingoing and an outgoing arc for the k-th visit of object i, if this particular visit is requested in the considered period $j \in P$. It is ensured that every time window of every visit $i_k \in V^j_r$ is fulfilled in (5.10). In (5.11) the domain definitions for the binary edge-decision variables $y_{i_k,i'_{k'}}$ are given.

In the context of the districting problem we use this model only for checking feasibility which can usually be done faster than solving the optimization problem to optimality. To this end we replace the objective function by $\min\{0\}$ and add the following constraints for limiting the makespan to $t^{\max}$:

$$\sum_{i_k \in V^j_r} (t^{\mathrm{wait}}_{i_k} + t^{\mathrm{visit}}_{i_k}) + \sum_{(i_k,i'_{k'}) \in \hat{A}^j_r} (y_{i_k,i'_{k'}} \cdot t^{\mathrm{travel}}_{\kappa(i_k),\kappa(i'_{k'})}) \leq t^{\max} \tag{5.12}$$

Heuristics

For larger districts the exact feasibility check using the MIP model might be too slow, hence we also propose a faster greedy construction heuristic followed by a variable neighborhood descent.

Given a sequence of visits τ, we first determine if a tour can be scheduled such that the time window constraints of all visits are satisfied. For this purpose, we compute the earliest possible arrival time a_{i_k} for each visit and minimize waiting times.

Feasibility of a tour: Since the (intermediate) tour τ starts at the depot at the earliest possible time, the departure at the depot a_{0_0} is set to 0. For each subsequent visit τ_u, the arrival time a_{τ_u} is the maximum of $T^e_{\tau_u}$ and the arrival time at the preceding visit $a_{\tau_{u-1}}$ including visit time $t^{\mathrm{visit}}_{\tau_{u-1}}$ and travel time $t^{\mathrm{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}$ from the preceding visit's object $\kappa(\tau_{u-1})$ to the current visit's object $\kappa(\tau_u)$. The depot has no requested visit times, therefore we define $t^{\mathrm{visit}}_{0_0} = t^{\mathrm{visit}}_{0_1} = 0$. Furthermore, for each object i the separation time t^{sep} between visit i_k and i_{k-1} for all $k > 1$ has to be respected. Formally:

$$
\begin{aligned}
a_{0_0} &= 0\\
a_{\tau_u} &= \begin{cases}
\max\{T^e_{\tau_u}, a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}\} & \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{for } u > 1, \gamma(\tau_u) = 1 & \\
\max\{T^e_{\tau_u}, a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}, & \\
\qquad a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}\} & \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{for } u > 1, \gamma(\tau_u) > 1 &
\end{cases}\\
a_{0_1} &= a_{\tau_l} + t^{\text{visit}}_{\tau_l} + t^{\text{travel}}_{\kappa(\tau_l),0}
\end{aligned}
$$

If for any arrival time a_{i_k} with $i_k \in V^j_r$ the following condition is violated, the sequence of visits is infeasible:

$$
a_{i_k} + t^{\text{visit}}_{i_k} \leq T^{\text{l}}_{i_k} \tag{5.13}
$$

The resulting tour duration $T(\tau) = a_{0_1} - a_{0_0}$ can be minimized while keeping τ feasible by delaying the departure at the depot by the so called *forward time slack* proposed by Savelsbergh [136] for the TSPTW. The *forward time slack* $F(\tau_u, \tau_{u'})$ for the partial tour $\tau' = \tau_u, \dots, \tau_{u'}$ adapted to our problem is

$$
\begin{aligned}
F'(\tau_u, \tau_{u'}) &= \begin{cases}
T^l_{\tau_u} - t^{\text{visit}}_{\tau_u} - a_{\tau_u} & \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{for } u = u' & \\
F'(\tau_u, \tau_{u'-1}) - T^l_{\tau_{u'-1}} + T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{travel}}_{\kappa(\tau_{u'-1}),\kappa(\tau_{u'})} & \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{for } u' > 1, \gamma(\tau_{u'}) = 1 & \\
\min\{F'(\tau_u, \tau_{u'-1}) - T^l_{\tau_{u'-1}} + T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{travel}}_{\kappa(\tau_{u'-1}),\kappa(\tau_{u'})}, & \\
\qquad F'(\tau_u, \tau_{\kappa(\tau_{u'})_{\gamma(\tau_{u'})-1}}) - T^l_{\tau_{\kappa(\tau_{u'})_{\gamma(\tau_{u'})-1}}} + T^l_{u'} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{sep}}\} & \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{for } u' > 1, \gamma(\tau_{u'}) > 1 &
\end{cases}\\
F(\tau_u, \tau_{u'}) &= \min_{v=u,\dots,u'} \{F'(\tau_u, \tau_v)\}
\end{aligned} \tag{5.14}
$$

The tour duration of tour τ must not exceed $t^{\max}$. Formally, if

$$
T(\tau) - \min(F(0_0, 0_1), \sum_{u=1}^{l} t^{\text{wait}}_{\tau_u}) < t^{\max} \tag{5.15}
$$

holds, the sequence τ is feasible, otherwise infeasible.

Routing Construction Heuristic: We developed a *Routing Construction Heuristic* (RCH) based on an insertion heuristic by starting from a partial tour $\tau' = (0_0, 0_1)$ containing only the start and end nodes and iteratively adding all visits $i_k \in V_r^j$ to τ'. A 2-step approach is used, where we first order the visits according to some criteria and then insert them at the first feasible or best possible insert position, respectively. For the insertion order we compute the *flexibility value* of each visit $i_k \in V_r^j$ where visits with less flexibility are inserted first:

$$\mathit{flex}(i_k) = T^l_{i_k} - T^e_{i_k} - t^{\text{visit}}_{i_k} \tag{5.16}$$

$$\mathit{flex}(i_k^{(1)}) \leq \mathit{flex}(i_k^{(2)}) \leq \cdots \leq \mathit{flex}(i_k^{(|V_r^j|)}) \tag{5.17}$$

Visits with less flexibility may be more difficult to insert as they need to be scheduled at a very specific time. Ties are broken randomly.

In a second phase we start by trying to insert the first visit, i.e., $i_k^{(1)}$, into the partial tour τ'. We start at the front, i.e., try to insert it after the start node 0_0, and move backwards to the end. Then, we either stop when we found the first feasible insert position in the first feasible variant or we compute insertion costs for each possible insert position and insert the visit at the position with the minimum costs for the best possible insertion variant. We define these costs as:

$$d_{i_k,u'} = \begin{cases} a_{\tau_{u'}} + t^{\text{visit}}_{\tau_{u'}} + t^{\text{travel}}_{\kappa(\tau_{u'}),\kappa(\tau_u)} - a_{\tau_u} & \text{if (5.19) and (5.20) hold} \\ \infty & \text{otherwise} \end{cases} \tag{5.18}$$

These insertion costs $d_{i_k,u'}$ determine the amount of time by which the visit τ_u has to be moved backwards in order to insert the new visit $\tau_{u'}$. Note that $d_{i_k,u'}$ may also be negative, if the space for insertion of visit $\tau_{u'}$ is bigger than necessary. However, this is desirable as we use those insert positions more likely which have bigger gaps and smaller gaps are kept for later inserts. In Section 5.2.5 we compare both, the first feasible and the best possible variant, to each other in terms of solution quality and runtime.

We further maintain global variables for the *forward time slack* $F(\tau')$ and all arrival times a_{τ_u} of each partial tour τ' computed during the execution of the insertion heuristic. For an insertion to be feasible, the latest allowed arrival time at visit $\tau_{u'}$ must be greater or equal to the earliest possible arrival at that visit considering the previous visit's earliest arrival, its visit time and the travel time between τ_{u-1} and $\tau_{u'}$. Furthermore, the earliest departure at $\tau_{u'}$ including the travel time between $\tau_{u'}$ and τ_u must be smaller or equal to the earliest arrival at τ_u delayed by the *forward time slack* of the partial tour from τ_u to the depot. Using the definition of the forward time slack in equality (5.14) and if inequality (5.13) holds, then the insertion is feasible if, in addition, also the following two inequalities hold:

$$T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} \geq \max\{a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_{u'})}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}\} \quad (5.19)$$

$$a_{\tau_{u'}} + t^{\text{visit}}_{\tau_{u'}} + t^{\text{travel}}_{\kappa(\tau_{u'}),\kappa(\tau_u)} \leq a_{\tau_u} + F(\tau_u, 0_1) \quad (5.20)$$

Local improvement: If the solution found by the RCH is infeasible we additionally employ a VND to reduce the number of infeasibilities and possibly come to a feasible solution. First, we insert each infeasible visit i_k into the tour on the position u' where the costs $d_{i_k,u'}$ are minimum. We use a lexicographical penalty function to penalize infeasible tours where the first criterion is the number of time window violations and the second criterion is the duration of the route as proposed by López-Ibáñez et al. [93]. We use three common neighborhood structures from the literature and search them in a best improvement fashion in random order while respecting the visit order:

Swap: This neighborhood considers all exchanges between two distinct visits.

2-opt: This is the classical 2-opt neighborhood for the traveling salesman problem where all edge exchanges are checked for improvement.

Or-opt: This neighborhood considers all solutions in which sequences of up to three consecutive visits are moved to another place in the same route.

If at some point during the algorithm the value of the penalty function is zero we terminate with a feasible solution.

5.2.4 Districting Problem

In the previous section we have already introduced a fast heuristic for efficiently testing feasibility of a given set of objects by building a single tour for each period through all requested visits of these objects. In the districting part of the DRPSC we face the problem of intelligently assigning objects to districts such that the number of districts is minimized. For checking the feasibility of this assignment we use the previously introduced RCH. Alternatively, we could also use our MIP model for solving these subproblems but, as we will see in Section 5.2.5, it is too slow to be used in practical scenarios. We propose a DCH and an iterative destroy & recreate algorithm where the former generates an initial solution and the latter tries to iteratively remove districts.

Districting Construction Heuristic

Starting with one district, objects are iteratively added to the existing districts $r \in R$. Whenever adding an object $i \in U$ to any of the available districts $r \in R$ would make the assignment infeasible, $i \in U$ is added to a newly created district r'. The overall DCH is shown in Algorithm 5.1 and explained below.

Algorithm 5.1: Districting Construction Heuristic

```
1:  init: $R \leftarrow \{1\}, U \leftarrow sort(I)$
2:  for all $i \in U$ do
3:    $inserted \leftarrow$ false
4:    for all $r \in R$ do
5:      if $insert(i, r)$ then
6:        $inserted \leftarrow$ true
7:        break
8:      end if
9:    end for
10:   if not $inserted$ then
11:     $r' \leftarrow$ create $|R| + 1$-th new empty district
12:     $R \leftarrow R \cup \{r'\}$
13:     $insert(i, r')$
14:   end if
15: end for
```

First, the set of districts R is initialized with the first empty district 1 and the set of objects I is sorted by extending the flexibility values as defined in equation (5.16) from visits to objects. All objects are sorted by the sum of their flexibility values $\sum_{j \in P} \sum_{i_k \in W_{i,j}} flex(i_k)$ in ascending order. As in the RCH, the resulting set U is denoted as the set of unscheduled visits. The DCH terminates when all $i \in U$ have been scheduled (2) and, as a consequence, all requested visits have been inserted successfully and we obtain a feasible solution to the DRPSC. The insertion of object i into district r (lines 5 and 13) is accomplished by checking for each scheduled visit $i_k \in W_{i,j}$ if i_k can be feasibly inserted into the particular district r (for the definition of feasibility of a tour see also Section 5.2.3). In line 5 the DCH inserts i either into the first feasible or into the best possible insert position, as described in Section 5.2.3. The *insert* function returns *false*, if no feasible insertion position is found for at least one $i_k \in W_{i,j}$, $\forall j \in P$. It returns *true*, if a feasible insertion position is found for each visit $i_k \in W_{i,j}$, $\forall j \in P$. If the loop over all districts (line 4) terminates without finding any feasible insertion position the variable *inserted* stays *false* and a new empty district is created in line 11. The proposed constructive algorithm will terminate with a feasible solution after $|U|$ iterations.

Iterative Destroy & Recreate

Nagata and Bräysy [101] proposed a *route elimination algorithm* for reducing the number of vehicles needed in the VRPTW. We apply the basic idea to the districting problem. The algorithm starts with the initial assignment where every object is reached by a separate route. Then, one district $r \in R$ is chosen for elimination at a time, maintaining all now unassigned objects in an *ejection pool* (EP). Then, it is tried to assign all objects of the EP to the remaining districts $R \setminus \{r\}$. If this is successful, the number of districts

Algorithm 5.2: District elimination algorithm

```
1:  init: $EP \leftarrow \emptyset,\ c_i \leftarrow 0\ \forall i \in I$
2:  choose a district $r^{\mathrm{del}} \in R$ for deletion
3:  $R \leftarrow R \setminus \{r^{\mathrm{del}}\}$
4:  $EP \leftarrow EP \cup \{i \mid i \in I_{r^{\mathrm{del}}}\}$
5:  while $EP \neq \emptyset \wedge$ termination criterion not met do
6:    $i^{\mathrm{ins}} \leftarrow \arg\max_{i \in EP} \{c_i\}$
7:    $R_f \leftarrow$ feasible districts for assignment of $i^{\mathrm{ins}}$
8:    $c_{i^{\mathrm{ins}}} \leftarrow c_{i^{\mathrm{ins}}} + |R| - |R_f|$
9:    if $R_f \neq \emptyset$ then
10:     assign object $i^{\mathrm{ins}}$ to a randomly chosen feasible district $r \in R_f$
11:   else
12:     select random district $r^{\mathrm{ins}} \in R$
13:     assign object $i^{\mathrm{ins}}$ to district $r^{\mathrm{ins}}$
14:     call VND for district $r^{\mathrm{ins}}$ (see Section 5.2.3)
15:     while $\exists$ an infeasible tour for any period of district $r^{\mathrm{ins}}$ do
16:       $i^{\mathrm{del}} \leftarrow \arg\min_{i \in I_{r^{\mathrm{ins}}}} \{c_i\}$
17:       $I_{r^{\mathrm{ins}}} \leftarrow I_{r^{\mathrm{ins}}} \setminus \{i^{\mathrm{del}}\}$
18:       $EP \leftarrow EP \cup \{i^{\mathrm{del}}\}$
19:       call VND for district $r^{\mathrm{ins}}$ (see Section 5.2.3)
20:     end while
21:   end if
22: end while
```

could be reduced by one and another district is chosen for elimination. We adapt this idea to the DRPSC and use the result of the DCH described in Section 5.2.4 as initial solution.

Let the assignment of an object $i \in I$ to a district $r \in R$ be feasible if and only if a feasible tour can be scheduled for all assigned visits of all objects for each period. Let c_i be a penalty value of object $i \in I$ denoting failed attempts of inserting object i into a district. Each time a visit cannot be inserted, this penalty value is increased by one, revealing objects which are difficult to assign to one of the available districts.

If the EP becomes empty, a feasible assignment of objects to districts is found. Subsequently, another iteration is started, destroying a district and reassigning its objects to the remaining ones. The overall district elimination algorithm is shown in Algorithm 5.2.

First, the EP is initialized to the empty set and the penalty values of all objects are set to 0. Starting with the solution provided by DCH a district is chosen for elimination in line 2. One of the following strategies is applied uniformly at random for selecting a district for elimination:

Minimum number of scheduled visits: This implies that only a minimum number of visits has to be reinserted to regain a feasible solution.

Shortest tour duration: Selecting a district where the maximum tour duration over all periods is minimal can be promising because this district might lead to a district with visits of shorter durations resulting in easier insert operations.

Maximum waiting times: Selecting a district with a loose schedule may indicate less or shorter visits, making them easier to reinsert.

After deleting a district all objects of this district are moved to the EP (line 4). As long as the EP contains objects, we try to assign each object to one of the remaining districts. An object with maximum penalty value is chosen for the next assignment (6). For the chosen object i^{ins} the feasible districts for assignment are computed. If there is at least one feasible district for an assignment of object i^{ins} (9) we assign the object to such a district uniformly at random (10). If it is not possible to feasibly assign the object i^{ins} to any of the remaining districts we randomly choose a district for assignment (11). Then, we apply the VND described in Section 5.2.3 trying to make the district feasible. If this is not possible and the assignment is still infeasible we iteratively try to remove objects with lowest penalty values from this district r^{ins} in the following loop (15), remove them from district r^{ins} (17), and finally add them to the *EP* (18). Then again, we call the VND from Section 5.2.3 trying to make the resulting tour from the actual assignment feasible. After an iteration of the outer loop the object with highest penalty value of the *EP* has been inserted and other objects previously assigned to this district may have been added to the *EP*. The idea behind this approach is to insert difficult objects first and temporarily remove easy to insert objects from the solution to reinsert them later. When the EP is empty, a new best assignment with one district less is found. This algorithm iterates until a termination criterion, e.g., a time limit is met.

5.2.5 Computational Results

To evaluate our proposed algorithm computational tests are performed on a benchmark set of instances. As the DRPSC is a new problem we created new instances[1] based on the characteristics of real-world data provided by an industry partner. The main characteristics of real-world data are: Most of the time windows are of medium size, the depot is centralized among the objects, travel times are rather small with respect to visit times and the number of visits of the objects is usually ranged from 1 to 4. The distance matrix is taken from TSPlib instances and we added the depot, the visits and the time windows in the following way: The depot is selected by taking the node for which the total distance to all other nodes is a minimum. Each node of the original instance has between 1 and v visits, where v is a parameter of the instance. Small time windows have a length between 5 and 30 minutes, medium time windows have a length of 2, 3, 4, or 5

[1]`https://www.ac.tuwien.ac.at/files/resources/instances/drpsc/hm16.tar.gz`

Instance						**RCH**		**RCH-VND**		**MIP**				**Rel. Difference**	
name	$\|I\|$	$\|V\|$	α	β	v	*obj*	t[s]	*obj*	t[s]	UB	LB	Gap	t[s]	Δ_{MIP}	Δ_{RCH}
burma14_01	13	19	0	0.2	2	495.80	< 0.01	333.49	0.03	332.62	332.62	0.00%	2025.56	0.26%	48.67%
burma14_02	13	20	0	0.2	2	624.47	< 0.01	374.60	0.02	352.93	343.50	2.67%	3600.00	8.30%	66.70%
burma14_03	13	21	0	0.2	2	525.49	0.01	440.86	0.05	433.42	421.91	2.66%	3600.00	4.30%	19.20%
burma14_04	13	19	0	0.2	2	607.11	< 0.01	397.14	0.03	395.15	387.89	1.84%	3600.00	2.33%	52.87%
burma14_05	13	22	0	0.2	2	606.07	< 0.01	409.24	0.03	356.71	337.43	5.40%	3600.00	17.55%	48.10%
burma14_06	13	19	0	0.2	2	409.54	< 0.01	273.28	0.01	272.93	272.93	0.00%	2318.17	0.13%	49.86%
burma14_07	13	23	0	0.5	2	714.04	< 0.01	508.69	0.05	493.48	456.82	7.43%	3600.00	10.20%	40.37%
burma14_08	13	17	0	0.5	2	372.63	< 0.01	312.70	0.02	311.10	311.10	0.00%	2.09	0.51%	19.16%
burma14_09	13	21	0	0.5	2	416.55	< 0.01	370.61	0.04	360.71	360.71	0.00%	3.35	2.67%	12.40%
burma14_10	13	20	0	0.5	2	386.02	< 0.01	342.00	0.02	336.25	336.25	0.00%	10.74	1.68%	12.87%

Table 5.1: Results of the MIP, RCH, and RCH-VND for the routing part.

hours, and visits with large time windows are unrestricted. For the instance generation the length of a time window is assigned randomly to a visit based on parameter values α and β: a small time window is chosen with probability α, a medium time window with probability β, and a large time window with probability $1 - \alpha - \beta$. Furthermore, we enforce that small and medium time windows of visits of the same object do not overlap and we choose the visit time uniformly at random from 3 to 20 minutes. For all our instances we set t^{sep} to 60 minutes and $t^{\max}$ to 10 hours.

The algorithm is implemented in C++ using Gurobi 6.5 for solving the MIP. For each combination of configuration and instance we performed 20 independent runs for the IDR while for the routing part we performed only one run because all tested algorithms for the routing part are deterministic. All runs were executed on a single core of an Intel Xeon processor with 2.54 GHz. The iterative destroy & recreate algorithm is terminated after a maximum of 900 CPU seconds. The MIP model for the routing part was aborted after 3600 CPU seconds.

In the first set of experiments the routing part of the DRPSC is examined more closely to evaluate RCH in comparison to the MIP model. Then, several configurations of our proposed algorithm for the whole problem are investigated.

Routing Part

First, the methods for the routing part are evaluated on a separate set of benchmark instances. In Table 5.1 the MIP model is compared to RCH, and RCH with the subsequent VND, denoted by RCH-VND. As the goal for the routing part is to minimize the makespan of a specific route, the maximum tour duration constraint is relaxed and the resulting makespan is given in minutes in the column *obj*. In the first four columns the instance parameters are specified. Sequentially, the instance name, the number of objects $|I|$, the maximum number of nodes of all objects $|V|$, the percentage of small (α), and medium time windows (β) and the maximum number of visits per objects v is given. For the RCH and RCH-VND we give the objective value (makespan in minutes) and the time needed for solving the instance. Then, the upper bound (UB), the lower bound (LB), the final optimality gap, and the time spent by Gurobi for solving the MIP model is shown. In the two remaining columns we present the relative gap between the MIP and RCH-VND $\Delta_{\text{MIP}} = (obj_{\text{RCH-VND}} - LB)/obj_{\text{RCH-VND}}$ as well as the relative gap between RCH and RCH-VND $\Delta_{\text{RCH}} = (obj_{\text{RCH}} - obj_{\text{RCH-VND}})/obj_{\text{RCH-VND}}$.

In Table 5.1 we see that the MIP model is able to solve easier instances to optimality, but soon has very high running times. RCH-VND yields very reasonable solutions with objective values close to the LB of the MIP for most cases. When looking at the relative gap between the RCH and RCH-VND (Δ_{RCH}), we can conclude that the VND improves greatly on the objective value with only a minor increase in running time. Moreover, Δ_{MIP} reveals that RCH-VND produces results close to the results of the MIP, and for those instances where the relative gap between the MIP and RCH-VND is greater than 10%, the MIP also has a relatively larger gap between UB and LB.

As we require a fast method for deciding if a route is feasible within the districting problem, we conclude that RCH-VND is a reasonable choice.

Districting Part

For testing the proposed algorithms for the DRPSC we used three different configurations. In the IDR-DCH$^{\text{f}}$ algorithm we used the DCH for generating an initial feasible solution candidate with the first feasible strategy in contrast to the IDR-DCH$^{\text{b}}$ where we used a best possible strategy. Both configurations are compared with the IDR-SCH, where a simple construction heuristic (SCH) as proposed by Nagata and Bräysy [102] is used. In the SCH each object is put in a separate district which results in a trivial initial solution candidate.

In Table 5.2 the results of the experiments are shown. Columns *obj* show the average objective value, i.e., the minimum number of districts at the end of optimization, after the full run of IDR while columns obj_{f} and obj_{b} show the average objective value, i.e., the average number of districts, after the respective construction heuristic. Columns t^* show the median time in seconds after which the best solution has been found during the run of IDR while t_{f} and obj_{b} show the median time after which the respective construction heuristic has found an initial solution. Columns *sd* show the standard deviation of the objective value for 20 runs of a single instance.

We observe that for most instances the final objective value of the IDR is the same for all three configurations. There are, however, differences for the construction heuristics alone and DCH$^{\text{b}}$ for most but not all instances better results but needed more time. The IDR-SCH works surprisingly well and was able to find good results in about the same amount of time as the other two (more sophisticated) configurations.

Instance							**IDR-SCH**			**IDR-DCH$^{\text{f}}$**					**IDR-DCH$^{\text{b}}$**				
name	runs	$\lvert I\rvert$	$\lvert V\rvert$	α	β	v	*obj*	*sd*	t^*[s]	obj_{f}	t_{f}[s]	*obj*	*sd*	t^*[s]	obj_{b}	t_{b}	*obj*	*sd*	t^*[s]
st70_1	20	69	105	0.1	0.7	2	3.0	0.0	11	6	< 0.1	3.0	0.0	5	5	0.1	3.0	0.0	9
st70_2	20	69	91	0.1	0.7	2	3.0	0.0	4	6	< 0.1	3.0	0.0	6	5	0.1	3.0	0.0	6
st70_3	20	69	106	0.1	0.7	2	3.0	0.0	11	6	< 0.1	3.0	0.0	13	5	0.1	3.0	0.0	11
rd100_1	20	99	152	0.1	0.5	2	6.0	0.0	8	9	< 0.1	6.0	0.0	10	9	0.1	6.0	0.0	14
rd100_2	20	99	160	0.1	0.5	2	6.0	0.0	16	8	0.1	6.0	0.0	14	8	0.1	6.0	0.0	6
rd100_3	20	99	152	0.1	0.5	2	6.0	0.0	3	7	0.1	6.0	0.0	2	8	0.1	6.0	0.0	7
tsp225_1	20	224	334	0.2	0.7	2	11.0	0.0	47	13	0.2	11.0	0.0	64	13	0.4	11.0	0.0	121
tsp225_2	20	224	341	0.2	0.7	2	11.0	0.0	46	13	0.3	11.0	0.0	67	13	0.5	11.0	0.0	36
tsp225_3	20	224	332	0.2	0.7	2	10.0	0.0	226	12	0.3	10.0	0.0	257	12	0.5	10.0	0.0	177
gr48_1	20	47	120	0.2	0.7	4	5.0	0.0	6	8	< 0.1	5.0	0.0	4	7	< 0.1	5.0	0.0	14
gr48_2	20	47	115	0.2	0.7	4	5.0	0.0	12	7	< 0.1	5.0	0.0	7	7	< 0.1	5.0	0.0	10
gr48_3	20	47	125	0.2	0.7	4	4.0	0.0	527	7	< 0.1	4.0	0.0	438	6	< 0.1	4.0	0.0	488
berlin52_1	20	51	133	0.0	0.7	4	5.0	0.0	3	6	< 0.1	5.0	0.0	2	7	0.1	5.0	0.0	7
berlin52_2	20	51	130	0.0	0.7	4	5.0	0.0	5	7	< 0.1	4.0	0.0	142	6	0.1	5.0	0.0	1
berlin52_3	20	51	140	0.0	0.7	4	5.0	0.0	19	7	< 0.1	5.0	0.0	14	6	0.1	5.0	0.0	16
ft70_1	20	69	167	0.1	0.5	4	8.0	0.0	12	12	< 0.1	8.0	0.0	12	10	0.1	8.0	0.0	18
ft70_2	20	69	180	0.1	0.5	4	8.0	0.0	33	11	< 0.1	8.0	0.0	15	11	0.1	8.0	0.0	18
ft70_3	20	69	144	0.1	0.5	4	7.0	0.0	41	9	< 0.1	7.0	0.0	36	9	0.1	7.0	0.0	19
ch150_1	20	149	360	0.2	0.5	4	11.0	0.0	589	15	0.2	11.0	0.0	480	15	0.4	11.2	0.4	881
ch150_2	20	149	402	0.2	0.5	4	12.0	0.0	342	17	0.2	12.0	0.0	338	15	0.4	12.0	0.0	387
ch150_3	20	149	357	0.2	0.5	4	11.0	0.0	655	14	0.2	11.0	0.0	520	13	0.4	11.0	0.0	808

Table 5.2: Results of the DCH and IDR for the districting part.

5.3 Efficient Consideration of Soft Time Windows in a Large Neighborhood Search

The approach described in Section 5.2 considers time windows in a strict sense. In practice, however, small time window violations typically do not matter much, and a larger flexibility in respect to them often allows substantially better solutions. In this section, we consider the *Districting and Routing Problem for Security Control with Soft Time Windows* (DRPSC-STW). Soft time windows may be violated to some degree, and their violation is considered in the objective function by penalty terms. In this context, the subproblem of determining optimal visiting times for a given candidate tour so that the total time window penalty is minimized arises. We call this problem *Optimal Arrival Time Problem* (OATP).

To classify the DRPSC-STW in context of the vehicle routing literature, one can see it as a *periodic vehicle routing problem with soft time windows* with additional constraints concerning separation time and maximum tour duration, where objects may have to be visited multiple times in each period. Separation-time constraints are a minimum time difference between two consecutive visits of the same object in a tour. Moreover, each tour for every district and period must not exceed a given maximum tour duration.

In this work, we primarily focus on the OATP and how it can be effectively solved. To this end we propose an approach based on *linear programming* (LP) and a faster heuristic approach using greedy techniques and dynamic programming. These mechanisms are embedded in a large neighborhood search (LNS) [113].

5.3.1 Related Work

We put our attention here on previous work dealing with soft time window. Although much more work is done on problem types with hard time windows, there already exists a significant number of works which introduce efficient methods to effectively handle soft time window constraints.

Ibaraki et al. [75] proposed a dynamic programming (DP) based approach to determine optimal starting times in conjunction with soft time windows which is applicable to a wider range of routing and scheduling applications. The total penalty incurred by time window violations is minimized. Compared to our approach they consider more general piecewise-linear penalty functions. Unfortunately, their approach is not directly applicable in our context as we have to additionally consider minimum separation times between visits of the same objects (i.e., objects can only be visited again if a minimum separation time between two consecutive visits is considered) and a maximum tour length. However, we will show later how this efficient DP method can nevertheless be utilized to some degree in our case.

Hashimoto et al. [64] extended the work of Ibaraki et al. to also consider flexible traveling times, which are also penalized if violated. They show, however, that the problem becomes NP-hard in case.

Taillard et al. [147] solve the vehicle routing problem with soft time windows by using tabu search. They do not consider any penalties for arriving too early but introduce a “lateness penalty” into the objective function. This penalty value is weighted by a given factor and the problem can be transformed into the vehicle routing problem with hard time windows by setting the weight factor to ∞.

Another work which shows the efficiency of applying DP for solving problems with soft time windows is by Ioachim et al. [76]. They solve the shortest path problem with time windows and linear node costs, where the linear node costs correspond to the modeling of soft time windows.

Fagerholt [48] published an approach for *ship scheduling with soft time windows*. He argues that by considering soft time windows, solution quality can be drastically improved and in practice small time window violations do not really matter. As in our work, a maximum allowed time window violation is used and earlier and later service is penalized. The approach can handle also non-convex penalty functions whereas in the literature most often only convex penalty functions are considered. The proposed solution approach uses a discretized time network in which nodes are duplicated according to possible start/arrival times. On the obtained shortest path network problem DP is applied for obtaining optimal arrival times.

To summarize related work, DP can frequently be an effective tool to determine optimal arrival/service times when considering soft time windows. Certain specificities of problems like maximal total tour duration and other constraints, however, frequently become an obstacle and prohibit the direct application of an efficient DP as the subproblem of determining optimal arrival times becomes NP hard. Nevertheless, DP may still be an important ingredient to deal with such situations in practice.

5.3.2 Problem Definition

In the DRPSC-STW we are given a set of objects $I = \{1, \dots, n\}$ and a depot 0, which is the start and end of each route. Travel times among the objects and the depot are given by $t^{\text{travel}}_{i,i'} > 0$ for $i, i' \in I \cup \{0\}$. We assume the triangle inequality to hold among these travel times. For every object $i \in I$ we are given a (small) number of visits $S_i = \{i_1, \dots, i_{|S_i|}\}$, and we are given a set of periods $P = \{1, \dots, p\}$. As not all visits have to take place in every period, subsets $W_{i,j} \subset S$ contain the visits of object i requested in period j for all $i \in I,\ j \in P$. The depot is visited two times, namely at the start of the tour and at the end of the tour. To ease modeling we define 0_0 to be the departure from the depot at the beginning and 0_1 to be the arrival at the depot at the end.

Each visit $i_k \in S_i, i \in I$ is associated with a visit time $t^{\text{visit}}_{i_k}$ and a particular time window $T_{i_k} = [T^{\text{e}}_{i_k}, T^{\text{l}}_{i_k}]$ in which the whole visit should preferably take place, already including its visit time. Visits $i_k \in S_i$ of an object $i \in I$ have to be visited in the given order, i.e., visit k has to be performed before visit k' iff $k < k'$.

Time windows of the visits are now softened such that an earlier start or later finishing of the service at an object is allowed. The maximum allowed earliness and lateness are, however, restricted by Δ, yielding the hard time windows $T^{\text{h}}_{i_k} = [T^{\text{he}}_{i_k}, T^{\text{hl}}_{i_k}] = [T^{\text{e}}_{i_k} - \Delta, T^{\text{l}}_{i_k} + \Delta]$, which must not be violated in any feasible solution.

An additional important requirement in the context of our security application is that any two successive visits $i_k, i_{k+1} \in W_{i,j}$ of the same object $i \in I$ must be separated by a *minimum separation time* t^{sep}. Obviously, visiting an object twice without a significant time inbetween would not make much sense. The maximum duration of any tour is given by $t^{\max}$.

Solutions to the DRPSC-STW are given by a tuple (D, τ, a) where $D = \{D_1, \dots, D_m\}$ is the partitioning of objects into districts, $\tau = (\tau_{r,j})_{r=1,\dots,m,\ j \in P}$ are the routes for each district and period, and a denotes the respective arrival times. Each tour $\tau_{r,j} = (\tau_{r,j,0}, \dots, \tau_{r,j,l_{r,j}+1})$ with $l_{r,j} = \sum_{i \in D_r} |W_{i,j}|$ starts and ends at the depot, i.e., $\tau_{r,j,0} = 0_0$ and $\tau_{r,j,l_{r,j}+1} = 0_1$, $\forall r = 1, \dots, m, j \in P$, and performs each visit in the respective ordering of the sequence. Each visit of a tour $\tau_{r,j,u}$ has to be associated with a specific arrival time $a_{r,j,u}$ and thus, $a = (a_{r,j,u})_{r=1,\dots,m,\ j=1,\dots,p,\ u=1,\dots l_{r,j}+1}$. An object always is immediately serviced after arrival but waiting is possible before leaving the object. A tour is feasible, if all visit, travel, and separation times are considered, each visit is performed at least within its hard time window and the total tour duration does not exceed $t^{\max}$.

While in our previous work [116] the primary objective was to minimize the number of districts (m), we consider this number now as pre-specified. For example, it can be obtained in a first optimization round by our previous method based on the hard time windows only. Now, in the DRPSC-STW, our objective is to minimize the total penalty incurred by all time window violations, which is

$$\min \sum_{r=1}^{m} \sum_{j \in P} \sum_{u=1}^{l_{r,j}} \omega_{r,j,u} \tag{5.21}$$

with

$$\omega_{r,j,u} = \begin{cases} T^{\mathrm{e}}_{i_k} - a_{r,j,u} & \text{if } a_{r,j,u} < T^{\mathrm{e}}_{i_k} \\ a_{r,j,u} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} & \text{if } a_{r,j,u} + t^{\mathrm{visit}}_{i_k} > T^{\mathrm{l}}_{i_k} \\ 0 & \text{otherwise} \end{cases} \tag{5.22}$$

5.3.3 Optimal Arrival Time Problem

When approaching the DRPSC-STW with an LNS in Section 5.3.5, we will have to solve for each tour in each period of each candidate solution the following subproblem: Given a candidate tour $\tau_{r,j} = (\tau_{r,j,0}, \ldots, \tau_{r,j,l_{r,j}+1})$ for some district $r = 1, \ldots, m$ and period $j = 1, \ldots, p$, what are feasible arrival times $a_{r,j,u}$ for the visits $u = 1, \ldots, l_{r,j} + 1$ minimizing $\sum_{u=1}^{l_{r,j}} \omega_{r,j,u}$? Remember that the solution must obey the minimum separation time t^{sep} between any two successive visits of the same object and the maximum tour duration $t^{\max}$. We call this subproblem *Optimal Arrival Time Problem* (OATP).

As we consider in the OATP always only one specific tour $\tau_{r,j}$, i.e., r and j are known and constant, we omit these indices in the following for simplicity wherever this is unambiguous. In particular, we write τ for the current tour, l for the tour's length, τ_h for the h-th visit, a_h for the respective arrival time, and ω_h for the respective time window penalty. Moreover, we introduce some further notations and definitions used in the next sections. Let us more generally define the time window penalty function $\rho_h(t)$ for visit $\tau_h = i_k$ when arriving at time t as the following piecewise linear function, see also Figure 5.2:

$$\rho_h(t) = \begin{cases} \infty & \text{if } t < T^{\mathrm{e}}_{i_k} - \Delta \\ T^{\mathrm{e}}_{i_k} - t & \text{if } T^{\mathrm{e}}_{i_k} - \Delta \le t < T^{\mathrm{e}}_{i_k} \\ t + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} & \text{if } T^{\mathrm{l}}_{i_k} < t + t^{\mathrm{visit}}_{i_k} \le T^{\mathrm{l}}_{i_k} + \Delta \\ \infty & \text{if } t > T^{\mathrm{l}}_{i_k} + \Delta \\ 0 & \text{otherwise.} \end{cases}$$

Let $V = \{i_k \mid i \in D_r,\ i_k \in W_{i,j}\}$ be the set of all object visits in the current tour. We define the auxiliary function $\kappa : V \mapsto D_r$ mapping visit $i_k \in V$ to its corresponding object $i \in D_r$, and function $\sigma(h)$ which finds the nearest successor index h' of the visit $\tau_{h'}$ with $h < h' \le l$ and $\kappa(\tau_h) = \kappa(\tau_{h'})$ if such a successive visit of the same object exists; otherwise

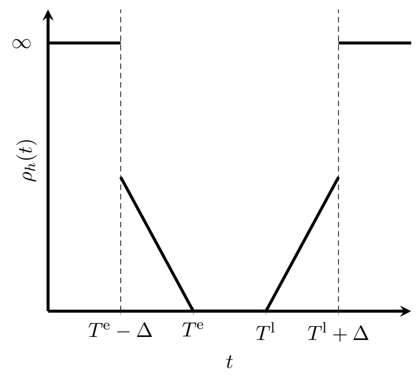


Figure 5.2: The time window penalty function $\rho_h(t)$.

$\sigma(h)$ returns -1. Correspondingly, function $\sigma^{-1}(h)$ returns the nearest predecessor index h' of the visit $\tau_{h'}$ with $1 \leq h' < h$ and $\kappa(\tau_h) = \kappa(\tau_{h'})$ if such a predecessor exists and -1 otherwise. For convenience, we also define $\zeta_h = t^{\text{visit}}_{\tau_{r,j,h}} + t^{\text{travel}}_{\kappa(\tau_{r,j,h}),\kappa(\tau_{r,j,h+1})}$ as the sum of the visiting time of the hth visit and the travel time from the hth visit to the $(h+1)$st visit.

Lower and Upper Bounds for Arrival Times

We compute lower and upper bounds for each visit's arrival time by determining routes in which we perform each visit as early as possible and as late as possible. For determining the earliest arrival time at the first visit we have to consider the maximum of the travel time from the depot to the first visit and the earliest possible time of the first visit's hard time window. The earliest possible arrival time for all other visits $h = 1, \ldots, l$ can be computed recursively by considering the dependency on the previous visit $h-1$, i.e., the visit time and travel time to the current visit h, the beginning of the hard time window $T^{\text{e}}_{\tau_h} - \Delta$ of the current visit h, and the separation time from a possibly existing previous visit of the same object $\sigma^{-1}(h)$ in the tour. This yields:

$$a_h^{\text{earliest}} = \begin{cases} -\infty & \text{if } h < 0 \\ T^{\text{e}}_{0_0} & \text{if } h = 0 \\ \max\left\{a^{\text{earliest}}_{h-1} + t^{\text{visit}}_{\tau_{h-1}} + t^{\text{travel}}_{\tau_{h-1},\tau_h}, T^{\text{e}}_{\tau_h} - \Delta, a^{\text{earliest}}_{\sigma^{-1}(h)} + t^{\text{sep}}\right\} & \text{if } h > 0 \end{cases}$$

When scheduling a latest tour the last visit of the tour has to be scheduled before arriving at the depot where also the travel time to the depot has to be considered, but on the other hand we have to also consider the end of the hard time window $T^{\text{l}}_{\tau_l} + \Delta$ of the last visit. For all other visits we can compute their latest possible arrival time by considering the next visit's arrival time, the travel time to the next visit, and the visit time at the current visit, the end of the hard time window of the current visit, i.e., $T^{\text{l}}_{\tau_l} + \Delta$, and the separation time by considering a possibly existing successive visit $\sigma(h)$ of the same object where $\kappa(\tau_h) = \kappa(\tau_{h'})$ with $h < h'$:

$$a_h^{\text{latest}} = \begin{cases} \infty & \text{if } h < 0 \\ T^{\text{l}}_{0_1} & \text{if } h = l+1 \\ \min\left\{a_{h+1}^{\text{latest}} - t^{\text{visit}}_{\tau_h} - t^{\text{travel}}_{\tau_h,\tau_{h+1}}, T^{\text{l}}_{\tau_h} + \Delta, a^{\text{latest}}_{\sigma(h)} - t^{\text{sep}}\right\} & \text{if } 0 \le h \le l \end{cases}$$

If for some h, $a_h^{\text{earliest}} > a_h^{\text{latest}}$, we immediately terminate as this OATP instance, i.e., underlying route, cannot have a feasible solution.

Linear Programming Model

The OATP is not an NP-hard optimization problem. We can solve it exactly by means of linear programming (LP) as we show in the following.

Variables a_{i_k} represent the arrival time of the k-th visit of object i, variables $p^{\text{e}}_{i_k}$ are used to compute the penalty when starting the service of visit i_k too early, and variables $p^{\text{l}}_{i_k}$ are used for the penalty when finishing the service of visit i_k too late. The LP is defined as follows:

$$\begin{aligned}
\min \quad & \sum_{i_k \in V} p^{\text{e}}_{i_k} + p^{\text{l}}_{i_k} && && (5.23)\\
\text{s.t.} \quad & t^{\text{travel}}_{0,\kappa(a_{\tau_1})} + a_{\tau_l} + t^{\text{visit}}_{\tau_l} + t^{\text{travel}}_{\kappa(\tau_l),0} - a_{\tau_1} + \le t^{\max} && && (5.24)\\
& a_{\tau_1} \ge t^{\text{travel}}_{0,a_{\tau_1}} + T^{\text{e}}_{0_0} && && (5.25)\\
& a_{\tau_l} + t^{\text{visit}}_{\tau_{a_l}} + t^{\text{travel}}_{\kappa(\tau_l),0} \le T^{\text{l}}_{0_1} && && (5.26)\\
& a_{\tau_i} \ge a_{\tau_{i-1}} + t^{\text{visit}}_{a_{\tau_{i-1}}} + t^{\text{travel}}_{\kappa(\tau_{i-1}),\kappa(\tau_i)} && \forall \tau_i \in \tau,\ i = 2,\ldots,l && (5.27)\\
& a_{i_k} \ge a_{i_{k'}} + t^{\text{visit}}_{i_{k'}} + t^{\text{sep}} && \forall i_k, i_{k'} \in V,\ k > k' && (5.28)\\
& T^{\text{e}}_{i_k} - \Delta \le a_{i_k} \le T^{\text{l}}_{i_k} + \Delta - t^{\text{visit}}_{i_k} && \forall i_k \in V && (5.29)\\
& p^{\text{e}}_{i_k} \ge T^{\text{e}}_{i_k} - a_{i_k} && \forall i_k \in V && (5.30)\\
& p^{\text{l}}_{i_k} \ge a_{i_k} + t^{\text{visit}}_{i_k} - T^{\text{l}}_{i_k} && \forall i_k \in V && (5.31)\\
& a_{i_k}, p^{\text{e}}_{i_k}, p^{\text{l}}_{i_k} \ge 0 && \forall i_k \in V && (5.32)
\end{aligned}$$

Objective function (5.23) minimizes the total penalty incurred by too late or too early arrival times of visits. Inequality (5.24) ensures that the makespan of the tour does not exceed the maximum allowed duration $t^{\max}$. Otherwise, the given visit order would be infeasible. Inequality (5.25) models the travel time from the depot to the first visit of the given order, i.e., the first visit can only be started after traveling from the depot to this visit. Inequality (5.26) specifies that the tour has to end latest at the end of the time window of the second visit of the depot. Inequalities (5.27) ensure that all travel times between consecutive object visits and visit times are respected. Inequalities (5.28) guarantee the minimum separation time between two consecutive visits of the same

Algorithm 5.3: Hybrid Heuristic for OATP

```
1: Input: Tour $\tau$
2: if not Feasible($\tau$) then
3:     return $\infty$
4: end if
5: if GreedyHeuristic($\tau$) $= 0$ then
6:     return 0
7: end if
8: return DPBasedHeuristic($\tau$)
```

object. Inequalities (5.29) ensure consideration of the hard time windows. The penalty values are computed by inequalities (5.30) and (5.31). If a visit is scheduled too early, then $T^{\mathrm{e}}_{i_k} - a_{i_k} > 0$ and an early penalty is incurred. Obviously, if the earliness penalty $p^{\mathrm{e}}_{i_k} > 0$, then $a_{i_k} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} < 0$ and thus, $p^{\mathrm{l}}_{i_k} = 0$. This holds vice versa if the lateness penalty $p^{\mathrm{l}}_{i_k} > 0$. If a visit is scheduled within its time window $[T^{\mathrm{e}}_{i_k}, T^{\mathrm{l}}_{i_k}]$, then $p^{\mathrm{e}}_{i_k} = p^{\mathrm{l}}_{i_k} = 0$ as $T^{\mathrm{e}}_{i_k} - a_{i_k} \leq 0$ and $a_{i_k} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} \leq 0$ and $p^{\mathrm{e}}_{i_k}, p^{\mathrm{l}}_{i_k} \geq 0$, $\forall i_k \in V$ according to equations (5.32).

5.3.4 Hybrid Heuristic for the OATP

While the above LP model can be solved in polynomial time, doing this many thousands of times within a metaheuristic for the DRPSC-STW for evaluating any new tour in any period of any district in any candidate solution is still a substantial bottleneck. We therefore consider a typically much faster heuristic approach in the following, which, as our experiments will show, still yields almost optimal solutions. We call this approach *Hybrid Heuristic* (HH) for the OATP as it is, in fact, a sequential combination of different individual components.

The overall approach is shown in Algorithm 5.3, and the individual components are described in detail in the subsequent sections. First, we show how to efficiently check the feasibility of a given instance (line 2), then, we apply a fast greedy heuristic which tries to solve the problem without penalties (line 5) using an earliest possible start time strategy. Finally, we apply an efficient DP-based heuristic to obtain a solution for the OATP.

Feasibility Check

The feasibility of a given tour, i.e., existence of feasible arrival times, can be efficiently checked by calculating the minimum tour duration and comparing it to $t^{\max}$. The minimum tour duration can be determined by fixing the arrival time at the depot to

$a_{l+1}^{\text{earliest}}$ and calculating the latest arrival times recursively backwards:

$$a_h^{\text{ms}} = \begin{cases} \infty & \text{if } h < 0 \\ a_h^{\text{earliest}} & \text{if } h = l+1 \\ \min\left\{a_{h+1}^{\text{ms}} - t_{\tau_h}^{\text{visit}} - t_{\tau_h,\tau_{h+1}}^{\text{travel}}, T_{\tau_h}^{\text{l}} + \Delta, a_{\sigma(h)}^{\text{ms}} - t^{\text{sep}}\right\} & \text{if } 0 \le h \le l \end{cases}$$

The tour is feasible iff $a_{l+1}^{\text{ms}} - a_0^{\text{ms}} \le t^{\max}$ holds.

Greedy Heuristic

A fast heuristic for solving the OATP is a greedy strategy that starts each visit as early as possible without violating any soft time window. If this heuristic is successful, no penalty occurs and the obtained solution is optimal. We can formulate this approach as follows:

$$a_h^{\text{greedy}} = \begin{cases} -\infty & \text{if } h < 0 \\ \max\left\{T_{0_0}^{\text{e}} + t_{0,\kappa(\tau_1)}^{\text{travel}}, T_{\tau_h}^{\text{e}}\right\} & \text{if } h = 1 \\ \max\left\{a_{h-1}^{\text{greedy}} + t_{\tau_{h-1}}^{\text{visit}} + t_{\tau_{h-1},\tau_h}^{\text{travel}}, T_{\tau_h}^{\text{e}}, a_{\sigma^{-1}(h)}^{\text{greedy}} + t^{\text{sep}}\right\} & \text{if } h > 1 \end{cases}$$

If for some h, $a_h^{\text{greedy}} > a_h^{\text{latest}}$, then the greedy heuristic cannot solve this problem instance and terminates.

Efficiently Solving a Relaxation by Dynamic Programming

The greedy strategy is fast, works reasonably well, and frequently yields an optimal solution for easy instances. When the constraints become tighter, however, it often fails. Therefore, we finally use a second, more sophisticated heuristic based on the following considerations.

The required minimum separation times for visits of same objects make the OATP, in contrast to other problems aiming at finding arrival times introducing a minimum penalty, e.g. [75], inaccessible for an efficient exact DP approach. One would need to somehow consider also all objects' last visits when storing and reusing subproblem solutions in the DP recursion.

However, in a heuristic approach we can exploit an efficient DP for the relaxed variant of the OATP in which we remove the separation time constraints. We denote this relaxed OATP by OATP$^{\text{rel}}$. As will be shown in Section 5.3.4, we will modify our instance data before applying this DP in order to obtain a heuristic solution that is feasible for our original OATP.

To solve OATP$^{\text{rel}}$ we apply DP inspired by Ibaraki et al. [75]. In contrast to this former work, however, we consider a maximum tour duration.

Let $g_h(t, t_0)$ be the minimum sum of the penalty values for visits $\tau_0, \ldots, \tau_h$ under the condition that all of them are started no later than at time t and the depot is left earliest at time t_0 with $t - t_0 \leq t^{\max}$. Here we assume that $T^{\mathrm{e}}_{0_0} \leq t_0$. Then, $g_h(t, t_0)$ can be expressed recursively by:

$$g_0(t, t_0) = \begin{cases} \infty & \text{if } t < t_0 \\ 0 & \text{otherwise} \end{cases}$$
$$g_h(t, t_0) = \min_{a_h^{\text{earliest}} \leq t' \leq \min\{t, t_0 + t^{\max}\}} g_{h-1}(t' - \zeta_{h-1}, t_0) + \rho_h(t') \quad \text{if } h > 0$$

Here, we assume the minimum of an empty set or interval to be ∞. The overall minimum time penalty of the tour τ is then $\min_{a_0^{\text{earliest}} \leq t_0 \leq a_0^{\text{latest}}} g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$. Thus, solving OATP$^{\text{rel}}$ corresponds to finding a departure time t_0 from the depot which minimizes function $f^{\text{rel}} = g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$.

Let t_0 be the value for which $f^{\text{rel}} = g_h(T^{\mathrm{l}}_{0_1}, t_0)$ yields a minimum penalty. Optimal arrival times for the visits and the arrival time back at the depot can now be expressed by:

$$\begin{aligned} a_{l+1}^{\text{rel}} &= \operatorname*{arg\,min}_{T^{\mathrm{e}}_{0_0} \leq t \leq T^{\mathrm{l}}_{0_1}} g_{l+1}(t, t_0) \\ a_h^{\text{rel}} &= \operatorname*{arg\,min}_{T^{\mathrm{e}}_{0_0} \leq t \leq a^{\text{rel}}_{h+1} - \zeta_h} g_h(t, t_0) \quad \text{if } 0 \leq h \leq l \end{aligned} \tag{5.33}$$

Now, let us consider the task of efficiently computing $g_h(t, t_0)$ in more detail. Recall that our time window penalty function $\rho_h(t)$ is piecewise linear for all visits $\tau_1, \dots, \tau_l$ and they have all the same shape as shown in Figure 5.2. Therefore, $g_h(t, t_0)$ is also piecewise linear. We store these piecewise linear functions of each recursion step of the DP algorithm in linked lists, whose components represent the intervals and the associated linear functions.

An upper bound for the total number of pieces in the penalty functions for all the visits $\tau_0, \dots, \tau_{l+1}$ is $5l + 2 = O(l)$. The computation of $g_{h-1}(t - \zeta_{h-1}, t_0) + \rho_h(t)$ and $g_h(t, t_0)$ from $g_{h-1}(t, t_0)$ and $\rho_h(t)$ can be achieved in $O(h)$ time, since the total number of pieces in $g_{h-1}(t, t_0)$ and $\rho_h(t)$ is $O(h)$. In order to calculate the function $g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$ for a given tour, we compute $g_h(t, t_0)$ for all $1 \leq h \leq l + 1$. This can be done in $O(l^2)$ time.

Now that we know how to efficiently calculate the minimum time window penalty value for a given departure time from the depot t_0, we draw our attention to the problem of finding a best departure time such that the overall penalty value for a given tour is minimized. Formally, we want to minimize function $g'(t_0) = g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$ on interval $t_0 \in [a_0^{\text{earliest}}, a_0^{\text{latest}}]$. Enumerating all possible t_0 values is obviously not a reasonable way to tackle this problem. Fortunately, there is a useful property of function $g'(t_0)$ which enables us to search more efficiently for its minimum.

Proposition 5. *Earliest optimal arrival times can only be delayed further when the depot departure time increases. More formally, let a_h^0 for $h = 0, \dots, l+1$ be earliest optimal arrival times calculated by $g'(t_0)$ for some t_0 and a_h^1 for $h = 0, \dots, l+1$ be the earliest optimal arrival times calculated by $g'(t'_0)$ for some t'_0. Then $t_0 \leq t'_0 \implies a_h^0 \leq a_h^1$ for $h = 0, \dots, l+1$.*

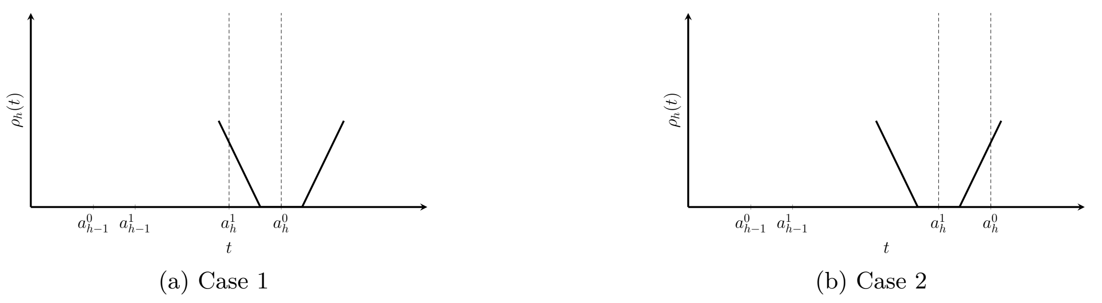


Figure 5.3: Visualization of the two case distinctions used in the proof of Proposition 5.

Proof. We show this with a proof by contradiction. Without loss of generality, suppose there is a visit h with $a^0_{h-1} \leq a^1_{h-1}$ and $a^1_h < a^0_h$. Let us consider two relevant cases in detail. Other cases can be refuted using similar arguments.

Case 1: Assume a^1_h is scheduled earlier than a^0_h and $a^1_h < T^{\mathrm{e}}_{\tau_h}$, see Figure 5.3a. a^1_h could only have been scheduled earlier than a^0_h falling below $T^{\mathrm{e}}_{\tau_h}$ threshold if and only if one of its subsequent visits $\tau_{h+1}, \ldots, \tau_{l+1}$ was forced to start earlier. This can only happen if the arrival time constraint, where we have to be back at the depot, is more tightened. But this clearly cannot be the case here, since $t_0 + t^{\max} \leq t'_0 + t^{\max}$. In other words, delaying the departure time at the depot also delays the arrival time constraint, when we have to be back at the depot.

Case 2: Assume a^1_h is scheduled earlier than a^0_h and $a^0_h > T^{\mathrm{l}}_{\tau_h} - t^{\mathrm{visit}}_{\tau_h}$, see Figure 5.3b. Since $a^0_h - a^0_{h-1} > a^1_h - a^1_{h-1}$, it is easy to see that a^0_h can be moved further to the left without introducing more penalty. Therefore, a^0_h cannot be the optimal start time for the visit h, since the T^{l}_h constraint violation caused by a^0_h can be reduced further.

□

Proposition 6. $\forall t'_0, t''_0 \mid g'(t'_0) < g'(t''_0), t'_0 < t''_0 \implies \forall t_0 \geq t''_0 : g'(t''_0) < g'(t_0)$.

Proof. Let $a^{\mathrm{earliest}'}_h$ for $h = 0, \ldots, l+1$ be the earliest possible arrival times when fixing t'_0 as the departure time from the depot and $a^{\mathrm{earliest}''}_h$ for $h = 0, \ldots, l+1$ the earliest possible arrival times when fixing t''_0 as the departure time from the depot. Furthermore, we define a''_h for $h = 0, \ldots, l+1$ to be earliest optimal arrival times calculated by $g'(t''_0)$.

We have shown that the earliest optimal arrival times can only be delayed further when postponing the departure time from the depot. Thus, the only way the overall penalty value can be increased is when pushing t_0 to the future causes more T^{l} threshold violations than what you can save by reducing T^{e} threshold violations.

More formally, if we have $g'(t_0') < g'(t_0'')$ with $t_0' < t_0''$, then there must exist $a_k^{\text{earliest}''} > T^l_{\tau_k} - t^{\text{visit}}_{\tau_k}$ with $a_k^{\text{earliest}'} < a_k^{\text{earliest}''}$ and $a_k^{\text{earliest}''} = a_k''$ for some $k \in \{0, \dots, l+1\}$. In other words, if the overall penalty value increases, then there are visits whose earliest possible arrival times are pushed furhter to the future exceeding T^{l} thresholds by t_0'' and their optimal arrival times are equal to earliest possible arrival times.

It is easy to see that once the earliest possible start time a_h^{earliest} starts to increase, it continues to increase strictly monotonically with an increasing departure time from the depot. Therefore, the overall penalties will increase strictly monotonically from t_0'' on with an increasing departure time from the depot until the solution becomes infeasible. □

These properties show that $g'(t_0)$ is in general a "U-shaped" function when disregarding all infeasible solutions yielding ∞, and we can use a bisection method to search efficiently for a minimum. The calculation of f^{rel} in this way is shown in Algorithm 5.4.

At each iteration step the middle point t of current search interval is sampled and we calculate an approximate subgradient $\nabla g'(t)$ of g' at t by $\nabla g'(t) = g'(t+\delta) - g'(t)$ where δ is a small constant value. If the subgradient $\nabla g'(t) > 0$, we know that t is in the strictly monotonically rising piece of g' and we continue our search in the left half. Otherwise the search continues in the right half. The bisection method proceeds until the search interval becomes smaller than some predetermined value ε.

DP-Based Heuristic for OATP

Obviously, OATP$^{\text{rel}}$ corresponds to the original OATP if there are no objects that are visited multiple times or $\sum_{i=h}^{\sigma(h)-1} \zeta_i \geq t^{\text{sep}}$ for $h = 1, \dots, l$ with $\sigma(h) \neq -1$. The main idea of our second heuristic is to increase the ζ_i values as necessary so that $\sum_{i=h}^{\sigma(h)-1} \zeta_i \geq t^{\text{sep}}$ holds for all $h = 1, \dots, l$ with $\sigma(h) \neq -1$. Then, when applying the DP, its solution will obviously fulfill the separation-time constraint.

Let visits τ_k and $\tau_{k'}$ with $k < k'$ and $\sum_{i=k}^{k'-1} \zeta_i < t^{\text{sep}}$ be two visits which take place at the same object. Then, one or more $\zeta_i \in \{\zeta_k, \dots, \zeta_{k'-1}\}$ must be extended so that $\sum_{i=k}^{l-1} \zeta_i = t^{\text{sep}}$. In order to decide which ζ_i we want to extend, we first calculate waiting times for all visits with earliest possible arrival times.

The waiting time at the visit τ_h is the amount of time we are forced to wait at the visit τ_{h-1} before we can travel to visit τ_h. Recall that we are forced to wait at visit τ_{h-1} if $a_{h-1} + \zeta_{h-1} < T^{\text{e}}_{\tau_h}$. Thus, the waiting times with earliest possible arrival times can be expressed as $w_h^{\text{earliest}} = \max\left\{0, a_h^{\text{earliest}} - a_{h-1}^{\text{earliest}} - \zeta_{h-1}\right\}$, $h = 1, \dots, l$. Using these waiting times as guidance, we extend the ζ_i value at the visit τ_i with the maximum waiting time $w_i^{\text{earliest}} = \max\left\{w_k^{\text{earliest}}, \dots, w_{l-1}^{\text{earliest}}\right\}$ where ties are broken randomly. The rationale behind this idea is that large w_h^{earliest} values often indicate the visits in an optimal solution, where extra waiting time actually is introduced to satisfy the separation-time constraints.

Algorithm 5.4: Calculation of f^{rel}

```
**Require:** $a_0^{\text{earliest}}$, $a_0^{\text{latest}}$
 1: **init:** $a \leftarrow a_0^{\text{earliest}}$, $b \leftarrow a_0^{\text{latest}}$, $v_1 \leftarrow f^{\text{rel}} \leftarrow g'(a)$
 2: **if** $v_1 = 0$ **or** $v_1 = \infty$ **or** $\nabla g'(t) > 0$ **then**
 3:   **return** $v_1$
 4: **end if**
 5: **while** $b - a > \varepsilon$ **do**
 6:   $t \leftarrow a + \frac{b-a}{2}$
 7:   $v_2 \leftarrow g'(t)$
 8:   **if** $v_2 < f^{\text{rel}}$ **then**
 9:     $f^{\text{rel}} \leftarrow v_2$
10:   **end if**
11:   **if** $f^{\text{rel}} = 0$ **or** $v_1 = v_2$ **then**
12:     **break**
13:   **end if**
14:   **if** $v_2 = \infty$ **or** $\nabla g'(t) \leq 0$ **then**
15:     $a \leftarrow t$
16:   **else**
17:     $b \leftarrow t$
18:   **end if**
19:   $v_1 \leftarrow v_2$
20: **end while**
21: **return** $f^{\text{rel}}$
```

Utilizing waiting times computed by earliest possible arrival times works well for the majority of instances but for some instances the ζ_h values are altered unfavorably so that the instances become infeasible. To counteract this problem, we propose alternative waiting times which are calculated using arrival times with minimum tour duration: $w_h^{\text{ms}} = \max\left\{0, a_h^{\text{ms}} - a_{h-1}^{\text{ms}} - \zeta_{h-1}\right\}$, $\forall h = 1, \ldots, l$. Visits with waiting times larger than 0 indicate visits in the tour with minimum tour duration for which additional waiting time had to be introduced in order to satisfy separation-time constraints. Using w_h^{ms} waiting times we can effectively complement situations where the approach utilizing w_h^{earliest} values yields infeasible or low-quality solutions. Therefore, we solve the DP-based heuristic twice, using both w_h^{earliest} and w_h^{ms} and take the best solution.

Even if the solution of this DP-based heuristic does not guarantee optimality in general, it works well in practice, producing near optimal solutions in significantly shorter computation times than the exact LP approach.

5.3.5 Large Neighborhood Search for the DRPSC-STW

Our overall approach for solving the DRPSC-STW follows the classical large neighborhood search metaheuristic [113] with an embedded variable neighborhood descent (VND) for local improvement.

We define our *destroy* and *repair* methods as follows. In order to destroy a current solution candidate, we select two out of m districts uniformly at random and remove all objects from these districts. The removed objects are copied to a so called *ejection pool.* Then, we apply the repair phase of the *district elimination algorithm* proposed by Prischink et al. [117]. The algorithm continues until all objects in the ejection pool are reassigned to the available districts. Using this *destroy* and *repair* methods, we guarantee that the solution stays feasible with the same number of districts. At each LNS iteration a VND is applied to locally improve the incumbent solution.

Variable Neighborhood Descent

We use three common neighborhood structures from the literature and search in a best improvement fashion. We apply these neighborhoods in the given order since we could not identify any significant advantages using different orderings. Infeasible solutions are discarded.

2-opt: Classical 2-opt neighborhood where all edge exchanges are considered.

swap: Exchanges all pairs of distinct visits within a route.

or-opt: Moves sequences of one to three consecutive visits at another place in the route.

The proposed VND is performed separately for each route of every district. Our local improvement component could also be very well parallelized since different routes can be optimized independently of each other, however this is not in the scope of this work. Since routes having no penalties are already optimal, they are excluded from local improvement.

5.3.6 Computational Results

For the computational results, we used the instances which have been created by Prischink et al. [117]. In a first optimization round, we solve the districting part of the DRPSC-STW by means of the district elimination algorithm proposed by Prischink et al., based on the hard time windows only, generating input[2] for the subsequent time window penalty minimization round with the LNS algorithm. As global parameters we have chosen $t^{\max}$ to be 12 hours and the maximum allowed penalty $\Delta = 60$ minutes, which represent typical values used in practical settings. Furthermore, we set HH (Algorithm 5.3) specific parameters $\delta = 1$ and $\varepsilon = 30$, which have been determined empirically. For our test instances, they give good balance between computational speed and accuracy. Every

[2] `https://www.ac.tuwien.ac.at/files/resources/instances/drpsc/evoc17.tgz`

Instance							**LNS-LP**				**LNS-HH**			
name	runs	$\|I\|$	$\|V\|$	α	β	v	#best	$\overline{\text{obj}}$ [s]	$\bar{t}$ [s]	#eval	#best	$\overline{\text{obj}}$ [s]	$\bar{t}$ [s]	#eval
berlin52_1	20	51	133	0.0	0.7	4	13	476.9	640.2	499,624.3	**20**	29.9	298.0	757,181.5
berlin52_2	20	51	130	0.0	0.7	4	13	235.1	662.5	525,862.1	**19**	7.9	239.0	586,096.8
berlin52_3	20	51	140	0.0	0.7	4	5	3,230.9	900.0	595,039.3	**15**	1,169.7	900.0	938,946.7
ch150_1	20	149	360	0.2	0.5	4	4	88,910.1	900.0	663,320.8	**16**	55,234.8	900.0	1,308,850.8
ch150_2	20	149	402	0.2	0.5	4	0	164,399.8	900.0	662,869.3	**20**	80,989.3	900.0	1,257,789.1
ch150_3	20	149	357	0.2	0.5	4	3	78,979.9	900.0	748,549.8	**17**	36,370.5	900.0	1,620,281.2
ft70_1	20	69	167	0.1	0.5	4	2	5,035.8	900.0	993,374.5	**18**	1,196.9	900.0	2,815,488.6
ft70_2	20	69	180	0.1	0.5	4	1	3,087.0	900.0	975,529.6	**20**	464.2	878.3	2,719,290.8
ft70_3	20	69	144	0.1	0.5	4	3	5,602.2	900.0	918,004.6	**17**	1,509.0	900.0	2,496,826.1
gr48_1	20	47	120	0.2	0.7	4	8	92,669.4	900.0	284,934.1	**12**	70,082.1	900.0	443,507.7
gr48_2	20	47	115	0.2	0.7	4	3	9,445.5	900.0	851,306.0	**17**	4,233.1	900.0	2,296,485.5
gr48_3	20	47	125	0.2	0.7	4	5	28,606.2	900.0	392,935.5	**15**	24,982.1	900.0	615,803.7
rd100_1	20	99	152	0.1	0.5	2	3	25,824.7	900.0	876,710.0	**17**	11,050.4	900.0	2,289,110.3
rd100_2	20	99	160	0.1	0.5	2	4	22,367.5	900.0	828,483.6	**16**	7,618.6	900.0	2,123,393.7
rd100_3	20	99	152	0.1	0.5	2	4	12,132.5	900.0	826,517.7	**17**	2,136.9	900.0	2,036,959.2
st70_1	20	69	105	0.1	0.7	2	5	15,052.1	900.0	755,594.8	**16**	4,380.1	900.0	1,761,210.0
st70_2	20	69	91	0.1	0.7	2	4	18,622.9	900.0	806,985.2	**16**	8,228.9	900.0	2,126,834.2
st70_3	20	69	106	0.1	0.7	2	3	7,022.6	900.0	696,001.3	**20**	380.5	673.1	1,140,718.4
tsp225_1	20	224	334	0.2	0.7	2	0	272,118.2	900.0	969,287.0	**20**	183,974.1	900.0	1,904,494.3
tsp225_2	20	224	341	0.2	0.7	2	0	340,426.5	900.0	692,597.6	**20**	293,867.6	900.0	1,375,567.5
tsp225_3	20	224	332	0.2	0.7	2	0	161,586.6	900.0	710,581.4	**20**	141,153.5	900.0	1,471,799.2
						Average	4.0	64,563.4	884.5	727,338.5	**17.5**	44,240.9	828.0	1,623,173.1

Table 5.3: Results of the LNS with embedded LP and HH as solution evaluation function

instance was given a maximum allowed time limit of 900 seconds for the execution of the LNS and we have performed 20 runs for every instance. All tests have been executed as single threads on an Intel Xeon E5540 2.53GHz Quad Core processor. The algorithms have been written in C++ and have been compiled with gcc-4.8 and for solving the LP we used Gurobi 7.0.

In Table 5.3 the results of the LNS-LP and LNS-HH can be found. In the instance column, we specify the instance parameters. Sequentially, the name of the used TSPlib instance (refer to Prischink et al. [117] for a more detailed description), the number of runs performed, the number of objects $|I|$, the total number of visits $|V|$, the percentage of large time windows (α), the percentage of mid-sized time windows (β) and the maximum number of allowed visits per object v is given. For the LNS-LP and LNS-HH the number of times the corresponding approach yields the best result, the average objective value over all runs of the instance, the average runtime, and the average number of objective function evaluations are given. Results show clearly that LNS-HH yields better objective values than LNS-LP since it is able to perform much more iterations within the given time limit due to fast objective function evaluations. It is also obvious that by increasing the instance size, the advantage of the efficient HH evaluation function is getting more pronounced. Moreover, a Wilcoxon signed-rank test shows that all observed differences on the overall number of best solutions among the LNS-LP and the LNS-HH are statistically significant with an error level of less than 1%.

We can conclude that LNS-HH is superior compared to LNS-LP due to significant performance advantage in the evaluation function, even though the HH-based evaluation function is only a heuristic method which in general does not yield proven optimal solutions although it can be observed that the optimality gap of HH is in most cases neglectably small.

5.4 Conclusions and Future Work

In this work we introduced a new vehicle routing problem which originates from the security control sector. The goal of the Districting and Routing Problem for Security Control is to partition a set of objects under surveillance into disjoint clusters such that for each period a route through all requested visits can be scheduled satisfying complex time window constraints. As the objects may require multiple visits, there needs to be a minimum separation time between each two visits which imposes an interesting additional challenge. The proposed heuristic solution approach starts with a greedy construction heuristic followed by an iterate destroy and recreate algorithm. The latter works by iteratively destroying districts and trying to insert the resulting unassigned objects into the other districts. The computational results reveal that the MIP model is able to solve smaller instances of the routing problem to optimality and that the quality of the initial solutions of the districting problem has only a minor influence on the final solution quality. There are several possibilities for extending this algorithm in future work. As the feasibility check for a district is time-consuming a caching mechanism to prevent checking the same assignment of objects all over again seems promising. This could even be extended to checking subsets of such assignments, which also must be feasible if any superset of these objects results in feasible routes. Another idea is to use neighborhood structures which exchange objects of two or more distinct clusters.

We also analyzed the DRPSC-STW where the DRPSC is extended by soft time windows. This problem is of high practical relevance as it is possible to significantly improve solution quality by introducing only a negelectable penalty.

As metaheuristic we propose an LNS for approaching the DRPSC-STW. A critical bottleneck of our LNS is the evaluation of solution candidates where one has to find the minimum penalty given a particular visit order. We show that this evaluation function can be efficiently implemented by an LP-based approach, and furthermore we developed a sophisticated hybrid heuristic which was able to drastically outperform the LP-based variant.

We have formulated an efficient method to determine optimal arrival times of a given visit order which can be embedded inside a metaheuristic framework to solve the penalty minimization part of the DRPSC-STW. On the one hand this is not only relevant for the DRPSC-STW, as soft time windows play in general an important role in many practical scenarios.

Future research goals include the extension of the current LNS by incorporating adaptiveness into the destroy and repair moves. Furthermore, the authors want to note that it is also possible to extend the VND local search into a VNS by including a shaking neighborhood like randomized k-swap neighborhood, c.f. [34]. This way, one can combine micro- and macro-diversifications during the search.

CHAPTER 6

Conclusions and Future Work

This work considered three different COPs with a strong background to real-world problems, two of them arising in bike-sharing systems and another vehicle routing problem from the domain of security control.

The BBSS problem was and still is a hot topic in combinatorial optimization as the number of PBSs is rising worldwide because cities want to benefit from its opportunities. As seen in the related work, there exists a large number of publications on BBSS, however many of them have different problem formulations or different optimization goals which makes a direct comparison hard to impossible.

For the static variant of BBSS, as it appears at Citybike Wien, we proposed efficient metaheuristics based on VNS which are able to provide good solutions in reasonable computation times. It has been one of the first metaheuristics solving large-scale instances up to 700 stations. Moreover, a time-indexed as well as a hop-indexed MIP model have been developed which, however, are only be able to solve small instances for the problem. For larger instances, also the developed GRASP yielded excellent results. A key aspect in our metaheuristics is the efficient calculation of loading instructions, for which we studied different approaches with different precision and runtimes. We compared four different approaches where three of them efficiently yielded optimal loading instructions based on maximum-flow formulations and linear programming as well as a greedy-based one outperforming the others in computation time.

For the dynamic case we extended the heuristic and metaheuristic methods from the static case to also include the calculation for the dynamic factors, i.e., the user demand. We, therefore, came up with a novel computation method for handling the dynamics occurring in conjunction with the user demand. Instead of discretizing time, as it is done in other works, we split up the user demand function into monotonically increasing and decreasing segments, which on the one hand resulted in faster computation times and on the other hand improved the accuracy of the computations.

In addition, to solve large-scale instances of the BBSS problem to optimality, we introduced a new simplified problem formulation that also complies with practical requirements. We allow only to transport full-vehicle loads of bikes from one station to another. Based on this problem formulation and an elaborated logic-based Benders decomposition, as well as a variant thereof, called branch-and-check, we have been able to solve instances up to 90 stations to proven optimality, which was not achieved in the literature before.

When solving the BSSPP, we introduced a novel approach that is not only applicable to the domain of bike sharing, but potentially also to other location-planning problems where large instance sizes need to be solved. Practical problem sizes of large cities, such as Vienna, are usually too big to be handled with traditional (meta)heuristic methods. Thus, instead of using a classical demand matrix, we applied a hierarchical clustering and determined aggregated demands for arcs on the respective cluster tree. As optimization technique, we came up with a multilevel refinement approach which can effectively use the hierarchically clustered input data by coarsening up the clustering tree until an instance size is reached which can be easily solved and then, propagate solution information downwards until a final solution to the original instance is obtained. Moreover, as shown in the case of the BSSPP, the proposed method has an exceptional scalability.

When approaching the DRPSC, we present an efficient route elimination algorithm which iteratively removes routes from a solution by maintaining a cleverly organized ejection pool such that object visits can be inserted into the remaining routes. Additional local search with smart neighborhoods complement the effectiveness of the algorithm. Moreover, we also considered soft time windows in a variant of the problem called DRPSC-STW. We found two efficient methods of computing arrival times for a given visit order of objects, so that the makespan is optimal in the case of soft time windows. We show that the problem is solvable in polynomial time by developing a linear-programming model. Furthermore, we developed a much faster hybrid heuristic which was able to compute the optimal solution in nearly all scenarios but is a much faster algorithm than the linear programming model. This hybrid heuristic embedded within a large neighborhood search showed exceptionally good results on the considered instance set. The hybrid heuristic could also be used in other problems as it combines an exact dynamic programming approach with a kind of bisection search. Moreover, we included proofs showing under which circumstances the hybrid heuristic yields proven optimal solutions.

6.1 Future Work

For the BBSS problem it would be promising to develop a (very) large neighborhood search (LNS) with the provided methods for the static problem formulation as ingredients. For instance, the VNS could be used to construct solutions and the MIP models could be used to optimize subproblems, i.e., some variables could be fixed and some could be optimized by the MIP. Furthermore, there exists also an approach for the static BBSS problem with constraint programming by Di Gaspero et al. [40, 41] which can possibly also be included in an LNS. This is also a viable option for the dynamic problem variant, to hybridize the already realized algorithms and check if results can be improved. Regarding the full-load route planning it would make sense to investigate an approach where the clustering part is done heuristically, in order to possibly achieve better scalability on larger instances.

For the BSSPP, several opportunities and research directions are possible. A POP-MUSIC [148] based approach could be a promising algorithm to be implemented for large-scale optimization. Furthermore, the solution evaluation is currently done exactly by solving an LP with many variables and constraints which restricts scalability. Instead, a heuristic approach for evaluating solutions could be developed, see also [12]. This would make the approach even more performant which could be useful to get more iterations of the algorithm and might improves the solution quality. Furthermore, there is still space for improvement regarding the extension heuristic. There, it would also be of interest to see how different extension techniques perform. Of course, it is important to solve instances for different cities and discuss the results with practitioners. Furthermore, this general and novel approach should definitely be also applied to other large-scale combinatorial optimization problems such as other facility location problems, the traveling salesman problem, and the vehicle routing problem.

Within the DRPSC there are many questions and problem variants that should also be considered for practice. In practice, it is often important to have a balanced workload for the members of the security staff. However, if the routes should be balanced within the objective function, it, nevertheless, makes no sense to artificially stretch routes such that they get balanced. The consideration of these aspects results in a bi-objective optimization problem. The objective function of the upper level is to optimize the deviation between the respective route lengths, and in the lower level, the route duration has to be minimized for each route such that routes and arrival times stay plausible. For the original problem with hard or soft time windows the developed approach based on the route elimination algorithm performs well, but a hybridization with some mixed integer linear programming technique might make sense for possibly further improving results. Additionally, we would like to derive dual bounds on the objective value. This might be achieved by considering certain relaxations of the underlying combinatorial optimization problem.

by some surrogate model of the problem or at least for smaller instance sizes.

Bibliography

[1] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3):75–102, 2002.

[2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, volume 2241 of *Lecture Notes in Computer Science*, pages 261–303. Springer Berlin Heidelberg, 2001.

[3] R. Aringhieri, M. Bruglieri, F. Malucelli, and M. Nonato. Metaheuristics for a vehicle routing problem on bipartite graphs with distance constraints. In *6th Metaheuristics International Conference*, pages 77–82, Vienna, Austria, 2005.

[4] N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001.

[5] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[6] R. Baldacci, A. Mingozzi, and R. Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(3):356–371, 2012.

[7] A. Baltz and A. Srivastav. Approximation algorithms for the euclidean bipartite TSP. *Operations Research Letters*, 33(4):403–410, 2005.

[8] R. Bellman. *Dynamic programming.* Dover Books on Computer Science. Dover Publications, 2013.

[9] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[10] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[11] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press Amsterdam, 2009.

[12] B. Biesinger, B. Hu, M. Stubenschrott, U. Ritzinger, and M. Prandtstetter. Optimizing charging station locations for electric car-sharing systems. In B. Hu and M. López-Ibáñez, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 10197 of *Lecture Notes in Computer Science*, pages 157–172. Springer International Publishing, 2017.

[13] C. Blum. Beam-ACO–hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, 32(6): 1565–1591, 2005.

[14] C. Blum. Beam-ACO for the longest common subsequence problem. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[15] C. Blum and G. R. Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, 2016.

[16] C. Blum and A. Roli. Hybrid metaheuristics: An introduction. In C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 1–30. Springer Berlin Heidelberg, 2008.

[17] C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels. *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2008.

[18] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[19] L. Caggiani, R. Camporeale, M. Ottomanelli, and W. Y. Szeto. A modeling framework for the dynamic management of free-floating bike-sharing systems. *Transportation Research Part C: Emerging Technologies*, 87:159–182, 2018.

[20] D. Chemla, F. Meunier, and R. W. Calvo. Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120–146, 2013.

[21] D. Chemla, F. Meunier, and R. W. Calvo. Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120–146, 2013.

[22] J. Chen, X. Chen, H. Jiang, S. Zhu, X. Li, and Z. Li. Determining the optimal layout design for public bicycle system within the attractive scope of a metro station. *Mathematical Problems in Engineering*, 2015, 2015. Article ID 456013.

[23] Q. Chen and T. Sun. A model for the layout of bike stations in public bike-sharing systems. *Journal of Advanced Transportation*, 49(8):884–900, 2015.

[24] C.-B. Cheng and C.-P. Mao. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, 46 (9–10):1225–1235, 2007.

[25] B. V. Cherkassky and A. V. Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[26] A. A. Ciré, E. Çoban, and J. N. Hooker. Logic-based benders decomposition for planning and scheduling: a computational analysis. *The Knowledge Engineering Review*, 31(5):440–451, 2016.

[27] G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[28] C. Contardo, C. Morency, and L.-M. Rousseau. Balancing a dynamic public bike-sharing system. Technical Report CIRRELT-2012-09, Université de Montréal, Montréal, Canada, March 2012. URL `https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2012-09.pdf`.

[29] I. Contreras. Hub location problems. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, pages 311–344. Springer International Publishing, 2015.

[30] W. Cook. Concorde TSP solver. `http://www.math.uwaterloo.ca/tsp/concorde/`, 2011. [Online; accessed 06-May-2015].

[31] C. Cotta, M. Sevaux, and K. Sörensen. *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 2008.

[32] R. F. Da Silva and S. Urrutia. A general VNS heuristic for the traveling salesman problem with time windows. *Discrete Optimization*, 7(4):203–211, 2010.

[33] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *New York*, 1951.

[34] T. Davidović, P. Hansen, and N. Mladenović. Variable neighborhood search for multiprocessor scheduling problem with communication delays. In *Proceedings of 4th Metaheuristics International Conference*, volume 4, pages 737–741, 2001.

[35] L. Davis. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 1991.

[36] C. Defryn, K. Sörensen, and W. Dullaert. Integrating partner objectives in horizontal logistics optimisation models. *Omega*, 82:1–12, 2019.

[37] M. Dell'Amico, E. Hadjicostantinou, M. Iori, and S. Novellani. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19, 2014.

[38] P. DeMaio. Bike-sharing: History, impacts, models of provision, and future. *Journal of Public Transportation*, 12(4):41–56, 2009.

[39] G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column Generation*. Springer US, 2005.

[40] L. Di Gaspero, A. Rendl, and T. Urli. A hybrid ACO+CP for balancing bicycle sharing systems. In M. J. Blesa, C. Blum, P. Festa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 7919 of *Lecture Notes in Computer Science*, pages 198–212. Springer Berlin Heidelberg, 2013.

[41] L. Di Gaspero, A. Rendl, and T. Urli. Constraint-based approaches for balancing bike sharing systems. In C. Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 758–773. Springer Berlin Heidelberg, 2013.

[42] M. Dorigo and M. Birattari. Ant colony optimization. In C. Sammut and G. I. Webb, editors, *Encyclopedia of machine learning*, pages 36–39. Springer US, 2011.

[43] M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2–3):243–278, 2005.

[44] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

[45] C. Duin and S. Voß. Steiner tree heuristics — a survey. In H. Dyckhoff, U. Derigs, M. Salomon, and H. C. Tijms, editors, *Operations Research Proceedings 1993*, pages 485–496. Springer Berlin Heidelberg, 1994.

[46] C. Duin and S. Voß. The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs. *Networks*, 34(3):181–191, 1999.

[47] G. Erdoğan, G. Laporte, and R. W. Calvo. The static bicycle relocation problem with demand intervals. *European Journal of Operational Research*, 238(2):451–457, 2014.

[48] K. Fagerholt. Ship scheduling with soft time windows: An optimisation based approach. *European Journal of Operational Research*, 131(3):559–571, 2001.

[49] R. Z. Farahani, M. Hekmatfar, A. B. Arabani, and E. Nikbakhsh. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, 64(4):1096–1109, 2013.

[50] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[51] I. A. Forma, T. Raviv, and M. Tzur. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation Research Part B: Methodological*, 71:230–247, 2015.

[52] I. Frade and A. Ribeiro. Bike-sharing stations: A maximal covering location approach. *Transportation Research Part A: Policy and Practice*, 82:216–227, 2015.

[53] C. Fricker and N. Gast. Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. *EURO Journal on Transportation and Logistics*, 5(3):261–291, 2016.

[54] L. M. Gambardella, E. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, editors, *New Ideas in Optimization*, chapter 5, pages 63–76. McGraw-Hill, UK, 1999.

[55] A. Gauthier, C. Hughes, C. Kost, S. Li, C. Linke, S. Lotshaw, J. Mason, C. Pardo, C. Rasore, B. Schroeder, and X. T. no. *The Bike-share Planning Guide*. Institue for Transportation & Development Policy, 2013.

[56] D. Gavalas, C. Konstantopoulos, and G. Pantziou. Design and management of vehicle-sharing systems: a survey of algorithmic approaches. In M. S. Obaidat and P. Nicopolitidis, editors, *Smart Cities and Homes: Key Enabling Technologies*, chapter 13, pages 261–289. Elsevier, 2016.

[57] M. Gendreau. An introduction to tabu search. In F. W. Glover and G. A. Kochenberger, editors, *Handbook of metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 2, pages 37–54. Springer US, 2003.

[58] M. Gendreau and J.-Y. Potvin. *Handbook of metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*. Springer International Publishing, 2019.

[59] M. Gendreau, J. Nossack, and E. Pesch. Mathematical formulations for a 1-full-truckload pickup-and-delivery problem. *European Journal of Operational Research*, 242(3):1008–1016, 2015.

[60] F. Glover and M. Laguna. Tabu search. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of combinatorial optimization*, pages 2093–2229. Springer US, 1998.

[61] F. W. Glover and G. A. Kochenberger. *Handbook of metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Springer US, 2003.

[62] Y. Han, E. Côme, and L. Oukhellou. Towards bicycle demand prediction of large-scale bicycle sharing system. In *Transportation Research Board 93rd Annual Meeting*, pages 1–17, 2014.

[63] I. Harjunkoski and I. E. Grossmann. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers & Chemical Engineering*, 26(11):1533–1552, 2002.

[64] H. Hashimoto, T. Ibaraki, S. Imahori, and M. Yagiura. The vehicle routing problem with flexible time windows and traveling times. *Discrete Applied Mathematics*, 154 (16):2271–2290, 2006.

[65] H. Hernández-Pérez and J.-J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, 2004.

[66] H. Hernández-Pérez and J.-J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2): 245–255, 2004.

[67] H. Hernández-Pérez and J.-J. Salazar-González. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks*, 50(4): 258–272, 2007.

[68] H. Hernández-Pérez, I. Rodríguez-Martín, and J. J. Salazar-González. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36(5):1639–1645, 2009.

[69] H. Hernández-Pérez, J. J. Salazar-González, and B. Santos-Hernández. Heuristic algorithm for the split-demand one-commodity pickup-and-delivery travelling salesman problem. *Computers & Operations Research*, 97:1–17, 2018.

[70] S. C. Ho and W. Szeto. Solving a static repositioning problem in bike-sharing systems using iterated tabu search. *Transportation Research Part E: Logistics and Transportation Review*, 69:180–198, 2014.

[71] J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[72] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[73] S.-R. Hu and C.-T. Liu. An optimal location model for a bicycle sharing program with truck dispatching consideration. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1775–1780. IEEE, 2014.

[74] T. Ibaraki and Y. Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82, 1994.

[75] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39(2):206–232, 2005.

[76] I. Ioachim, S. Gélinas, F. Soumis, and J. Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204, 1998.

[77] R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983.

[78] R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems: erratum. *Operations Research Letters*, 5(4):215–216, 1986.

[79] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[80] L. G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979.

[81] B.-M. Kim, C. Kloimüllner, and G. R. Raidl. Efficient consideration of soft time windows in a large neighborhood search for the districting and routing problem for security control. In B. Hu and M. López-Ibáñez, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 10197 of *Lecture Notes in Computer Science*, pages 91–107. Springer International Publishing, 2017.

[82] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[83] C. Kloimüllner and G. R. Raidl. Full-load route planning for balancing bike sharing systems by logic-based Benders decomposition. *Networks*, 69(3):270–289, 2017.

[84] C. Kloimüllner and G. R. Raidl. Hierarchical clustering and multilevel refinement for the bike-sharing station planning problem. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors, *Learning and Intelligent Optimization*, volume 10556 of *Lecture Notes in Computer Science*, pages 150–165. Springer International Publishing, 2017.

[85] C. Kloimüllner and G. R. Raidl. A novel approach for solving large-scale instances in the bike sharing station planning problem. Technical report, Institute of Logic and Computation, TU Wien, 2019. submitted to 13th Learning and Intelligent Optimization Conference.

[86] C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: An approach for the dynamic case. In C. Blum and G. Ochoa, editors, *Evolutionary Computation in Combinatorial Optimisation*, volume 8600 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2014.

[87] C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl. A cluster-first route-second approach for balancing bicycle sharing systems. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2015*, volume 9520 of *Lecture Notes in Computer Science*, pages 439–446. Springer International Publishing, 2015.

[88] E. Klotz and A. M. Newman. Practical guidelines for solving difficult linear programs. *Surveys in Operations Research and Management Science*, 18(1):1–17, 2013.

[89] M.-C. Lai, H. Sohn, and D. Bricker. A hybrid Benders/genetic algorithm for vehicle routing and scheduling problem. *International Journal of Industrial Engineering: Theory, Applications and Practice*, 19(1), 2012.

[90] G. Laporte, F. Meunier, and R. W. Calvo. Shared mobility systems. *4OR*, 13(4): 341–360, 2015.

[91] J.-R. Lin and T.-H. Yang. Strategic design of public bicycle sharing systems with service level constraints. *Transportation Research Part E: Logistics and Transportation Review*, 47(2):284–294, 2011.

[92] J.-R. Lin, T.-H. Yang, and Y.-C. Chang. A hub location inventory model for bicycle sharing system design: Formulation and solution. *Computers & Industrial Engineering*, 65(1):77–86, 2013.

[93] M. López-Ibáñez and C. Blum. Beam-ACO for the travelling salesman problem with time windows. *Computers & Operations Research*, 37(9):1570–1583, 2010.

[94] M. López-Ibáñez, C. Blum, J. W. Ohlmann, and B. W. Thomas. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing*, 13(9):3806–3815, 2013.

[95] L. M. Martinez, L. Caetano, T. Eiró, and F. Cruz. An optimisation algorithm to establish the location of stations of a mixed fleet biking system: An application to the city of Lisbon. *Procedia – Social and Behavioral Sciences*, 54:513–524, 2012.

[96] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[97] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[98] N. Mladenović, R. Todosijević, and D. Urošević. An efficient GVNS for solving traveling salesman problem with time windows. *Electronic Notes in Discrete Mathematics*, 39:83–90, 2012.

[99] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. A. Kochenberger, editors, *Handbook of metaheuristics*, pages 105–144. Springer US, 2003.

[100] P. Moscato, C. Cotta, and A. Mendes. Memetic algorithms. In G. C. O. V. Babu, editor, *New optimization techniques in engineering*, volume 141 of *Studies in Fuzziness and Soft Computing*, pages 53–85. Springer Berlin Heidelberg, 2004.

[101] Y. Nagata and O. Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5):333–338, 2009.

[102] Y. Nagata, O. Bräysy, and W. Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4):724–737, 2010.

[103] J. W. Ohlmann and B. W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1):80–90, 2007.

[104] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[105] W. Ouyang, C. W. Yu, K.-M. Yu, K.-J. Lin, J.-H. Yu, H.-W. Chang, L.-L. Tai, and C.-H. Lin. Station decision problem in bicycle ad hoc networks. In *9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pages 876–881. IEEE, 2012.

[106] W. Ouyang, C. W. Yu, K.-M. Yu, K.-J. Lin, H.-W. Chang, H.-N. Hsieh, L.-L. Tai, and C.-H. Lin. Solving station decision problem in bicycle ad hoc networks. *International Journal of Ad Hoc and Ubiquitous Computing*, 16(2):93–102, 2014.

[107] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *The International Journal Of Production Research*, 26(1):35–62, 1988.

[108] A. Pal and Y. Zhang. Free-floating bike sharing: Solving real-life large-scale static rebalancing problems. *Transportation Research Part C: Emerging Technologies*, 80: 92–116, 2017.

[109] P. Papazek, G. R. Raidl, M. Rainer-Harbach, and B. Hu. A PILOT/VND/GRASP hybrid for the static balancing of public bicycle sharing systems. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2013*, volume 8111 of *Lecture Notes in Computer Science*, pages 372–379. Springer Berlin Heidelberg, 2013.

[110] P. Papazek, C. Kloimüllner, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: An analysis of path relinking and recombination within a GRASP hybrid.

In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 792–801. Springer International Publishing, 2014.

[111] J. Pfrommer, J. Warrington, G. Schildbach, and M. Morari. Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1567–1578, 2014.

[112] S. Pirkwieser and G. R. Raidl. A variable neighborhood search for the periodic vehicle routing problem with time windows. In C. Prodhon et al., editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France, 2008.

[113] D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter 13, pages 399–419. Springer US, 2010.

[114] E. Prescott-Gagnon, G. Desaulniers, and L.-M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204, 2009.

[115] C. Prins, P. Lacomme, and C. Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.

[116] M. Prischink. Metaheuristics for the districting and routing problem for security control. Master's thesis, TU Wien, Institute of Computer Graphics and Algorithms, May 2016. supervised by G. Raidl, B. Biesinger, and C. Kloimüllner.

[117] M. Prischink, C. Kloimüllner, B. Biesinger, and G. R. Raidl. Districting and routing for security control. In M. J. Blesa, C. Blum, A. Cangelosi, V. Cutello, A. D. Nuovo, M. Pavone, and E.-G. Talbi, editors, *Hybrid Metaheuristics*, volume 9668 of *Lecture Notes in Computer Science*, pages 87–103. Springer International Publishing, 2016.

[118] R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

[119] G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida, M. J. Blesa Aguilera, C. Blum, J. M. Moreno Vega, M. Pérez Pérez, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, pages 1–12. Springer Berlin Heidelberg, 2006.

[120] G. R. Raidl. Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244:66–76, 2015.

[121] G. R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, editors, *Hybrid metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer-Verlag Berlin Heidelberg, 2008.

[122] G. R. Raidl, B. Hu, M. Rainer-Harbach, and P. Papazek. Balancing bicycle sharing systems: Improving a VNS by efficiently determining optimal loading operations. In M. J. Blesa, C. Blum, P. Festa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 7919 of *Lecture Notes in Computer Science*, pages 130–143. Springer Berlin Heidelberg, 2013.

[123] G. R. Raidl, T. Baumhauer, and B. Hu. Speeding up logic-based benders' decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem. In M. J. Blesa, C. Blum, and S. Voß, editors, *Hybrid Metaheuristics*, volume 8457 of *Lecture Notes in Computer Science*, pages 183–197. Springer International Publishing, 2014.

[124] M. Rainer-Harbach, P. Papazek, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: A variable neighborhood search approach. In M. Middendorf and C. Blum, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7832 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, 2013.

[125] M. Rainer-Harbach, P. Papazek, B. Hu, G. R. Raidl, and C. Kloimüllner. PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems. *Journal of Global Optimization*, 63(3):597–629, 2015.

[126] T. Raviv, M. Tzur, and I. A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229, 2013.

[127] C. Reeves. Genetic algorithms. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter 5, pages 109–139. Springer US, 2010.

[128] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 219–249. Springer US, 2003.

[129] C. S. ReVelle and H. A. Eiselt. Location analysis: A synthesis and survey. *European Journal of Operational Research*, 165(1):1–19, 2005.

[130] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming.* Elsevier, 2006.

[131] C. Rudloff and B. Lackner. Modeling demand for bicycle sharing system – neighboring stations as a source for demand and a reason for structural breaks. Technical report, Austrian Institute of Technology, Vienna, Austria, 2013.

[132] C. Rudloff and B. Lackner. Modeling demand for bikesharing systems. *Transportation Research Record: Journal of the Transportation Research Board*, 2430:1–11, 2014.

[133] A. s Frank, E. Triesch, B. Korte, and J. Vygen. On the bipartite travelling salesman problem. Technical Report 98866-OR, Research Institute for Discrete Mathematics, 1998.

[134] G. K. Saharidis, A. Fragkogios, and E. Zygouri. A multi-periodic optimization modeling approach for the establishment of a bike sharing network: a case study of the city of athens. In S. I. Ao, O. Castillo, C. Douglas, D. D. Feng, and J.-A. Lee, editors, *Proceedings of The International MultiConference of Engineers and Computer Scientists 2014*, volume 2210 of *Lecture Notes in Engineering and Computer Science*, pages 1226–1231. Newswood Limited, 2014.

[135] J.-J. Salazar-González and B. Santos-Hernández. The split-demand one-commodity pickup-and-delivery travelling salesman problem. *Transportation Research Part B: Methodological*, 75:58–73, 2015.

[136] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[137] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[138] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1998.

[139] B. Schroeder. *Bicycle Sharing 101: Getting the Wheels Turning.* Moonshine Media, 2014.

[140] J. Schuijbroek, R. C. Hampshire, and W.-J. van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257(3):992–1004, 2017.

[141] M. Sevaux and K. Sörensen. Hamiltonian paths in large clustered routing problems. In *Proceedings of the EU/MEeting 2008 workshop on metaheuristics for logistics and vehicle routing, EU/ME*, volume 8, pages 411–417, 2008.

[142] A. Shurbevski, H. Nagamochi, and Y. Karuno. Approximating the bipartite TSP and its biased generalization. In S. P. Pal and K. Sadakane, editors, *Algorithms and Computation*, volume 8344 of *Lecture Notes in Computer Science*, pages 56–67. Springer International Publishing, 2014.

[143] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[144] K. Sörensen and N. Vergeylen. The bike request scheduling problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2015*, volume 9520 of *Lecture Notes in Computer Science*, pages 294–301. Springer International Publishing, 2015.

[145] A. Srivastav, H. Schroeter, and C. Michel. Approximation algorithms for pick-and-place robots. *Annals of Operations Research*, 107(1–4):321–338, 2001.

[146] M. Straub, C. Rudloff, A. Graser, C. Kloimüllner, G. R. Raidl, M. Pajones, and F. Beyer. Semi-automated location planning for urban bike-sharing systems. In *Proceedings of the 7th Transport Research Arena (TRA 2018)*, pages 1–10, Vienna, Austria, 2018.

[147] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[148] É. D. Taillard and S. Voss. Popmusic — partial optimization metaheuristic under special intensification conditions. In *Essays and surveys in metaheuristics*, volume 15 of *Operations Research/Computer Science Interfaces Series*, chapter 27, pages 613–629. Springer US, 2002.

[149] E.-G. Talbi, editor. *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 2013.

[150] E. S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming – CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin Heidelberg, 2001.

[151] C.-K. Ting and X.-L. Liao. The selective pickup and delivery problem: Formulation and a memetic algorithm. *International Journal of Production Economics*, 141(1): 199–211, 2013.

[152] P. J. Van Laarhoven and E. H. Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, volume 37 of *Mathematics and Its Applications*, chapter 2, pages 7–15. Springer Netherlands, 1987.

[153] N. A. Vergeylen. *A novel approach to city bicycle repositioning.* PhD thesis, University of Antwerp, Faculty of Applied Economics, April 2018. supervised by K. Sörensen.

[154] N. A. Vergeylen, K. Sörensen, and D. P. Cuervo. Solution space analysis for the bike request scheduling problem. Technical Report 2018-005, University of Antwerp,

Antwerp, Belgium, February 2018. URL `https://repository.uantwerpen.be/docman/irua/eda2ca/149166.pdf`.

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

[156] P. Vogel, B. A. Neumann Saavedra, and D. C. Mattfeld. A hybrid metaheuristic to solve the resource allocation problem in bike sharing systems. In M. J. Blesa, C. Blum, and S. Voß, editors, *Hybrid Metaheuristics*, volume 8457 of *Lecture Notes in Computer Science*, pages 16–29. Springer International Publishing, 2014.

[157] S. Voß, A. Fink, and C. Duin. Looking ahead with the PILOT method. *Annals of Operations Research*, 136:285–302, 2005.

[158] C. Walshaw. A multilevel approach to the travelling salesman problem. *Operations Research*, 50(5):862–877, 2002.

[159] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, 131(1–4):325–372, 2004.

[160] L. A. Wolsey. *Integer Programming.* Wiley, 1998.

[161] T.-H. Yang, J.-R. Lin, and Y.-C. Chang. Strategic design of public bicycle sharing systems incorporating with bicycle stocks considerations. In *The 40th International Conference on Computers Indutrial Engineering*, pages 1–6. IEEE, 2010.

Dipl.-Ing. Christian Kloimüllner

Born on February 8th, 1987
Hofwiese 13, 3204 Kirchberg/Pielach
christian.kloimuellner@gmail.com
(+43) 680 21 29 655

WORK EXPERIENCE

RISE, *Software Engineer* — Dec 2010 - Today

Full-Stack developer in different projects. The majority of the work done was implemented in Java. Modern development stack includes Spring Boot in the backend and Angular in the frontend with a RESTful API.

TU WIEN, *Researcher* — Feb 2013 - Aug 2017

PhD student and researcher at the Vienna University of Technology. Research interests are in general combinatorial optimization problems and (meta)heuristics as well as exact algorithms for solving them.

B.S.O. EDV- UND BETRIEBSBERATUNG GESMBH, *Trainee* — Jul 2004

Responsible for various office activities. The main activity was to work with financial documents of the customers, analyzing and archiving them. Moreover, a further responsiblity was to help the IT support of the company.

ING. HESS GMBH LÜFTUNGSTECHNIK, *Trainee* — Jul 2002

Assistant of the CTO in this company which is based in St. Pölten. Various responsibilities including customer contact and customer coordination. Responsible for IT, document archiving, and answering phone calls.

EDUCATION

DR.TECHN. ENVIRONMENTAL INFORMATICS — Feb 2013 - Today
Vienna University of Technology

The conceptual framework of the doctoral college consists of two interdependent layers, namely enabling technologies and environmental solutions.

PhD thesis: Algorithmic Approaches for Optimization Problems in Bike Sharing and Security Control

DIPL.-ING. COMPUTATIONAL INTELLIGENCE — Oct 2009 - Jun 2012
Vienna University of Technology

The objective of the work in the field of *Computational Intelligence* is both to understand the principles that make intelligent behavior possible and to develop methods therefore.

Master thesis: Visualisation and Graphical Editing of Answer Sets: The Kara System

BSC BUSINESS INFORMATICS — Oct 2008 - Jan 2013
Vienna University of Technology

Business Informatics is concerned with information, knowledge and information processing in organisations and in society. It thus lies at the interface between people, organisations and IT. The teaching covers information and communications systems in business and in society.

BSC INFORMATIKMANAGEMENT — Oct 2008 - Mar 2013
University of Vienna

Intramural study which was held by the University of Vienna and the Vienna University of Technology. The goal is to teach the students technically and scientifically in pedagogical, didactic, and subject-didactic skills.

BSC SOFTWARE & INFORMATION ENGINEERING — Oct 2007 - Jul 2009
Vienna University of Technology

Software engineering is concerned with the development of software, from analysis and design to implementation, quality assurance and software maintenance. Information engineering covers the generation, collection, processing, distribution and presentation of information.

PUBLICATIONS

2018

M. Straub, C. Rudloff, A. Graser, C. Kloimüllner, G. R. Raidl, M. Pajones, and F. Beyer. Semi-automated location planning for urban bike-sharing systems. In *Proceedings of the 7th Transport Research Arena (TRA 2018)*, pages 1–10, Vienna, Austria, 2018

2017

C. Kloimüllner and G. R. Raidl. Full-load route planning for balancing bike sharing systems by logic-based Benders decomposition. *Networks*, 69(3):270–289, 2017

C. Kloimüllner and G. R. Raidl. Hierarchical clustering and multilevel refinement for the bike-sharing station planning problem. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors, *Learning and Intelligent Optimization*, volume 10556 of *Lecture Notes in Computer Science*, pages 150–165. Springer International Publishing, 2017

B.-M. Kim, C. Kloimüllner, and G. R. Raidl. Efficient consideration of soft time windows in a large neighborhood search for the districting and routing problem for security control. In B. Hu and M. López-Ibáñez, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 10197 of *Lecture Notes in Computer Science*, pages 91–107. Springer International Publishing, 2017

2016

M. Prischink, C. Kloimüllner, B. Biesinger, and G. R. Raidl. Districting and routing for security control. In M. J. Blesa, C. Blum, A. Cangelosi, V. Cutello, A. D. Nuovo, M. Pavone, and E.-G. Talbi, editors, *Hybrid Metaheuristics*, volume 9668 of *Lecture Notes in Computer Science*, pages 87–103. Springer International Publishing, 2016

2015

M. Rainer-Harbach, P. Papazek, B. Hu, G. R. Raidl, and C. Kloimüllner. PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems. *Journal of Global Optimization*, 63(3):597–629, 2015

C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl. A cluster-first route-second approach for balancing bicycle sharing systems. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2015*, volume 9520 of *Lecture Notes in Computer Science*, pages 439–446. Springer International Publishing, 2015

2014

P. Papazek, C. Kloimüllner, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: An analysis of path relinking and recombination within a GRASP hybrid. In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 792–801. Springer International Publishing, 2014

C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: An approach for the dynamic case. In C. Blum and G. Ochoa, editors, *Evolutionary Computation in Combinatorial Optimisation*, volume 8600 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2014

2012

C. Kloimüllner. Visualisation and graphical editing of answer sets: The kara system. Master's thesis, Vienna University of Technology, Institute of Information Systems, May 2012. supervised by H. Tompits and J. Puehrer

2011

C. Kloimüllner, J. Oetsch, J. Puehrer, and H. Tompits. Kara: A system for visualising and visual editing of interpretations for answer-set programs. In *19th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011) and 25th Workshop on Logic Programming (WLP 2011)*, pages 152–164. INFSYS Research Report, 2011

SKILLS & INTERESTS

🇺🇸	Cambridge BEC (B2)
🇩🇪	Native Speaker
🇨🇿	Conversational Level

- Java, C, C++, Answer-Set Programming, Javascript, Ruby
- XML, XSL, XSLT, XPATH, HTML
- J2EE, Seam, Spring, div. JSF Bibliotheken, Hibernate
- JQuery, Angular, react, Apache Shiro
- MongoDB, PostgreSQL, Oracle

- SVN, git
- make, maven, buildr
- Tomcat, JBoss, GlassFish
- Algorithms and Combinatorial Optimization (e.g., Vehicle Routing), Semantic Web
- Machine Learning, RESTful API/Services

Tennis,	Austrian Champion in 2010, former international ranking
Soccer,	Youth Vice Champion in Lower Austria
Running,	Attendee and finisher in many half marathons and one time Vienna City Marathon Finisher